

Segurança No Desenvolvimento JavaEE

Fernando Lozano

www.lozano.eti.br

[<fernando@lozano.eti.br>](mailto:fernando@lozano.eti.br)

Arquiteto de Soluções – Neki Technologies

www.neki.com.br

Linux.Java.Net Community

SouJava - Rio de Janeiro, Brasil

Sobre o Autor

- ▶ Consultor Independente com mais de 10 anos de atuação na área de TI, em Sistemas de Informação e Integração de Redes
- ▶ Detentor de diversas certificações profissionais da Microsoft, IBM, Sun, Red Hat e LPI
- ▶ Ex-conselheiro do Linux Professional Institute
- ▶ Colunista da Revista Java Magazine
- ▶ Colaborador do IBM Developer Works e NetBeans Magazine
- ▶ Community Manager do Java.Net
- ▶ Webmaster / Coordenador de Tradução da Free Software Foundation
- ▶ Autor do livro “Java em GNU/Linux”, ed. Alta Books

Patrocinador



“Missão: Otimizar Resultados”
Especialista em soluções open source
baseadas na plataforma Java



Consultoria em Desenvolvimento
e Infra-Estrutura de Sistemas



Capacitação e Mentoring
em Tecnologia e Processos

Parcerias Neki



Gerenciamento de Riscos
Soluções em Segurança e
Biometria



Professional Open Source

Agenda

- ▶ Recursos de segurança da plataforma JavaEE
- ▶ Ataques mais comuns sobre aplicações web
- ▶ Melhores práticas de segurança para aplicações

Segurança de Software

- ▶ Tradicionalmente implementada por tecnologias de autenticação, autorização, auditoria e encriptação
- ▶ Estes mecanismos são suficientes para aplicações tradicionais, mas não para aplicações distribuídas, especialmente aplicações web
- ▶ A plataforma Java foi a primeira no mercado a incorporar recursos voltados especificamente para aplicações distribuídas

O Programador Arrogante

- ▶ Todo programador acredita que é capaz de programar qualquer coisa..
- ▶ .. E que vai ser fácil programar
- ▶ Entretanto a programação de aplicações (voltadas para usuários finais) exige um conhecimento teórico totalmente diverso da programação de infra-estrutura (software básico)
- ▶ Um projeto de aplicação nunca tem orçamento ou prazo para que este programador adquira o conhecimento e experiência que faltam

É um Programador Idiota

- ▶ O resultado final é que o programador, em vez de se dedicar a aprender a usar a infra-estrutura de segurança existente e comprovada
- ▶ Gasta tempo em uma implementação incompleta e falha de mecanismos de segurança que já existem
- ▶ Máxima dos especialistas em segurança:
“Qualquer idiota é capaz de conceber um esquema de segurança que ele mesmo não consegue quebrar”

Recursos de Segurança do Java EE

- ▶ Security Manager
- ▶ JCA – Java Cryptography Architecture
- ▶ JAAS – Java Authorization and Authentication System
- ▶ JACC – Java Authentication Contract for Containers
- ▶ Segurança declarativa para Servlets e EJBs
- ▶ Pouco usados na prática pelos desenvolvedores Java, por simples desconhecimento

Recursos Complementares do Java EE

- ▶ Logging – pode ser usado para gerar logs de auditoria
- ▶ JMX – permite obter em tempo real informações de performance ou status de componentes de aplicação, do container ou da própria JVM
- ▶ Classloaders – permite compartimentalizar classes de diferentes origens ou aplicações na mesma JVM, de modo que uma não interfere no funcionamento da outra, nem mesmo por meio de atributos estáticos públicos

Security Manager

- ▶ Recurso que permite a criação de *sandboxes*, como a utilizada para executar applets Java dentro de um navegador web
- ▶ Permite (ou não) a invocação de métodos de acordo com a origem do chamador
- ▶ Cada método sujeito a autorização deve interagir explicitamente com o Security Manager
- ▶ Métodos da API padrão do Java SE usam este recurso

JCA, JCE, JSSE, PKI

- ▶ Suportam vários algoritmos de criptografia simétrica e assimétrica
- ▶ Implementam os protocolos SSL e TLS para comunicação segura
- ▶ Permitem a criação e manipulação de certificados e assinaturas digitais, além de chaves públicas e privadas

JAAS

- ▶ Permite criar módulos de login vinculados a diferentes bases de dados de usuários (arquivos, bancos de dados, diretórios LDAP, etc)
- ▶ Isola estes módulos da interface com o usuário (ou do protocolo de rede) para obter login, senha, tokens, biometria ou outras formas de credenciais
- ▶ Permite empilhar módulos no estilo do PAM do Unix

Módulos de Login Empilháveis

- ▶ Permitem a composição e reuso de módulos especializados como parte de uma política mais ampla de autorização
- ▶ Permite ainda configurar mecanismos de autenticação alternativos
- ▶ Por exemplo, um módulo de login verifica senhas contra um banco de dados, se isto não funcionar a senha é verificada contra uma base LDAP, e em seguida o login é autorizado apenas em horário comercial

JACC

- ▶ Define como um container Web ou EJB deve fornecer o contexto de uma requisição remota e sessão de usuário para um Security Manager
- ▶ Permite definir regras de autorização baseada nos argumentos da requisição (parâmetros HTTP ou argumentos de métodos remotos)
- ▶ Elimina a maioria dos casos em que é necessário inserir lógica programática de segurança
- ▶ Infelizmente o Security Manager tem escopo da JVM e não do deployment WAR / EJB-JAR / EAR

Infra-Estrutura x Aplicação

- ▶ Security Manager, JAAS e JCA são APIs de infraestrutura fornecidas pelo Java SE
- ▶ O JAAS é mandatório no Java EE 1.3, e o JAAC no Java EE 1.4
- ▶ Elas atendem à programação de containers: servidores JavaEE e frameworks de interface gráfica como o NetBeans Platform ou o Eclipse RCP
- ▶ O programador de aplicações informação não deveria lidar diretamente com elas

Segurança Declarativa

- ▶ Estabelece a visão do programador de aplicação sobre os recursos de segurança
- ▶ Estipula que as regras de controle de acesso sejam definidas pela associação de recursos (URLs, métodos remotos) a grupos (roles) que autorizados a acessá-los
- ▶ Esta associação é especificada no descritor de deployment de um pacote WAR ou EJB-JAR
- ▶ O container tem liberdade em como configurar o mapeamento de usuários a roles

Segurança Programática

- ▶ A segurança declarativa é complementada por uma API simples para permitir lógica procedural em situações que não podem ser especificadas declarativamente
- ▶ Por exemplo, um usuário só pode editar um registro (ou EJB) criado por ele mesmo
- ▶ Os métodos são definidos nas APIs de Servlets e EJB, da mesma forma que a sintaxe para o descritor de deployment

Segurança Declarativa x Programática

- ▶ Especifica **O QUE** restringir
- ▶ A aplicação não precisa ser modificada ou recompilada para mudar as regras
- ▶ A integração com provedores de segurança é responsabilidade do container
- ▶ Especifica **COMO** restringir
- ▶ A aplicação tem que ser alterada para modificar as regras
- ▶ A integração com provedores de segurança é codificada dentro da aplicação

Responsabilidades do Programador

- ▶ Declarar no deployment descriptor as regras que definem a política de segurança da aplicação
- ▶ Definir um conjunto de roles alinhado com os papéis de usuários (atores) nos casos de uso da aplicação
- ▶ Usar as informações fornecidas pelo container quando é necessário embutir alguma regra de autorização na lógica de negócio

Responsabilidades do Administrador

- ▶ Configurar o servidor de aplicações para usar a base de usuários preferencial do ambiente (Banco de dados de RH, Active Directory, NIS...)
- ▶ Obter um front-end para a administração da base de usuários e dos mapeamentos a roles
- ▶ Configurar parâmetros de criptografia e expiração de senhas adequados
- ▶ Monitorar logs do servidor de aplicações, base de dados de usuários, firewall e IDS para identificar possíveis ataques

Usar o Padrão Java x Criar um Mecanismo Próprio

- ▶ Dificilmente o desenvolvedor de aplicações tem know-how suficiente para criar um mecanismo de autenticação e autorização que não deixe furos graves, permitindo invasões
- ▶ Mecanismos de autenticação in-house em geral são incompatível com o modelo de interação da web (por exemplo, bookmarks do navegador)
- ▶ Mecanismos de autenticação in-house não são flexíveis para acomodar evoluções na estrutura como single-sign on ou biometria

Melhores Práticas

- ▶ SEMPRE use os mecanismos fornecidos pelo Java SE e EE em detrimento de mecanismos desenvolvidos *in-house*
- ▶ O front-end de administração de usuários e roles pode e deve ser independente das aplicações
- ▶ Senhas e outras credenciais devem trafegar na rede sempre usando canais encriptados por SSL ou TLS
- ▶ Ou então deve ser usado um mecanismo de tiket como o Kerberos

Segurança na Internet

- ▶ Agora sua aplicação está:
 - ▶ Autenticando usuários
 - ▶ Autorizando o acesso a funcionalidades de acordo com o usuário logado
 - ▶ Salvando dados ou se comunicando usando criptografia
- ▶ Por que isto **ainda não é suficiente** para se considerar a aplicação segura?

Princípios de Segurança

- ▶ *Controle* – tudo tem que ser explicitamente autorizado
- ▶ *Prevenção* – deve-se impedir (bloquear) acessos indevidos
- ▶ *Contenção* – como não existe sistema 100% seguro, deve-se minimizar o estrago causado por eventuais ataques
- ▶ *Monitoração* – é preciso vigiar os próprios mecanismos de segurança para perceber que eles foram contornados

Segurança e Automação

- ▶ Foram controlar todas as possibilidades de acesso?
- ▶ Foram mapeadas todas as formas de ataque?
- ▶ Há barreiras de contenção para limitar todos os ataques
- ▶ Temos informação para monitorar todas as aplicações?
- ▶ O problema real é que o universo de possibilidades aumenta muito em aplicações web

Aplicações Web

- ▶ Em uma aplicação tradicional, não existe a possibilidade de se “corromper” os dados armazenados em memória entre uma tela e outra
- ▶ Em uma aplicação web, nada garante que o usuário segue o fluxo normal de navegação da aplicação – o hacker pode “pular telas”
- ▶ É muito fácil “fabricar” requisições HTTP contendo parâmetros que nunca poderiam ser gerados pelas páginas reais da aplicação

Aplicações x Firewall

- ▶ Todo firewall tem que deixar abertas as portas que permitem o acesso às aplicações
- ▶ Para o firewall, não é possível diferenciar uma requisição válida e uma requisição “fabricada” direcionadas para uma mesma porta
- ▶ Portanto nenhum firewall protege um servidor ou uma rede (e os dados armazenados neles) de falhas de segurança nas aplicações web

Aplicações x IDS

- ▶ Um IDS (ou um firewall de “camada 7”) poderia incorporar heurísticas para diferenciar requisições reais de fabricadas por hackers
- ▶ Mas ele não tem conhecimento prévio de todos os ataques possíveis, como um anti-vírus tem (depende da **SUA** aplicação)
- ▶ E ele não pode salvar requisições suspeitas para análise posterior, como um anti-spam faz
- ▶ Ou seja, a eficácia da proteção automatizada é muito baixa para aplicações

Ataques Contra Aplicações

- ▶ Buffer Overflow
- ▶ Data Tampering
- ▶ Script Injection
- ▶ Impersonalization

- ▶ A maior parte desta discussão se aplica igualmente a PHP, ASP e outros ambientes de desenvolvimento web

Buffer Overflow

- ▶ Envia-se um valor de parâmetro maior (em bytes) do que o esperado
- ▶ Este valor sobrescreve outras áreas de memória, alterando variáveis ou mesmo código
- ▶ Permite ao cracker inserir código na aplicação, que tem acesso aos mesmos dados e servidores que a aplicação real teria
- ▶ Não existem buffer overflows em código Java!

Buffer Overflow x Java

- ▶ Geralmente ocorre em código nativo (C, Delphi, Assembler)
- ▶ Linguagens de script como PHP e VBA/ASP dependem de componentes escritos em código nativo para a maioria das operações, e portanto herdam seus bugs de buffer overflow
- ▶ Como a maior parte da API do Java é escrita em Java, e a JVM não permite desabilitar verificações de tipo, tamanho e faixa, só podem haver buffer overflows na própria JVM escrita em C

Data Tampering

- ▶ Envia-se um valor diferente do esperado pela aplicação
- ▶ Este valor induz a aplicação a fazer uma operação diferente da que faria normalmente, dando acesso a dados que deveriam estar ocultos
- ▶ Ou então o valor coloca a aplicação em loop, ou provoca erros mais adiante (como divisão por zero) efetivamente derrubando a aplicação em um ataque DoS – *Denial of Service*

Exemplo

- ▶ Uma página lista registros visíveis pelo usuário, filtrados pelo nome do usuário autenticado pelo servidor
- ▶ A página seguinte é um formulário de edição, que recebe da página com a lista o id ou pk (chave primária) do registro a editar
- ▶ O cracker chama diretamente a página de edição, fornecendo o id de um registro que não apareceria para ele na página de listagem e assim editando dados que ele não estaria autorizado

Script Injection

- ▶ É uma forma de data tampering que insere código para execução pelo interpretador de scripts presente no servidor ou no cliente
- ▶ Este código pode ser HTML, Javascript, SQL, PHP, VBA ou qualquer outra “linguagem de macros”
- ▶ O cracker insere o código como parte dos dados enviados para a aplicação. Este código poderá ser executado imediatamente pelo servidor, ou pelo cliente (navegador) como parte da página de resposta

Exemplo 1/2

- ▶ Uma página de consulta lista registros baseado em região geográfica, montando um comando SQL como
 - ▶ `select nota_fiscal, valor from compras where estado = 'RJ'`
- ▶ A string “RJ” é recebida como parâmetro de uma requisição HTTP (em uma URL GET ou formulário POST)

Exemplo 2/2

- ▶ O hacker envia como string um outro comando SQL, que será executado junto à consulta:
 - ▶ `select nota_fiscal, valor from compras where estado = 'RJ' ; delete from compras where fornecedor = 'ACME'`
- ▶ O hacker fabrica uma requisição contendo o parâmetro:
 - ▶ `"RJ" ; delete from compras where fornecedor = 'ACME"`
- ▶ Que é para o banco de dados um comando SQL composto válido

Cross-Site Script

- ▶ É uma variação do ataque de injeção que tem por objetivo contornar firewalls
- ▶ Em vez de inserir código para execução imediata, o código é inserido para execução posterior por um outro usuário
- ▶ Muitas vezes, código “não executável” como tags `` do HTML, objetos Flash e referências a style sheets (CSS) são o veículo do ataque, porque eles provocam conexões de rede

Exemplo

- ▶ O cracker assina um livro de visitas que permite formatação HTML nas mensagens
- ▶ Ele insere dentro do HTML tags `<script>` contendo código JavaScript
- ▶ O administrador do site, ao ver as últimas mensagens do livro de visita, executa o código JavaScript em sua estação de trabalho
- ▶ O código JavaScript envia para o cracker por e-mail todas as senhas armazenadas pelo navegador do administrador

Impersonalization

- ▶ Consiste em assumir uma sessão já autenticada por um outro usuário, se passando por ele ou pela sua estação de trabalho
- ▶ Em geral envolve a captura e replay ou então a previsão de um cookie de session_id
- ▶ O suporte a sessões HTTP de um container Web JavaEE gera session_ids resistentes a previsão, mas não a captura
- ▶ Neste caso use SSL ou force a renovação do session_id a intervalos curtos

Melhores Práticas

- ▶ Valide tudo, o tempo todo, no servidor
- ▶ Valide os dados esperados (validação positiva), nunca os dados considerados inválidos (validação negativa)
- ▶ Valide também os dados recuperados de arquivos e bancos de dados
- ▶ Sempre formate e “escape” dados que serão exibidos para o usuário, ou inseridos em um comando ou script

Valide Tudo no Servidor

- ▶ A validação feita no navegador pode ser desligada pelo usuário e não terá sido realizada por uma requisição fabricada
- ▶ Ela serve para melhorar o tempo de resposta, nunca para dar segurança e integridade aos dados
- ▶ Dados enviados para o navegador podem ser modificados antes de serem retornados, por isso devem ser re-validados

Valide Tudo o Tempo Todo

- ▶ Qualquer página pode ser executada “fora de ordem” por isso a página tem que re-validar todos os seus dados de entrada, mesmo que eles já tenham validados por uma página anterior
- ▶ Não podem existir campos de texto livre, isto é, cujo conteúdo é aceito integralmente e sem restrição, pois eles são veículos para ataques de injeção
- ▶ Sempre valide o tamanho esperado de todos os campos, mesmo que sejam “ilimitados” para o banco de dados (memos e BLOBs)

Validação Positiva

- ▶ É mais fácil definir todas as possibilidades de dados válidos do que todas as possibilidades de dados inválidos
- ▶ Por isso deve-se validar o dado esperado, e recusar qualquer dado diferente
- ▶ Sempre é possível restringir caracteres especiais e tags de formatação aceitos em campos de texto livre

Valide Arquivos e BDs

- ▶ Tudo o que vem de fora da página em que ser validado
- ▶ Dados oriundos de arquivos, BDS e conexões de rede (como WebServices) também devem ser validados
- ▶ Eles podem ter sido previamente corrompidos por outras aplicações ou serem veículos de ataques indiretos de injeção como o Cross Site Scripting
- ▶ Não confie que outras aplicações sejam seguras!

Formate e Escape Dados de Saída

- ▶ Toda linguagem de script tem sintaxes para “anular” o significado especial de certos símbolos
- ▶ Esta sintaxe é chama de “escape”
- ▶ Por exemplo, uma contra-barra (\) anula o significado de um apóstrofo como fechar string no SQL
- ▶ Garanta que dados de saída (enviados para um navegador, banco de dados ou outra aplicação) serão tratados como dados puros

Como o Java Ajuda?

- ▶ PreparedStatements SQL previnem ataques de Injeção de SQL
- ▶ Suporte a expressões regulares facilita validar dados de entrada, e escapar dados de saída
- ▶ O Security Manager pode ser configurado para impedir que a aplicação acesse arquivos ou inicie conexões de rede inesperados
- ▶ Informações armazenada no container não podem ser adulteradas, ao contrário de informações armazenadas no navegador

Além do Java

- ▶ Use mecanismos do SO e da rede para dar proteção adicional (segurança em profundidade)
- ▶ Nunca instale componentes nem ative serviços que não sejam estritamente necessários
- ▶ Cada usuário e componente deve ter as permissões mínimas necessárias para realizar suas atividades
- ▶ Segurança exige trabalho intenso e constante!

Perguntas?

www.owasp.org

mysqldump.azundris.com/archives/49-security-and-the-real-world.html

java.sun.com

www.lozano.eti.br