Módulo 6

Programação WEB

Lição 11

Segurança na WEB

Versão 1.0 - Nov/2007

Autor
Daniel Villafuerte

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
Allan Souza Nunes
Allan Wojcik da Silva
Angelo de Oliveira
Aurélio Soares Neto
Bruno da Silva Bonfim
Carlos Fernando Gonçalves
Denis Mitsuo Nakasaki
Emanoel Tadeu da Silva Freitas
Felipe Gaúcho
Jacqueline Susann Barbosa
João Vianney Barrozo Costa
Luciana Rocha de Oliveira
Luiz Fernandes de Oliveira Junior
Marco Aurélio Martins Bessa
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauro Cardoso Mortoni
Paulo Oliveira Sampaio Reis
Pedro Henrique Pereira de Andrade
Ronie Dotzlaw
Sergio Terzella
Thiago Magela Rodrigues Dias
Vanessa dos Santos Almeida
Wagner Eliezer Roncoletta

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **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

Este módulo de programação WEB não estaria completo sem uma discussão sobre como deixar seguras as aplicações desenvolvidas. Características atraentes, grande usabilidade e boa manutenção parecem pequenos se a aplicação falhar em confiabilidade junto aos clientes WEB.

Ao final desta lição, o estudante será capaz de:

- Obter segurança na comunicação entre servidor e cliente usando *Session Socket Layer*
- Aprender medidas para evitar as maiores falhas em aplicações WEB

2. SSL

O SSL transformou-se de fato no padrão de fato em segurança na comunicação entre clientes e servidores. Representa uma camada de Socket Segura. É uma camada de protocolo que se interpõe entre a camada padrão de TCP/IP e a camada acima, os protocolos de aplicação como o HTTP. Permite ao servidor se autenticar no cliente e depois disso codifica o canal de comunicação.

Uma discussão completa sobre os mecanismos da SSL está fora do escopo deste material. A finalidade é entender que o uso desta tecnologia garante um canal seguro entre o servidor e o cliente. Confidência é preservada mais adiante automaticamente devido à medida em que se detecte a parte incluída como parte do SSL.

2.1. Configurando o SSL nas aplicações

Para apreciar os beneficios da SSL nas aplicações, precisamos configurar o ambiente em que o servidor está rodando para que aceite conexões com SSL. Diferentes contêineres de *servlets* possuem diferentes formas de configuração. Nesta lição aprenderemos como configurar o *Sun Application Server* 8.1.

2.1.1. Certificados

Um das primeiras coisas que precisamos configurar em nosso servidor para comunicações baseadas em SSL é um certificado de segurança válido. Pense em um certificado como um passaporte que identifica o proprietário e contém outra informações importantes. Certificados normalmente são emitidos através de Autoridades Certificadoras (*Certification Authorities* ou CA). Uma CA atua como um escritório para a emissão de passaportes pois valida a identidade do dono do certificado e os sinais do certificado de forma que este não possa ser forjado ou falsificado.

Há várias Autoridades Certificadoras famosas. Uma das mais populares é a *Verisign*. É decisão do administrador qual CA utilizar como servidora de um certificado.

Na ausência de um certificado de uma CA confiável, um certificado temporário pode ser criado utilizando as ferramentas incluídas dentro da Java SDK. Observe, porém, que clientes perspicazes não continuam com transações confidenciais uma vez que descobrem que o certificado utilizado é criado por nós mesmos.

2.1.2. Gerando o Certificado com a Chave Particular

É mais fácil executar o seguinte conjunto de operações no mesmo diretório que o servidor utiliza para armazenar seus certificados. Este pode ser encontrado na pasta %APP_SERVER_HOME %/domains/domain1/config.

No diretório especificado, execute a seguinte a linha de comando:

```
keytool -genkey -alias keyAlias
-keyalg RSA -keypass keypassword
-storepass storepassword
-keystore keystore.jks
```

- ***keyAlias*** – o apelido ou ID que este certificado utilizará.
- **keypassword** – a senha para a chave particular que será utilizada na encriptação.
- **storepassword** – a senha para o keystore.

Pode ser um pouco confuso compreender a necessidade de se ter duas senhas na criação de um novo certificado. Porém, lembremos de que todas as entradas de chave residem no que é chamado um *keystore*. Um *keystore* pode armazenar uma ou mais chaves. *keypassword* indica a senha da chave particular que o nosso certificado usará, enquanto *storepassword* indica a senha do *keystore* que conterá a chave. O diretório em que estamos operando já tem um arquivo *keystore*, entretanto, com uma senha existente. Assim precisaremos fixar um novo *storepass*.

Esta senha pode ser mudada mais tarde, usando o aplicativo *keytool*:

```
keytool -keystore keystore.jks -storepass newPassword
```

2.1.3. Criando um Certificado

Depois de gerada a chave que será usada pelo certificado, podemos criar o próprio arquivo de certificado:

```
keytool -export -alias keyAlias
-storepass storepassword
-file certificateFileName
-keystore keystore.jks
```

A instrução anterior utiliza o aplicativo *keytool* para gerar o arquivo de certificado que usa a chave particular indicada por *keyAlias* que está contida no *keystore*.

2.1.4. Gerenciando o Certificado

Para o servidor de aplicação reconhecer o certificado criado há pouco, precisamos adicioná-lo este na lista de certificados confiáveis. O servidor contém um arquivo nomeado que *cacerts.jks* que contém os certificados confiáveis. Podemos adicionar nosso arquivo usando o aplicativo *keytool*:

```
keytool -import -v -trustcacerts
-alias keyAlias
-file certificateFileName
-keystore cacerts.jks
-keypass keypassword
```

Será solicitado a senha do *Application Server*.

2.1.5. Criando segurança com um HTTP *listener*

Depois que criamos um certificado e o registramos com sucesso para uso no servidor de aplicação, podemos criar um HTTP *listener* que fará uso do certificado. Dessa forma, o servidor de aplicações poderá ser usado para comunicações seguras.

Para fazer isto, primeiro crie uma entrada no Console de Administração do *Sun Application Server*. Nas opções a esquerda expanda a aba nomeada *Configuration* e expanda a opção HTTP *Service*:

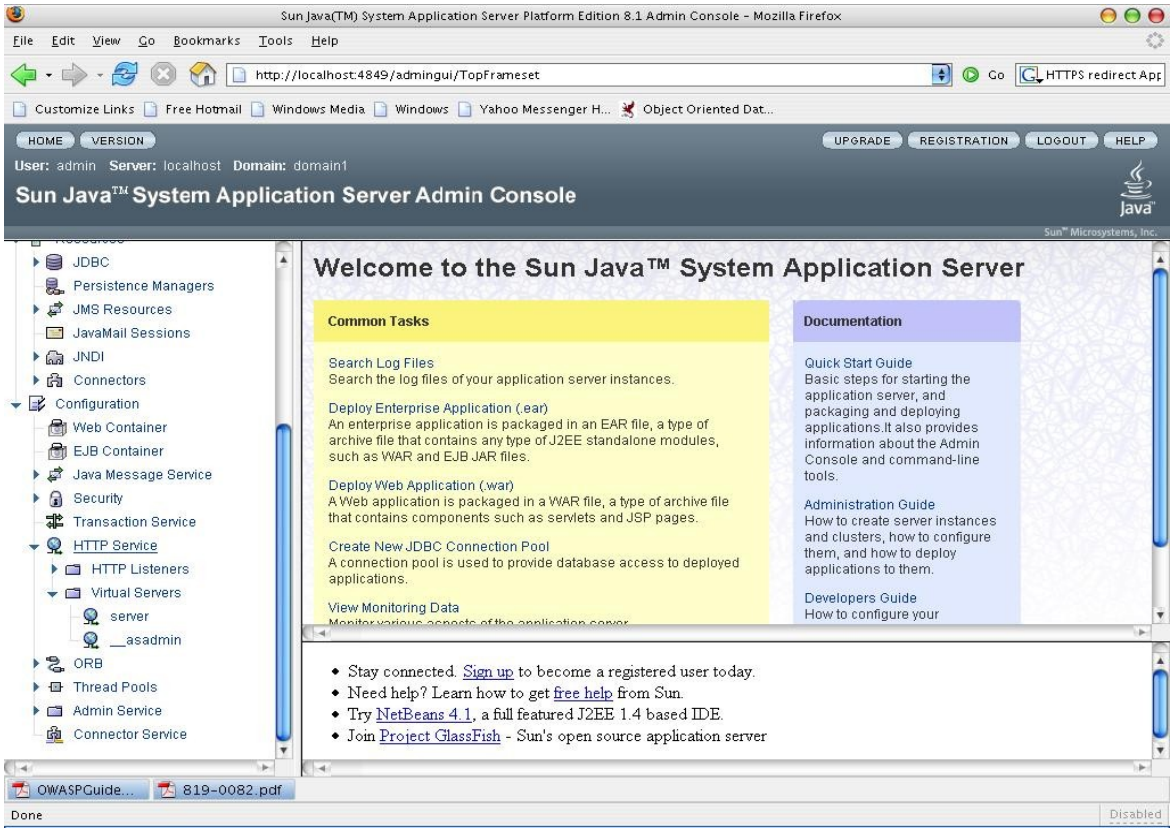

Figura 1 - Console de Administração

A seguir, selecione a opção *HTTP Listeners* e, no painel à direita, pressione o botão *New*.

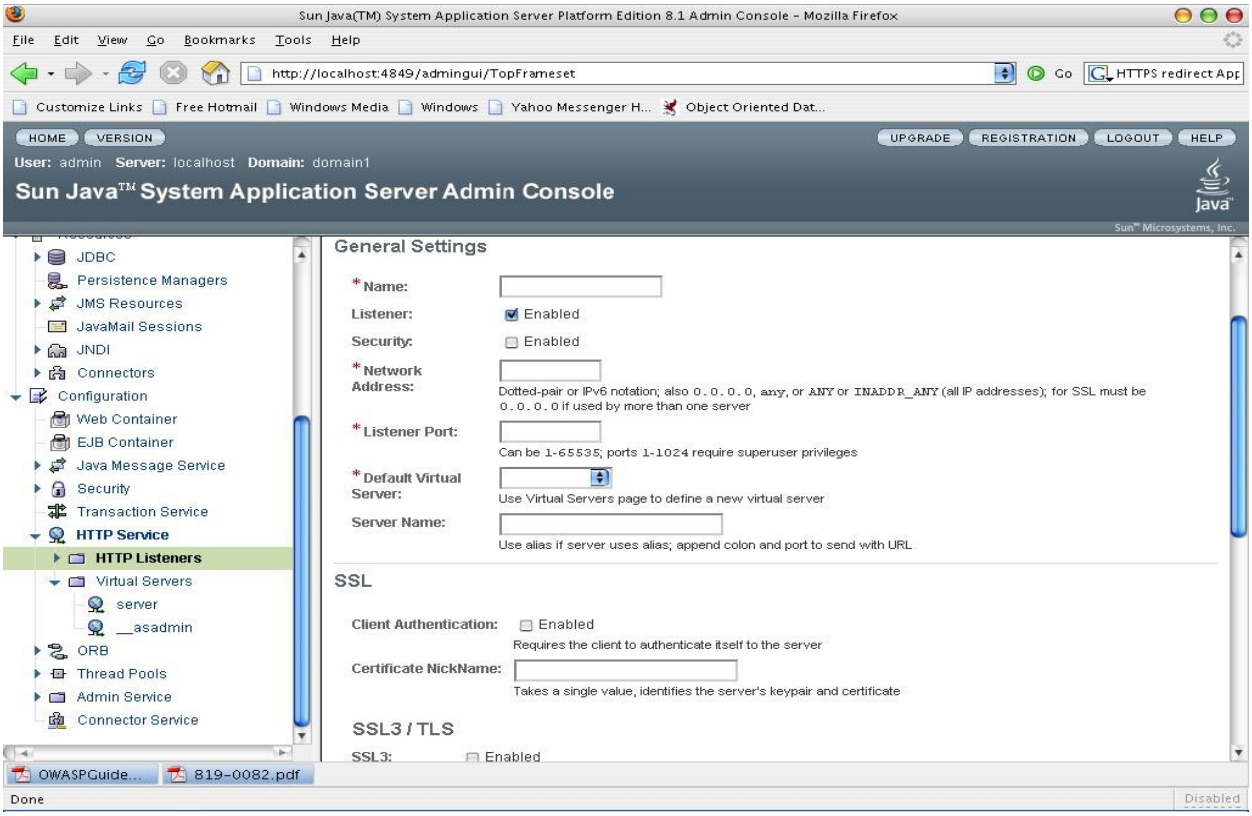

Figura 2 - Criando um HTTP Listener

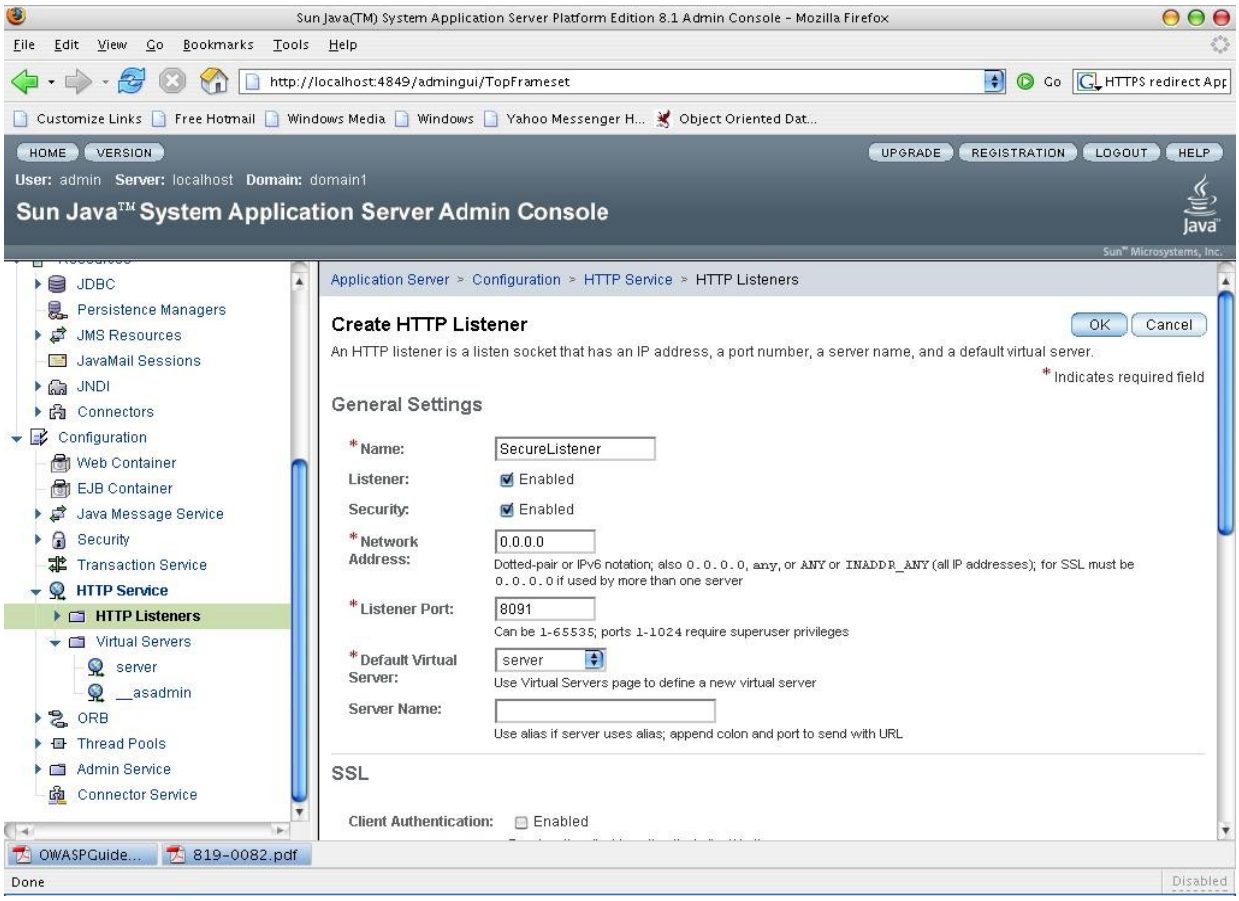

Figura 3 - Mudando as configurações gerais

As telas anteriores são o resultado do preenchimento com novos valores. Os valores que podem mudar são o nome do *listener* e a porta que serão utilizados de acordo com as preferências do administrador.

Reinicie o servidor e a configuração nova poderá ser utilizada acessando o endereço:

```
https://serverAddress:listenerPort/index.html
```

Com os dados fornecidos neste exemplo, o endereço seria:

```
https://localhost:8091/index.html
```

Para utilizar comunicação segura entre cliente e servidor, redirecione o usuário para a porta do *listener* seguro e desta forma obter o acesso a aplicação.

3. Falhas de Segurança em Aplicações WEB

A *Open Web Application Security Project* (Segurança de Projeto Aberto de Aplicação WEB), ou simplesmente OWASP é um projeto *open-source* criado por uma entidade sem fins lucrativos que se propõe a achar causas da não segurança em softwares e encontrar soluções. Vejamos a seguir, uma lista com tópicos de falhas de segurança em aplicações WEB. Recomenda-se que qualquer aplicação WEB seja criada protegida dessas falhas como um padrão mínimo de segurança.

Olhemos esta lista, e vejamos como podemos proteger nossas aplicações.

3.1. Entrada inválida

Todas as aplicações WEB recebem dados através de uma requisição HTTP realizada pelo usuário e fazem uso desses dados informados para executar suas operações. *Hackers* podem manipular qualquer parte da requisição (uma consulta, informação em *Cookies*, cabeçalhos) para tentar evitar o mecanismo de segurança de um sítio ou o fluxo de controle de normal.

Há diferentes tipos de ataques relacionados a este problema. Que serão discutidos de forma mais profunda:

- Cross Site Scripting
- Buffer Overflows
- Falhas na injeção

Existem alguns detalhes que devem ser observados ao se controlar a validação da aplicação. Primeiro, não é suficiente em qualquer aplicação WEB confiar somente em *scripting* executados no lado do cliente (por exemplo *JavaScript*). Este tipo de código normalmente submete de forma estática quando descobre que a informação é inválida. Porém, nada faz para impedir que *hackers* geram suas próprias requisições de HTTP independentes. Resumindo, usar somente validação no lado cliente deixa a aplicação WEB aberta para ataques.

Segundo, algumas aplicações usam uma abordagem "negativa" em validação: tentam descobrir se existe algum elemento prejudicial nos parâmetros de requisição. O problema com este tipo de abordagem é que só pode proteger contra um conjunto limitado de ataques. São prevenidos só os ataques reconhecidos pelo código de validação. Há um número crescente de *hackers* com métodos a usar para evitar a segurança por uma informação não validada; é possível que um novo método não seja reconhecido pela aplicação e sobreponha este tipo de validação e cause destruição. É sempre melhor fazer uso de uma aproximação "positiva": definir um formato ou padrão para valores possíveis e assegurar que a informação recebida respeite esta regra.

3.2. Controle de Acesso Quebrado

Há muitas aplicações que categorizam seus usuários em papéis diferentes e provêem níveis diferentes de interação ou tipos diferentes de conteúdo para cada uma dessas categorias. Como por exemplo, a maioria das aplicações define um papel de usuário e um papel de administrador: só o papel de administrador é autorizado a ter acesso a páginas ou executar ações que são responsabilidade da administração do sítio.

O problema é que algumas aplicações não obrigam esta autorização efetivamente. Como exemplo, alguns programas executam através de um posto de fiscalização dentro do fluxo de tela habitual que só usuários registrados podem passar: o usuário deverá provar que é autorizado através de um nome e uma senha. Porém, essa conferencia não obriga a conferência mais de uma vez após passar este posto de fiscalização: uma vez que um usuário consegue "saltar" este posto, eles podem executar as operações livremente.

Outros problemas relacionados ao controle de acesso incluem:

- **Chaves inseguras** – alguns sítios fazem uso de algum tipo de chave ou recorrem a seus usuários ou funções. Estas chaves podem ser descobertas, porém e se um *hacker* puder fazer uso delas para se passar por um usuário autorizado, então o sítio está aberto ao ataque.

- Permissões de arquivo – a maioria dos servidores de aplicação WEB confiam em um arquivo externo para proporcionar uma lista de usuários autorizados a quais recursos podem ou não ter acesso. Se este arquivo for acessível externamente, então os *hackers* podem modificá-lo para se incluírem na lista de usuários permitidos.

O que podemos fazer para resolver estes problemas? Para uma solução mais simples, podemos desenvolver filtros ou componentes semelhantes aos que podem ser aplicados através recursos sensíveis. Estes devem assegurar que serão autorizados por usuários certificados. Se proteger de chaves inseguras, devemos desenvolver a aplicação de forma a não confiar no segredo de qualquer chave para proporcionar o controle de acesso. Em relação ao problema do arquivo de permissão, estes devem ser colocados em locais inacessíveis para os navegadores WEB e, além disso, marcados no mesmo nível do sistema operacional disponíveis a papéis específicos.

3.3. Autenticação Quebrada e Gerenciamento de Seção

Autenticação e administração de sessão significa controlar a autenticação de usuários e a administração de sessões ativas. Há várias áreas na qual isso pode ser negligenciado, e deste modo, expor a vulnerabilidade do sistema:

- **Tamanho da Senha** - Forçar que a senha da aplicação tenha um tamanho mínimo, o que pode ser conferido verificando o comprimento da senha e sua complexidade. Considere uma aplicação que deixe seus usuários registrados criarem suas próprias senhas: não se deve permitir senhas menores que 5 caracteres e palavras simples que podem ser adivinhadas por *hackers*.
- **Uso da Senha** – Nossa aplicação deve obrigar um número de máximo de tentativas que um usuário pode efetuar durante a sua entrada no sistema por um determinado período de tempo. Isto protege nossa aplicação de ataques como o de forçar a senha onde os *hackers* tentam repetidamente novas senhas. Caso um número determinado de tentativas aconteça, isso deveria ser registrado para fornecer aos administradores uma indicação para alertar de possíveis ataques.
- **Armazenamento da senha** – De forma alguma deve-se armazenar senhas dentro da aplicação. Preferencialmente devem ser armazenadas em um banco de dados externo, um arquivo protegido, entre outros. Devem ser registradas em um formato codificado. Para que aquela informação sensível não se espalhe no caso de uma brecha dentro da aplicação.

Outro assunto relacionado a senhas é que nunca devem ser deixadas no código de fonte da aplicação.

- **Sessão com Proteção de Chave** – Existem servidores que usam chave de sessão para identificar um usuário participante em uma sessão. Porém, esta sessão de chave pode ser recuperada por alguém na mesma rede, de tal forma que outra pessoa possa se fazer passar por um cliente autorizado.

Uma das melhores maneiras de se prevenir uma chave de sessão seja recuperada por alguém na mesma rede é colocar comunicações entre o servidor e o cliente em um canal de SSL protegido.

3.4. Cross Site Scripting

Cross Site Scripting acontece quando alguém conseguir utilizar uma aplicação WEB para enviar códigos maliciosos para outro usuário. Isto pode ser feito embutindo o conteúdo ativo (como *JavaScript*, objetos *ActiveX* ou *Flash*) em uma requisição que gera uma resposta em HTML para ser vista por outro usuário. Uma vez que outros usuários têm acesso a este conteúdo, os navegadores não sabem que estes não serão confiáveis.

Um dos melhores modos de se prevenir ataques *Cross Site Scripting* é validar todos os dados que entram nas requisições do usuário (cabeçalhos, cookies, parâmetros do usuário, entre outras). Uma abordagem "negativa" que não deveria ser usada é tentar filtrar todas as formas de conteúdo ativo embutido, pois não é uma forma cem por cento efetiva.

3.5. Buffer Overflows

Ataques podem usar *Buffer Overflows* para corromper a ordem de execução de uma aplicação WEB. Isso pode ser feito enviando requisições cuidadosamente desenvolvidas que ordenam que o servidor execute um código arbitrário.

Problemas de *Buffer Overflows* normalmente são difíceis de descobrir e de explorar através de *hackers*. Porém, os atacantes ainda conseguem identificar este tipo de vulnerabilidade contando com vários produtos. Entretanto, graças ao projeto do ambiente de aplicações Java que são executados dentro de um servidor padrão Java EE, estão imunes a estes tipos de ataque.

Para assegurar nossa segurança, entretanto, é sempre melhor monitorar constantemente a disponibilidade de correções e novas versões dos produtos que estamos utilizando.

3.6. Falhas de Injeção

Um tipo popular de vulnerabilidade é denominada de Falhas de Injeção na qual os *hackers* podem enviar ou "injetar" chamadas ao sistema operacional ou para recursos externos como os bancos de dados.

Uma falha de injeção comum é a *SQL injection*. Examine a seguinte URL:

```
http://someServer/someApp/someAction?searchString=jedi
```

Esta URL acima é possivelmente gerada por uma aplicação que possui um formulário em que há uma busca na base de dados por uma chave chamada 'jedi'. Implementações que não validam suas entradas de dados provavelmente se pareceriam com o código SQL a seguir:

```
select * from someTable where someField='value'
```

Onde *value* é o valor do parâmetro *searchString* recebido diretamente da requisição HTTP. E se, por exemplo, algum hacker digitar diretamente a seguinte URL:

```
http://someServer/someApp/someAction?searchString=jedi'%20AND%20true;%20DROP
%20DATABASE;'
```

A query SQL gerada seria então algo como:

```
select * from someTable where someField='jedi' AND true;DROP DATABASE;''
```

A primeira sentença seria aceita devido a "*AND true*" na cláusula *where* do comando *select.* A segunda sentença seria então executada, ocasionando muitos danos à aplicação.

Este é um tipo de ataque feito através de entradas inválidas. Com exceção á rigorosa validação de informações em requisições do usuário, existem duas outras precauções que podemos ter:

- Em vez de usar sentenças simples como: *SELECT*, *INSERT*, *UPDATE* e *DELETE*, crie funções que executam funcionalidades equivalentes. As funções tem o benefício de tratar os parâmetros como dados, ao contrário de simplesmente executar o conteúdo. Requerem que os parâmetros estejam no tipo específico declarado. Elas fornecem uma quantidade significativa de proteção contra injeção de SQL, embora a validação também deva ser executada.
- Fornecer à aplicação somente a quantidade mínima de privilégios que necessitam para executar as funcionalidades desejadas. Por exemplo, se a aplicação for inicialmente uma aplicação de visualização de dados, não dê privilégios para INSERT, UPDATE ou DELETE. Não deixe a aplicação WEB acessar a base de dados usando usuário com privilégio de administrador. Isto pode minimizar os danos que os *hackers* podem fazer.

3.7. Armazenamento Inseguro

Aplicações WEB usualmente necessitam armazenar informações sigilosas como senhas, informações de cartão de crédito e outras. Devido a essa natureza sigilosa, estes itens são geralmente (o que deveria ser obrigatoriamente) encriptados para impedir acessos não

permitidos. Entretanto, tais implementações encriptadas às vezes são fracas ou vulneráveis a ataques persistentes.

Estes são alguns erros geralmente cometidos :

- Falha ao encriptar os dados críticos.
- Armazenamento inseguro de chaves, certificados e senhas.
- Armazenamento impróprio de segredos na memória.
- Funções de randomizações pobres.
- Escolha ruim dos algoritmos para encriptar.
- Tentar inventar novos algoritmos para encriptar.

Considere o seguinte cenário: existe uma aplicação que, na modelagem dos atributos e estados do usuário, inclua a senha como parte do objeto Usuário. Entretanto, após a entrada do usuário em um mecanismo de autenticação, a aplicação armazena o objeto Usuário na sessão. O problema com esse cenário é que a senha pode agora facilmente ser recuperada por alguém que consiga quebrar ou penetrar na sessão do usuário.

Uma boa política para evitar armazenamento inapropriado de informações sigilosas é não modelar a senha como atributo da classe que representa o usuário se esta classe precisa ser armazenada em algum lugar na aplicação. Em vez de encriptar o número do cartão de crédito do usuário, simplesmente solicite-o sempre que necessário.

Também é melhor empregar algoritmos externos para encriptar em vez de desenvolver um. Certifique-se de que o algoritmo que você usará foi submetido a exame público e se provou ser de confiança.

3.8. Negação de Serviço

Negação de Serviço (*Denial of Service* ou *DoS*) refere-se a ataques maliciosos realizados por *hackers* utilizando concorrência múltipla de requisições ao servidor. Devido ao grande número de requisições, o servidor torna-se ocupado e fica indisponível para executar outros serviços aos usuários.

Os ataques de DoS fazem mais que apenas consumir a banda do servidor. Podem também consumir recursos limitados importantes tais como a memória, conexão a base de dados, elevar o tempo de resposta do processador, serviços ou recursos específicos da aplicação.

Normalmente é muito difícil proteger completamente a aplicação contra esse tipo de ataque, porque não existe uma solução 100% segura. Uma boa regra é limitar o número de recursos disponibilizados aos usuários ao mínimo necessário. Pode ser também uma boa idéia estabelecer cotas que determinam a carga que um usuário pode impor ao sistema.

Um exemplo dessa limitação de recursos é característica comum entre as implementações de *bulletin boards*. Limitam o usuário a executar operações de buscas uma vez a cada 20 segundos. Isto certifica que o usuário não irá consumir uma grande parcela de conexões à base de dados disponível.

Outra solução possível é projetar a aplicação WEB de forma tal que os usuários não autorizados tenham pouco ou nenhum acesso a conteúdo que necessite da base de dados ou algum outro recurso caro.

3.9. Gerenciamento inseguro de configurações

Usualmente o grupo de desenvolvimento da aplicação é diferente do grupo que possui a responsabilidade de administração do servidor que hospeda a aplicação. Infelizmente, há um limite de segurança que se pode ter na aplicação quando a segurança do servidor é em grande parte a determinante em como sua aplicação é considerada segura. Uma configuração imprópria no lado do servidor pode invalidar todos os esforços dos desenvolvedores para proteger a aplicação.

Estes são alguns erros em configurações de servidores que provam esta problemática:

- Falhas por *Unpatched Software* no servidor – os administradores não estão cientes de

liberações de novos *releases* de *patches* para o servidor.
- Falhas de segurança no servidor que permitem a listagem de diretórios ou os chamados "*directory traversal attacks*".
- *Backups* desnecessários ou arquivos de exemplo incluindo scripts, aplicações, arquivos de configuração e páginas WEB.
- Permissões impróprias de arquivos e diretórios.
- Serviços desnecessários como administração remota ou gerenciamento de conteúdo permitidos e esquecidos.
- Usuários *default* com senhas *default*.
- Acesso a funções administrativas ou de depuração.
- Excessiva informação em mensagens de erro.
- Configuração ruim de certificados SSL e parâmetros para encriptar.
- Uso de auto-assinaturas em certificados para prover autenticações.
- Uso de certificados *default*.
- Autenticação imprópria com sistemas externos.

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.