Boletim Técnico da Escola Politécnica da USP
Departamento de Engenharia de Computação e
Sistemas Digitais

ISSN 1413-215X
BT/PCS/0122

Modelo de Segurança da Linguagem Java Problemas e Soluções

Claudio Massanori Matayoshi
Wilson Vicente Ruggiero

São Paulo - 2001

O presente trabalho é parte da dissertação de mestrado apresentada por Claudio Massanori Matayoshi, sob a orientação do Prof. Dr. Wilson Vicente Ruggiero.: "Modelo de Segurança da Linguagem Java Problemas e Soluções", defendida em 25/04/01, na EPUSP.

A íntegra da dissertação encontra-se à disposição com o autor e na Biblioteca de Engenharia Elétrica da Escola Politécnica da USP.

FICHA CATALOGRÁFICA

Matayoshi, Claudio Massanori
Modelo de segurança da linguagem Java : problemas e soluções / C.M. Matayoshi, W.V. Ruggiero. – São Paulo : EPUSP, 2001.
13 p. – (Boletim Técnico da Escola Politécnica da USP, Departamento de Engenharia de Computação e Sistemas Digitais, BT/PCS/0122)

1. JAVA (Linguagem de programação) 2. Segurança em computador I. Ruggiero, Wilson Vicente II. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais III. Título IV. Série
ISSN 1413-215X CDD 005.133
005.8

Modelo de Segurança da Linguagem Java
Problemas e Soluções

Claudio Masanori Matayoshi – cmmatayo@larc.usp.br
Dr. Wilson Vicente Ruggiero – wilson@larc.usp.br

Departamento de Engenharia de Computação e Sistemas Digitais
EPUSP - Escola Politécnica da Universidade de São Paulo

Resumo : Entre as inúmeras tecnologias que surgiram para permitir adicionar processamento nas páginas web apresentadas pelos navegadores, a linguagem Java tem se destacado, com seu objetivo de permitir uma total independência de plataforma. Entretanto, ao possibilitar a execução no computador do usuário de um código de origem externa, inúmeras questões referentes à segurança devem ser analisadas. Este trabalho é resultado da análise do modelo de segurança adotado pela linguagem Java e sua correspondente implementação sob a forma de máquina virtual. Foi identificado um problema potencial presente no estado atual da linguagem que poderiam ser explorados em ataques. Um mecanismo adicional de segurança foi proposto e implementado para solucionar este problema, assim como sugestões para incrementar o modelo de segurança.

Abstract : Among many technologies that allow embedding software programs in web pages, the Java language stands out due to its platform independence. However, running applications from external sources rises many issues related to security. This work is result of Java security model and Java Virtual Machine implementation analysis. A potential security issue in current security model is identified. A system to avoid this vulnerability is proposed and implemented. An improvement to the security model is suggested..

1. Introdução

A abertura da Internet para uso comercial, aliada ao surgimento da World Wide Web (WWW) com sua interface gráfica de fácil uso, geraram um enorme crescimento na quantidade de usuários e variedade de serviços disponíveis nestes últimos anos. Atualmente, grande parte das instituições públicas e privadas fornecem algum tipo de informação ou serviço através da Internet.

A interação básica do usuário com a Internet ocorre através do programa cliente do protocolo http (hyper-text transfer protocol). Tal programa cliente é denominado navegador (ou "browser").

Para utilizar o potencial de processamento disponível nas máquinas dos usuários e possibilitar a criação de aplicações até então tecnicamente inviáveis surgiram inúmeras tecnologias : Java, JavaScript, ActiveX, Plug-Ins etc.

De uma maneira geral, estas tecnologias permitem que o navegador realize um processamento de informações na máquina do usuário, executando um código que é enviado junto com as páginas WWW. Neste ponto surge a grande (e mais que justificável) preocupação do usuário quanto a segurança em permitir que um código externo seja executado em sua máquina sem comprometer a integridade e confidencialidade de seus recursos e informações.

Para permitir que a linguagem Java tivesse aceitação junto aos usuários das aplicações, a preocupação com a segurança está presente desde a sua concepção inicial. Entretanto, esta preocupação não impede que eventuais incidentes surjam.

A tecnologia Java tem evoluído constantemente, ultrapassando os limites de uma linguagem de programação e passando a abranger todo um ambiente para execução de aplicações. Uma série de aplicações tem sido propostas e implementadas utilizando este ambiente Java como base, não ficando mais restritas apenas a aplicações embutidas em páginas Web (os chamados *applets*).

Cada vez mais aplicações estão baseados neste ambiente e as questões de segurança tornam-se um fator primordial para a confiança dos usuários e dos responsáveis pelos sistemas de informação.

A partir da avaliação do modelo de segurança e da análise do histórico de incidentes realizadas neste trabalho, identificou-se uma vulnerabilidade potencial existente no estado atual da linguagem. Esta vulnerabilidade pode ser explorada com sucesso por ataques contra as versões atuais dos navegadores Internet mais utilizados atualmente. Um mecanismo adicional de segurança foi proposto e implementado para solucionar o problema, e recomendações para alteração no modelo de segurança são realizadas para eliminar tal vulnerabilidade.

O artigo está estruturado da seguinte forma : a Seção 2 descreve o modelo de segurança da linguagem Java e a Seção 3 discute uma forma possível de ataque e propõe uma solução. Leitores já familiarizados com a linguagem Java e o seu modelo de segurança poderão prosseguir a leitura a partir da Seção 3.

2. O Modelo de Segurança da Linguagem Java

A segurança é um aspecto fundamental para a linguagem Java.

Na principal aplicação desta linguagem (distribuição de *applets* embutidas em páginas WWW para serem executadas nas máquinas dos usuários), o usuário estará recebendo um código através de uma rede de dimensões mundiais (a Internet). Neste momento surge a questão de como ter certeza de que não se está trazendo um vírus ou um cavalo de tróia[1] para dentro do computador ?

A questão de segurança na linguagem Java apresenta dois aspectos [GMPS97] :

- Plataforma para permitir a execução de aplicações Java de forma segura (aspecto abordado nesta dissertação de mestrado)
- Ferramentas e serviços de segurança para aplicações (implementação de funcionalidades criptográficas, manipulação de certificados digitais, etc)

O modelo de segurança Java tem evoluído constantemente. Os três principais marcos desta evolução são apresentados a seguir.

2.1. Modelo de Segurança Java 1.0.x (modelo de segurança original)

O modelo de segurança original é conhecido como modelo "sandbox", que se propõe a fornecer um ambiente bastante restrito para execução de código não confiável. Todo código Java obtido através da rede (*applets* Java embutidos em páginas html que são trazidos de fora do ambiente local do usuário) é considerado código não confiável.

Este ambiente restrito e controlado é chamado de "sandbox". O programa atua dentro dos limites definidos pela "sandbox", não devendo realizar nenhuma ação fora destes limites.

Neste modelo, *applets* Java trazidos a partir da rede não podem realizar inúmeras ações, entre elas :

- ler ou escrever no sistema de arquivos da máquina do usuário
- estabelecer conexões de rede para qualquer servidor, a não ser com o servidor do qual o *applet* foi carregado
- criar novos processos
- carregar dinamicamente novas bibliotecas e chamar diretamente métodos nativos

Ao evitar que o código trazido através da rede execute certas operações, o modelo de segurança Java pretende proteger o usuário de programas hostis.

Aplicações Java instaladas no computador do usuário (também conhecidas como aplicações "stand-alone") são consideradas confiáveis, tendo acesso irrestrito aos recursos do sistema.

---

[1] *Cavalo de Tróia : programa desenvolvido com o intuito de ao ser executado realizar ações ofensivas e/ou destrutivas no computador da vítima.*

2.2. Modelo de Segurança Java 1.1.x

O modelo original de segurança impedia a utilização por qualquer *applet* dos recursos locais do computador do usuário e restringia o destino das conexões de rede, mesmo que o usuário quisesse permitir tal acesso. Isto gerou uma limitação nas funcionalidades que poderiam ser implementadas em Java.

A versão 1.1 flexibilizou este cenário, introduzindo o conceito de *applet* assinado digitalmente. Um *applet* assinado digitalmente é tratado como código local confiável se a chave utilizada para assinatura é reconhecida como confiável pelo sistema que recebe o *applet*.

A assinatura digital do *applet* permite que se identifique o seu autor, ficando a cargo do usuário ou administrador do sistema no qual o *applet* será executado determinar se confia ou não na origem da aplicação. Os *applets* assinados por autores confiáveis têm acesso irrestrito aos recursos do computador.

As classes que implementam um *applet*, outros recursos que sejam necessários para sua execução (como imagens) e a assinatura digital de seu responsável são distribuídos em arquivos no formato JAR (Java Archive). Este formato prevê também a compactação do seu conteúdo, otimizando a distribuição.

2.3. Modelo de Segurança Java 2

A nova arquitetura de segurança introduz a possibilidade de controle de acesso com maior detalhamento.

Diferentemente do modelo da versão 1.1.x, que permite apenas a escolha entre acesso irrestrito ou nenhum acesso aos recursos locais, agora é possível detalhar quais tipos de recursos estarão disponíveis, dependendo do responsável pelo código e do servidor que disponibilizou as classes através do conceito de domínios de proteção ("protection domains").

Um domínio de proteção é formado por uma fonte de código e as permissões de acessos para as aplicações provenientes desta fonte de código [OAK98]. Um domínio de proteção é o conjunto da política de segurança e da definição de sua área de atuação [MDOY98]. A fonte do código é determinada por sua origem (URL pela qual o código é obtido) e por um responsável (chamado na literatura original de "principal") identificado pela entidade que assina o código.

A política de segurança, através dos domínios de proteção, é definida pelo usuário ou pelo administrador do sistema.

As checagens de segurança passaram a ser realizadas também para aplicações "stand-alone" além dos *applets*.

Nesta versão foram realizadas uma série de ajustes estruturais internos na implementação para reduzir os riscos de surgimento de brechas de segurança [GMPS97]. Este esforço envolveu a revisão do projeto e implementação das classes SecurityManager e ClassLoader, assim como o mecanismo de controle de acesso a recursos.

3. Vulnerabilidade Remanescente – Incorporação de Classes de Origem Suspeita no *CLASSPATH*

Analisando o modelo de segurança da linguagem Java e as implementações de JVM[2], foi detectada uma vulnerabilidade que poderia ser explorada num ataque :

Supondo o seguinte cenário : a página do site-alvo do ataque possui um *applet* implementado pela classe contida no arquivo "Aplicacao.class" que implementa uma função fundamental. Por exemplo, criptografia adicional de dados que serão submetidos ao servidor. Vamos chamar este arquivo de "Aplicacao.class alvo".

Nada impede que qualquer programador mal-intencionado crie um outro *applet* diferente, implementado por uma classe de mesmo nome, que consequentemente terá o arquivo contendo os *bytecodes*[3] de nome

[2] JVM – máquina virtual Java

"Aplicacao.class". Caso este *applet* mal-intencionado seja executado no lugar do *applet* original do site alvo do ataque, ele poderia tentar desviar o usuário para um site falso (numa atualização do ataque "man-in-the-middle") ou enviar informações fornecidas pelo usuário que acredita estar utilizando o *applet* oficial. Vamos chamar este arquivo de "Aplicacao.class falso".

O processo de carga de classes da JVM separa *applets* obtidos de origens distintas, evitando que a JVM considere o arquivo de *bytecodes* "Aplicacao.class" falso fornecido por um site preparado para disparar o ataque como sendo o arquivo de *bytecodes* que deva ser utilizado para execução do *applet* alvo do ataque.

Experimentalmente verificamos como a máquina virtual dos navegadores se comportariam caso encontrassem em seu sistema local de arquivos o arquivo contendo os *bytecodes* "Aplicacao.class falso". Mais especificamente, o arquivo "Aplicacao.class falso" foi colocado nos diretórios contendo as classes básicas da máquina virtual Java dos navegadores (chamado de "*CLASSPATH*").

Notamos que tanto no Netscape Communicator 4.75, Microsoft Internet Explorer 4.02 e Microsoft Internet Explorer 5 tivemos o mesmo resultado : ao acessar a página do site alvo, contendo referência ao arquivo de *bytecodes* "Aplicacao.class original", os três navegadores carregaram o arquivo de *bytecodes* que encontraram no seu sistema local de arquivos ("Aplicacao.class falso") em vez de realizarem a carga dos *bytecodes* no site alvo.

Antes de solicitar os *bytecodes* de uma classe ao servidor de onde foi carregada uma página html contendo um *applet*, a JVM implementada no navegador através do class loader primordial verificará se tal classe já não está localizada em seu *CLASSPATH* [ROU97]. O *CLASSPATH* é um diretório situado na máquina do usuário onde estão todas as classes "básicas" da linguagem Java, o chamado "Java Core".

Para adicionar uma classe nova ao *CLASSPATH* a única ação a ser feita é copiar o arquivo contendo os *bytecodes* da classe neste diretório. Esta facilidade pode ser bastante útil, pois se permite acrescentar classes na máquina do usuário, dispensando a busca através da rede destas classes, tornando mais rápida a execução dos *applets*.

Entretanto, esta confiança automática nos arquivos situados no *CLASSPATH* pode ser considerada temerária. Existem diversas formas de levar um usuário leigo a instalar um arquivo "Aplicacao.class falso" no *CLASSPATH* de seu navegador, principalmente se este não estiver familiarizado com os riscos que estão envolvidos. Nos experimentos descritos no capítulo 6 é criado um redirecionamento do navegador do usuário provocado por um arquivo *bytecode* mal-intencionado instalado no *CLASSPATH* do computador do usuário alvo.

O arquivo "Aplicacao.class falso" acaba se tornando um ataque extremamente bem dirigido, pois afetará apenas o site-alvo que referencia este arquivo de *bytecodes*, não afetando todos os demais sites que utilizam Java (e que não utilizem uma classe chamada "Aplicacao"). Apesar do usuário estar acessando a página html originada do site real, o *applet* que estará embutido e recebendo as suas informações não foi carregado do site original.

3.1 Possíveis Ataques

Três formas de ataque foram desenvolvidas para explorar esta característica de carregar a classe que foi instalada manualmente no *CLASSPATH* :

Bloqueio de acesso a serviço (denial of service) : foi criado um *applet* de mesmo nome do *applet* alvo que simplesmente não realiza a ação esperada, apesar de apresentar a mesma aparência do *applet* real

redirecionamento para um site falso : foi criado um *applet* que ao ser carregado redireciona o usuário para um site específico, que pode ser um site falso para gerar o ataque do gênero "man-in-the-middle", conforme descrito em [MR98].

---

[3] Bytecodes : código gerado pelo compilador Java, que será executado pela máquina virtual Java

retransmissão de informações : foi criado um *applet* que além de realizar o processamento esperado, retransmite informações obtidas para um servidor arbitrário. Por meio de ferramentas como decompiladores de *bytecodes* Java é possível obter um código fonte bastante próximo do original, o que permite para um programador mal-intencionado manter a lógica original do *applet* e acrescentar a retransmissão de informações obtidas para um servidor arbitrário, que esteja preparado para coletar estas informações.

3.2. A Solução

Uma solução possível para solucionar a vulnerabilidade descrita no capítulo anterior exigiria alterações na JVM, como é descrita nas conclusões deste artigo. Entretanto, não é razoável suspender a utilização da tecnologia Java enquanto se aguarda a próxima versão da máquina virtual.

Propomos a seguir uma abordagem capaz de neutralizar o ataque descrito sem que seja necessária alteração na máquina virtual Java dos navegadores de todos os usuários do sistema. A solução se baseia num mecanismo implementado no servidor web que fornece as páginas contendo os *applets* e atendem às requisições dos arquivos de *bytecodes* das classes Java.

A implantação de um arquivo de *bytecodes* falso no *CLASSPATH* do navegador do usuário-alvo também depende de um nome específico de *applet*. A solução se baseia em gerar aleatoriamente os nomes das classes *applets* utilizados no site a ser protegido, impossibilitando a detecção e desvio para o site falso pelo *applet* "espião" e inviabilizando a implantação de um arquivo de *bytecodes* falso na máquina do usuário [MW98].

Esta solução impede não apenas os ataques de monitoração por um *applet* espião e de implantação de classes suspeitas no *CLASSPATH*, mas também os ataque que eventualmente venham a ser criados que dependam do conhecimento prévio do nome de uma classe Java.

Para permitir a implantação desta solução algumas questões devem ser resolvidas :

geração dinâmica da página contendo o *applet*
determinação do nome da classe
adaptação dinâmica e envio do arquivo contendo os *bytecodes* (arquivo .class)

3.2.1. Geração Dinâmica da Página Contendo o *Applet*

Uma página html que apresenta um *applet* referencia o nome da classe que implementa este *applet*.

Como o nome da classe que implementa o *applet* da página deve ser aleatório, esta página não poderá ser mais uma página estática armazenada no servidor WWW. Ela deve passar a ser gerada dinamicamente através de algum mecanismo como scripts Perl, scripts ASP, ISAPI, NSAPI, servlets etc. No estudo realizado, foi utilizada a geração desta página através de ISAPI (Information Server API) no servidor Microsoft Internet Information Server.

Uma vez que a página contendo o *applet* tenha o seu código html definido, a ISAPI é codificada para que responda a uma determinada requisição recebida de uma navegador com o html da página, substituindo o nome da classe que implementa o *applet* pelo nome aleatório. O codebase (referência aos diretórios onde se localizam os arquivos contendo os *bytecodes* do *applet*) deve ser ajustado para que seja invocada a ISAPI quando o navegador tentar obter a classe que implementa o *applet*.

Os demais links do site também devem ser adaptados para ao referenciarem a página contendo o *applet* seja chamada a ISAPI.

O nome da classe deve seguir um algoritmo, conforme apresentado no próximo item.

3.2.2. Determinação do Nome da Classe

Obviamente as restrições quanto a formação dos nomes das classes da linguagem Java devem ser seguidas, como por exemplo, o nome de uma classe não pode começar por um número (caracteres de 0 a 9).

O navegador solicitará ao servidor que este envie o arquivo contendo os *bytecodes* da classe que implementa o *applet*. O navegador solicitará ao servidor um arquivo com a seguinte URL :

http:// <servidor> / <codebase> / <nome-da-classe>.class

Um servidor a ser protegido pode fornecer mais de um *applet* que seja potencial alvo de ataque, assim como um único *applet* pode exigir mais de uma classe para ser devidamente executado.

Como o servidor a ser protegido deverá fornecer mais de uma classe com nome alterado dinamicamente, é necessária uma forma de relacionar o novo nome da classe com o nome original.

Desta forma, o novo nome da classe deve ser determinado de tal forma que possua uma propriedade através da qual possa ser posteriormente relacionada com a classe original que o servidor deve enviar como resposta á requisição.

Sugere-se que o nome aleatório tenha o mesmo tamanho que o nome da classe original para facilitar a adaptação do arquivo de *bytecodes*, como será comentado no item 3.2.3.

3.2.2.1. Funções para Mapear Nomes Aleatórios às Classes Reais

O nome aleatório deve possuir a seguinte característica :

F (<nome-aleatório>) = indicador para classe original

Onde F deve ser uma função de n para 1, onde n são os inúmeros nomes aleatórios possíveis para uma classe original.

Existem inúmeras funções que podem ser adotadas para a determinação do nome aleatório da classe.

Módulo N Calculado sobre o Cheksum do Nome Aleatório

O checksum do nome aleatório pode ser calculado pela somatória do código ASCII dos caracteres que formam o nome da classe. Aplicando-se o módulo N sobre o resultado do checksum, chega-se a um resultado de 0 a (N – 1), o que permite ao servidor distinguir até N classes distintas que poderão ter o nome gerado aleatoriamente.

Por exemplo, suponha que o servidor deva fornecer 6 classes com o nome alterado. O processo de determinação do nome aleatório da primeira classe poderia ser o seguinte :

O valor de N cujo módulo será aplicado sobre o checksum deve ser maior que 6. Vamos supor que N seja 10.

Supondo que o nome aleatório desejado deva ter 8 caracteres, os 7 primeiros caracteres poderão ser sorteados entre os caracteres válidos para nomes de classes. O último caractere a ser escolhido deve ser tal que o módulo N aplicado sobre o checksum do nome resultante seja igual a 1 (número escolhido para representar a classe).

Passo 1 : sortear os 7 primeiros caracteres : abcdefg

Passo 2 : calcular o checksum obtido até então : ASCII(a) + ASCII (b) + ... + ASCII (f) + ASCII (g) = 61 + 62 + 63 + 64 + 65 + 66 + 67 = 448

Passo 3 : determinar o último caractere adequado :

- módulo 10 (448 + ASCII(último caractere)) = 1

- possíveis ASCII (último caractere) = 3, 13, 23, 33, 43, 53, 63 ...
- último caractere adequado : ASCII ("c") = 63

Passo 4 : nome adequado "abcdefgc"

Processo de determinação da classe original :

Passo 1 : nome da classe solicitada pelo navegador : "abcdefgc"

Passo 2 : determinar a classe a ser fornecida : ASCII (a) + ASCII (b) + ... + ASCII (f) + ASCII (g) + ASCII (c) = 61 + 62 + 63 + 64 + 65 + 66 + 67 + 63 = 511

Passo 3 : módulo 10 (511) = 1 (deve ser enviado o *applet* 1)

Para maior segurança do processo, o valor de N cujo módulo será aplicado sobre o checksum do nome da classe não deve ser um valor fixo, devendo ser alterada periodicamente e ser de conhecimento e controle apenas do sistema. O conhecimento total da regra e dos parâmetros envolvidos no processo de geração dos nomes das classes protegidas poderia eventualmente prejudicar o sucesso desta solução.

Tabela Associando Nomes Aleatórios às Classes Originais

Outra forma para determinar a associação entre nomes aleatórios às classes originais é por meio da utilização de tabelas relacionando o nome aleatório gerado às classes originais.

Nesta solução não é necessário que o nome aleatório siga uma determinada regra de formação, contanto que não seja um nome já atribuído a uma classe anteriormente e que ainda possa ser solicitada pelo navegador do usuário. A não repetição de um nome já utilizado anteriormente pode ser facilmente implementada por uma inspeção no conteúdo da tabela.

O tempo de manutenção de entradas nesta tabela deve ser tal que mantenha as informações da associação de um nome aleatório à classe original pelo menos até que o navegador realize a solicitação do arquivo *bytecode* da classe, quando a entrada pode ser retirada.

Esta característica permite que os registros antigos da tabela sejam periodicamente retirados, evitando o consumo excessivo de recursos.

A implementação poderia ser realizada por uma tabela armazenada em memória ou em uma base de dados.

Criptografia de Chave Assimétrica do Nome da Classe

Uma terceira forma de gerar uma associação entre o nome aleatório e a classe original seria adotar funções de criptografia com chaves assimétricas.

O nome aleatório seria formado pela seguinte expressão :

Nome aleatório = Criptografia chave pública (nome da classe original + seqüencial)

Ou

Nome aleatório = Criptografia chave pública (indicador da classe original + seqüencial)
O indicador da classe original poderia ser por exemplo um número que serve como referência.

A porção seqüencial acrescentada ao final garante que o resultado da criptografia com chave pública será sempre diferente, não se repetindo enquanto forem utilizados números diferentes (o que pode ser controlado pelo servidor).

O resultado da criptografia com chaves assimétricas em geral é um vetor binário cujo comprimento é função do tamanho das chaves utilizadas [KNU98].

Por exemplo, no algoritmo assimétrico RSA o resultado da criptografia é um vetor binário de comprimento igual ao menor múltiplo do tamanho das chaves que seja maior ou igual ao comprimento do texto criptografado. Caso se utilize um par de chaves RSA de 512 bits, o resultado da criptografia de até 512 bits de dados (64 bytes) terá também 512 bits (64 bytes). Este vetor deverá ser codificado de forma que possa ser representado pelos caracteres válidos para compor um nome de classe.

O processo de obtenção da classe original a partir da solicitação do navegador pode ser expressado da seguinte forma :

Classe original + seqüencial = Decriptografia com chave privada (nome aleatório)

O processo de criptografia com chave assimétrica pode ser utilizado em conjunto com a solução anterior de tabela associando o novo nome à classe original. O resultado da criptografia seria usado como o nome aleatório que será armazenado na tabela, evitando o processamento para descriptografar o nome da classe solicitada pelo navegador.

A desvantagem deste método é que o novo nome das classes são dependentes do tamanho do par de chaves utilizados no processo, o que pode dificultar o processo de geração dinâmica dos arquivos *bytecodes*, conforme descrito no item 3.2.3.

Outras Formas de Determinação do Nome Aleatório e Associação com a Classe Original
Uma grande quantidade de variações podem ser implementadas para a determinação do nome aleatório e da sua associação com a classe original.

Por exemplo, para contornar a característica da criptografia com chave assimétrica gerar nomes muito longos (dependendo do tamanho do par de chaves utilizado), o resultado da criptografia poderia ser utilizado como entrada para uma função que escolheria os bytes que formarão o nome aleatório de comprimento desejado.

O principal é que estas implementações mantenham a seguinte característica : o padrão de formação dos novos nomes de classe não deve ser previsível nem facilmente deduzível por entidades externas ao sistema.

3.2.3. Adaptação Dinâmica e Envio do Arquivo Contendo os *Bytecodes*

Ao determinar aleatoriamente o nome da classe, como o servidor irá atender o pedido pelo arquivo contendo os *bytecodes* desta classe ?

Uma possibilidade seria alterar o arquivo fonte, adaptando adequadamente o nome da classe para o nome aleatório, compilá-lo e disponibilizar o arquivo contendo os *bytecodes*. Todo esse processo realizado por demanda, a medida que os nomes aleatórios fossem sendo gerados.

Outra possibilidade seria compilar previamente uma quantidade considerável de classes mudando apenas os nomes. Os nomes "aleatórios" deverão ser sorteados entre estes arquivos previamente compilados.

Entretanto existe uma alternativa menos dispendiosa, que é apresentada a seguir :

Na estrutura de um arquivo *bytecode*, existe uma tabela de constantes (constant pool) onde são armazenadas entre outras informações as strings literais, valores de constantes, nomes de classes, interfaces, variáveis e métodos [VEN96]. A diferença entre dois arquivos *bytecodes* gerados a partir do mesmo código fonte no qual se alterou apenas o nome da classe basicamente são apenas as entradas na tabela de constantes que representam o nome da classe e os nomes dos métodos construtores (que tem o mesmo nome da classe). Portanto para se obter um arquivo de *bytecodes* válidos de uma classe da qual se alterou o nome basta trocar as entradas na tabela de constantes referentes ao nome da classe e o nome dos métodos construtores.

No formato de um arquivo contendo os *bytecodes* que definem uma classe, existe uma indicação do comprimento da tabela de constantes [LY00]. Para que não seja necessário corrigir também o tamanho

das estruturas internas afetadas além das referências à classe, basta que o novo nome da classe tenha o mesmo comprimento em bytes que o nome original.

Na implementação testada, adotou-se uma maneira simplificada para realizar esta adaptação dinâmica dos *bytecodes* das classes que passaram a ter o seu nome gerado aleatoriamente. Como se assumiu a restrição do nome aleatório ter exatamente o mesmo tamanho do nome "original" da classe, o processo de adequação das entradas da tabela de constantes se resume a simplesmente substituir todas as seqüências de bytes equivalentes ao nome original da classe pelo nome aleatório. Desta maneira evita-se o ajuste da entrada que indica o comprimento da tabela de constantes e uma interpretação mais detalhada da estrutura da seqüência de *bytecodes*.

Este método simplificado impõe uma restrição ao código fonte da classe Java original, que é a de não permitir que ocorra nenhuma outra substring igual ao nome da classe a ser substituída, a não ser na definição do nome da classe e de seus construtores. Por exemplo, caso o nome da classe original fosse "abc" e o nome aleatório passasse a ser "123", supondo que no código fonte houvesse uma referência a uma string literal "testeabc". Ao ser submetido ao método simplificado de adaptação dos *bytecodes*, no código resultante a string literal passaria a ser "teste123", pois este método aplica uma busca e substituição de todas as seqüências de bytes iguais ao novo nome da classe indistintamente, e não apenas daquelas referentes ao nome da classe e dos construtores.

Um outro problema que poderia surgir com este método simplificado seria a eventualidade de uma seqüência de *bytecodes* coincidir com o nome aleatório, o que em geral é pouco provável. Em todo caso, para uma solução mais robusta e sem estas restrições deve-se realizar esta adaptação dinâmica dos *bytecodes* através de um método que interprete a estrutura dos *bytecodes*, limitando as alterações apenas aos campos da estrutura do arquivo da classe Java que armazenam o nome da classe e o nome dos métodos construtores.

Desta maneira dispõe-se de um método para gerar os *bytecodes* correspondentes a uma classe com nome aleatório a partir dos *bytecodes* da classe original sem a necessidade de recompilar o código fonte.

Finalmente, esta adaptação dinâmica dos *bytecodes* pode ser realizada pela ISAPI (ou mecanismos equivalentes) de maneira que ao receber a solicitação de um arquivo de *bytecodes* Java com o nome aleatório, a partir deste nome aleatório possa ser determinado qual a classe "original" e enviar os *bytecodes* devidamente adaptados.

3.2.4. Estendendo a Troca para Classes Utilizadas Pelos *Applets*

Um *applet* pode utilizar classes adicionais especialmente desenvolvidas para uma determinada finalidade além da classe principal que o implementa. Estas classes também são requisitadas por demanda pelo navegador quando são referenciadas pelo *applet*, e são também possíveis alvos do ataque baseado na incorporação de classes no *CLASSPATH*.

O mesmo método apresentado neste capítulo pode ser utilizado para que as classes utilizadas pelo *applet* também tenham o seu nome alterado dinamicamente.

Na tabela de constantes de uma classe Java, existe uma estrutura na qual são relacionados os nomes das classes dos objetos que são utilizados.

Suponha que a classe que implementa o *applet* "abc" utilize também a classe "xyz" e que gostaríamos de proteger ambas do ataque de incorporação de classes no *CLASSPATH*.

O processo de proteção destas classes passa a ser o seguinte :

- navegador do usuário solicita página que contém o *applet* abc
- sistema gera dinamicamente a página que contém o *applet*, informando o nome aleatório da classe que implementa o *applet*
- navegador do usuário solicita o arquivo contendo os *bytecodes* que implementam o *applet* com nome aleatório

- sistema gera dinamicamente o arquivo contendo os *bytecodes* do *applet* abc, substituindo o nome da classe e o construtor pelo nome aleatório do *applet*. Adicionalmente, substitui a referência à classe xyz pelo nome aleatório da classe xyz
- navegador do usuário solicita o arquivo contendo os *bytecodes* com o nome alterado da classe xyz.
- sistema gera dinamicamente o arquivo contendo os *bytecodes* da classe xyz, substituindo o nome da classe e o nome dos métodos construtores pelo nome aleatório da classe xyz.
- navegador do usuário executa o *applet* utilizando as classes fornecidas pelo sistema proposto normalmente.

Este processo pode ser aplicado em outros níveis : o *applet* poderia utilizar mais de uma classe que se queira proteger, assim como a classe xyz utilizada pelo *applet* também poderia ela mesma utilizar outras classes que devam ser protegidas. O princípio básico é a substituição do nome da classe e dos nomes dos métodos construtores e das referências aos nomes das outras classes a serem protegidas na tabela de constantes do arquivo contendo os *bytecodes*.

3.3 Implementação da Solução Proposta

Nos testes da solução proposta, foi implementado o mecanismo com as seguintes características :

- ISAPI codificado em C para fornecer as páginas a serem protegidas e atender às requisições de *bytecodes* das classes com os nomes alterados

- código html das páginas a serem protegidas se encontram codificadas no fonte da ISAPI (são "hard-coded"), conforme comentado

- Utilização da função módulo N variável do checksum do nome aleatório para determinar a classe original, podendo fornecer até 6 classes distintas a serem protegidas

- Possibilidade de uma classe utilizar no máximo uma outra classe para ter o nome protegido

- nome das classes a serem protegidas, o diretório em que se encontram os arquivos contendo os *bytecodes* originais, assim como a informação se estas utilizam outras classes que devam ser protegidas são codificadas no fonte da ISAPI (são "hard-coded")

- Os arquivos originais contendo os *bytecodes* das classes a serem protegidas devem estar disponíveis num diretório comum acessível pela ISAPI.

Foi adotado o mecanismo módulo N do checksum do nome aleatório para determinar a associação com a classe original. O valor de N e o número que identifica cada classe original são definidos no instante da inicialização do sistema.

Utilizando o site-alvo desprovido da proteção de geração aleatória de nomes de classes, obtivemos o seguinte resultado com o ataque "Incorporação de Classes de Origem Suspeita no *CLASSPATH*" :

a) Bloqueio de acesso ao serviço
O navegador Microsoft Internet Explorer 5.5, Netscape Communicator 4.75 e Netscape 6 foram atingidos por este ataque : ao carregar a página alvo do ataque, no lugar da classe original do *applet* disponível no site real foi carregado a classe implantada no computador do usuário.

b) Redirecionamento para um site falso
O navegador Microsoft Internet Explorer 5.5, Netscape Communicator 4.75 e Netscape 6 foram atingidos por este ataque : ao carregar a página alvo do ataque foi ativada o *applet* implantado no computador do usuário, e num segundo momento a janela do navegador foi redirecionada para o site falso, podendo ser o início do ataque "man-in-the-middle".

c) Retransmissão de informações
O ataque foi bem sucedido com navegador Microsoft Internet Explorer 5.5. Ao carregar a página alvo do ataque, foi carregada a classe implantada no computador do usuário. As informações digitadas pelo usuário foram transmitidas para o servidor preparado para receber as informações e o processamento

normal da lógica do *applet* original foi realizada. Sem que o usuário percebesse, as informações foram replicadas para terceiros e o processamento original do *applet* foi realizado sem demonstrar algum sinal suspeito.

Com o Netscape Communicator 4.75, o ataque se converteu num bloqueio de acesso ao serviço : ao carregar a página alvo do ataque foi efetivamente carregada a classe implantada no computador do usuário, mas ao se tentar transmitir as informações obtidas do usuário para o servidor de coleta das informações, foi gerada uma exceção devido a tentativa de conexão de rede não permitida :

security.Couldn't connect to '132.147.160.190' with origin from 'local-*CLASSPATH*-classes'.

A implementação do Netscape Communicator 4.75 se apresentou mais rigorosa que o da Microsoft, evitando que fosse bem sucedida a tentativa de replicar informações obtidas do usuário sem que este percebesse.

O ataque foi totalmente bem sucedido com o Netscape 6. A partir desta versão, a Netscape está utilizando a máquina virtual Java desenvolvido pela própria Sun Microsystems, por meio do Java Remote Environment 1.3 que é instalado juntamente com o navegador. As configurações padrão de política de segurança que são utilizadas logo após a instalação do navegador permitiram a transmissão dos dados obtidos pelo usuário para o servidor de coleta e prosseguiu normalmente com o procedimento. Na versão 4.75 do Netscape Communicator é utilizada uma implementação de JVM da própria Netscape.

Utilizando o site-alvo devidamente protegido com a geração aleatória de nomes de classes, constatamos a imunidade do sistema. Todos os navegadores testados executaram corretamente os *bytecodes* com nomes de classes modificadas.

4. Conclusões

A principal contribuição deste trabalho foi a identificação da vulnerabilidade descrita no item 3, "Incorporação de Classes de Origem Suspeita no *CLASSPATH*" e da solução para evitá-la.

Esta vulnerabilidade continua presente nas versões recentes dos navegadores Internet, podendo ser explorada para afetar o acesso a sites específicos. A conseqüências dos possíveis ataques podem ser o impedimento do funcionamento de um *applet* específico, redirecionar o usuário para um site falso ou retransmitir informações fornecidas pelo usuário enquanto se prossegue na função original do *applet*.

A solução apresentada tem a vantagem de não depender de uma atualização da JVM dos navegadores dos usuários que acessam o site alvo em potencial, constituindo-se de um sistema que deve ser instalado apenas no servidor web a ser protegido.

Além da incorporação de classes de origem suspeita no *CLASSPATH*, o sistema proposto evita os ataques que eventualmente venham a ser criados que se baseiem no conhecimento prévio do nome de uma classe.

A seguir apresentamos sugestões de mudanças no modelo de segurança para eliminar esta vulnerabilidade.

O ataque baseado na adição de classes de origem duvidosa no *CLASSPATH* do usuário é decorrente em parte da política de segurança adotada. Nesta política, as classes instaladas na máquina do usuário são confiáveis, pois como "não são importadas [diretamente] de fora da organização, e são (em teoria) apenas instaladas por indivíduos confiáveis, estas aplicações Java não acrescentam novas preocupações quanto a segurança" [FM96]. Entretanto, esta abordagem transfere uma grande responsabilidade para o usuário, que em muitos casos pode ser leigo em questões técnicas.

Uma sugestão para dificultar a instalação de classes mal-intencionadas no *CLASSPATH* da máquina do usuário seria aceitar no *CLASSPATH* apenas classes digitalmente assinadas. O conceito de assinatura digital pressupõe que a entidade que irá gerar as classes deverá obter um certificado junto a uma autoridade certificadora (Certification Authority ou CA) reconhecida pela máquina virtual Java. No processo de obtenção do certificado, esta entidade deverá se identificar junto a CA. Desta maneira, os arquivos contendo classes terão sua integridade garantida e a origem claramente indicada [ITUT93].

Desta maneira, caso alguém queira gerar uma classe mal-intencionada para ser instalada na máquina de algum usuário desavisado, ele terá que obter um certificado válido de uma CA de confiança da máquina virtual Java, o que retira o conforto do anonimato de suas ações.

Outra sugestão, seria uma modificação no modelo de segurança, com a utilização do binômio (nome da classe, code-base da classe) para permitir a verificação se uma classe presente no *CLASSPATH* efetivamente pode ser utilizada em substituição à classe que seria obtida remotamente da rede. Desta forma, o usuário ou administrador do sistema terá certeza do escopo de utilização das classes que estão presentes em seu computador. Por exemplo, a instalação no *CLASSPATH* de uma classe assinada por uma entidade que o usuário não conheça e que deva atuar no code-base de uma instituição financeira levantaria imediatamente suspeitas.

O lançamento cada vez mais freqüente de novas versões de navegadores e novas funcionalidades incorporadas ao ambiente Java torna o teste sistemático das máquinas virtuais Java fundamental para detectar eventuais novas falhas de segurança decorrentes da nova implementação e constatar correções de problemas anteriores.
O modelo de segurança da linguagem Java não deve ser encarado como um modelo perfeito e intocável. Sugestões de aperfeiçoamento devem ser consideradas, discutidas e futuramente incorporadas para aumentar a confiança no ambiente de programação Java.

6. Bibliografia

[FBDW96] FELTEN, E. W.; BALFANZ, D.; DEAN, D.; WALLACH, D. S. Web Spoofing : An Internet Con Game. Dept. of Computer Science / Princeton University, 1996. (Technical Report 540-96) . Disponível em http://www.cs.princeton.edu/sip/pub/spoofing.html. Acessado em dezembro de 2000.

[FM96] FRITZINGER, J. S.; MUELLER, M. Java Security. Sun Microsystems Inc., 1996

[GMPS97] GONG, L.; MUELLER, M.; PRAFULLCHANDRA, H.; SCHEMERS, R. Going Beyond the Sandbox : An Overview of the New Security Architecture in the Java Development Kit 1.2. In: USENIX SYMPOSIUM ON THE INTERNET TECHNOLOGIES AND SYSTEMS, California. 1997. Proceedings. Monterey, California, s.ed., dec, 1997, n.p.

[ITUT93] ITUT-T Telecommunication Standardization Sector of ITU. X.509 Data Networks and Open System Communications - Directory, Novembro de 1993

[KNU98] KNUDSEN, J. Java Cryptography. O'Reilly & Associates Inc, 1998[LY00] LINDHOLM, T.; YELLIN, F. The Java Virtual Machine Specification. Disponível em http://java.sun.com/docs/books/vmspec. Acessado em janeiro de 2001.

[MDOY98] MACGREGOR, R.; DURBIN, D.; OWLETT, J.; YEOMANS, A. Java Network Security. Prentice Hall, 1998

[MW98] MATAYOSHI, C. M.; RUGGIERO, W. V. Modelo de Segurança da Linguagem Java. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 16., Rio de Janeiro, 1998. Anais. Rio de Janeiro, s.ed., mai. 1998. p. 577-595.

[MF99] MCGRAW, G.; FELTEN, E. W.. Securing Java – Gettubg Down to Business with Mobile Code. 2. ed. Wiley Computer Publishing, 1999

[OAK98] OAKS, S. Java Security. O'Reilly, 1998.

[ROU97] ROULO, M. Reduce the launch time of your *applets*: Store them on client machines. Java World, jun. 1997. Disponível em http://www.javaworld.com/jw-06-plugins.html. Acessado em dezembro de 2000.

[SUN00] Sun Microsystems, Inc. Chronology of security-related bugs and issues, November 11th 2000. Disponível http://java.sun.com/sfaq/chronology.html. Acessado em janeiro de 2001.

[VEN96] VENNERS, B. Under the Hood: The Java class file lifestyle - An introduction to the basic structure and lifestyle of the Java class file. Java World. jul. 1996. Disponível em http://www.javaworld.com/javaworld/jw-07-1996/jw-07-classfile.html. Acessado em junho de 2000.

BOLETINS TÉCNICOS - TEXTOS PUBLICADOS

BT/PCS/9301 - Interligação de Processadores através de Chaves Ômicron - GERALDO LINO DE CAMPOS, DEMI GETSCHKO

BT/PCS/9302 - Implementação de Transparência em Sistema Distribuído - LUÍSA YUMIKO AKAO, JOÃO JOSÉ NETO

BT/PCS/9303 - Desenvolvimento de Sistemas Especificados em SDL - SIDNEI H. TANO, SELMA S. S. MELNIKOFF

BT/PCS/9304 - Um Modelo Formal para Sistemas Digitais à Nível de Transferência de Registradores - JOSÉ EDUARDO MOREIRA, WILSON VICENTE RUGGIERO

BT/PCS/9305 - Uma Ferramenta para o Desenvolvimento de Protótipos de Programas Concorrentes - JORGE KINOSHITA, JOÃO JOSÉ NETO

BT/PCS/9306 - Uma Ferramenta de Monitoração para um Núcleo de Resolução Distribuída de Problemas Orientado a Objetos - JAIME SIMÃO SICHMAN, ELERI CARDOSO

BT/PCS/9307 - Uma Análise das Técnicas Reversíveis de Compressão de Dados - MÁRIO CESAR GOMES SEGURA, EDIT GRASSIANI LINO DE CAMPOS

BT/PCS/9308 - Proposta de Rede Digital de Sistemas Integrados para Navio - CESAR DE ALVARENGA JACOBY, MOACYR MARTUCCI JR.

BT/PCS/9309 - Sistemas UNIX para Tempo Real - PAULO CESAR CORIGLIANO, JOÃO JOSÉ NETO

BT/PCS/9310 - Projeto de uma Unidade de Matching Store baseada em Memória Paginada para uma Máquina Fluxo de Dados Distribuído - EDUARDO MARQUES, CLAUDIO KIRNER

BT/PCS/9401 - Implementação de Arquiteturas Abertas: Uma Aplicação na Automação da Manufatura - JORGE LUIS RISCO BECERRA, MOACYR MARTUCCI JR.

BT/PCS/9402 - Modelamento Geométrico usando do Operadores Topológicos de Euler - GERALDO MACIEL DA FONSECA, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9403 - Segmentação de Imagens aplicada a Reconhecimento Automático de Alvos - LEONCIO CLARO DE BARROS NETO, ANTONIO MARCOS DE AGUIRRA MASSOLA

BT/PCS/9404 - Metodologia e Ambiente para Reutilização de Software Baseado em Composição - LEONARDO PUJATTI, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9405 - Desenvolvimento de uma Solução para a Supervisão e Integração de Células de Manufatura Discreta - JOSÉ BENEDITO DE ALMEIDA, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/9406 - Método de Teste de Sincronização para Programas em ADA - EDUARDO T. MATSUDA, SELMA SHIN SHIMIZU MELNIKOFF

BT/PCS/9407 - Um Compilador Paralelizante com Detecção de Paralelismo na Linguagem Intermediária - HSUEH TSUNG HSIANG, LÍRIA MATSUMOTO SAITO

BT/PCS/9408 - Modelamento de Sistemas com Redes de Petri Interpretadas - CARLOS ALBERTO SANGIORGIO; WILSON V. RUGGIERO

BT/PCS/9501 - Sintese de Voz com Qualidade - EVANDRO BACCI GOUVÊA, GERALDO LINO DE CAMPOS

BT/PCS/9502 - Um Simulador de Arquiteturas de Computadores "A Computer Architecture Simulator" - CLAUDIO A. PRADO, WILSON V. RUGGIERO

BT/PCS/9503 - Simulador para Avaliação da Confiabilidade de Sistemas Redundantes com Reparo - ANDRÉA LUCIA BRAGA, FRANCISCO JOSÉ DE OLIVEIRA DIAS

BT/PCS/9504 - Projeto Conceitual e Projeto Básico do Nível de Coordenação de um Sistema Aberto de Automação, Utilizando Conceitos de Orientação a Objetos - NELSON TANOMARU, MOACYR MARTUCCI JUNIOR

BT/PCS/9505 - Uma Experiência no Gerenciamento da Produção de Software - RICARDO LUIS DE AZEVEDO DA ROCHA, JOÃO JOSÉ NETO

BT/PCS/9506 - MétodOO - Método de Desenvolvimento de Sistemas Orientado a Objetos: Uma Abordagem Integrada à Análise Estruturada e Redes de Petri - KECHI HIRAMA, SELMA SHIN SHIMIZU MELNIKOFF

BT/PCS/9601 - MOOPP: Uma Metodologia Orientada a Objetos para Desenvolvimento de Software para Processamento Paralelo - ELISA HATSUE MORIYA HUZITA, LÍRIA MATSUMOTO SATO

BT/PCS/9602 - Estudo do Espalhamento Brillouin Estimulado em Fibras Ópticas Monomodo - LUIS MEREGE SANCHES, CHARLES ARTUR SANTOS DE OLIVEIRA

BT/PCS/9603 - Programação Paralela com Variáveis Compartilhadas para Sistemas Distribuídos - LUCIANA BEZERRA ARANTES, LIRIA MATSUMOTO SATO

BT/PCS/9604 - Uma Metodologia de Projeto de Redes Locais - TEREZA CRISTINA MELO DE BRITO CARVALHO, WILSON VICENTE RUGGIERO

BT/PCS/9605 - Desenvolvimento de Sistema para Conversão de Textos em Fonemas no Idioma Português - DIMAS TREVIZAN CHBANE, GERALDO LINO DE CAMPOS

BT/PCS/9606 - Sincronização de Fluxos Multimídia em um Sistema de Videoconferência - EDUARDO S. C. TAKAHASHI, STEFANIA STIUBIENER

BT/PCS/9607 - A importância da Completeza na Especificação de Sistemas de Segurança - JOÃO BATISTA CAMARGO JÚNIOR, BENÍCIO JOSÉ DE SOUZA

BT/PCS/9608 - Uma Abordagem Paraconsistente Baseada em Lógica Evidencial para Tratar Exceções em Sistemas de Frames com Múltipla Herança - BRÁULIO COELHO ÁVILA, MÁRCIO RILLO

BT/PCS/9609 - Implementação de Engenharia Simultânea - MARCIO MOREIRA DA SILVA, MOACYR MARTUCCI JÚNIOR

BT/PCS/9610 - Statecharts Adaptativos - Um Exemplo de Aplicação do STAD - JORGE RADY DE ALMEIDA JUNIOR, JOÃO JOSÉ NETO

BT/PCS/9611 - Um Meta-Editor Dirigido por Sintaxe - MARGARETE KEIKO IWAI, JOÃO JOSÉ NETO

BT/PCS/9612 - Reutilização em Software Orientado a Objetos: Um Estudo Empírico para Analisar a Dificuldade de Localização e Entendimento de Classes - SELMA SHIN SHIMIZU MELNIKOFF, PEDRO ALEXANDRE DE OLIVEIRA GIOVANI

BT/PCS/9613 - Representação de Estruturas de Conhecimento em Sistemas de Banco de Dados - JUDITH PAVÓN MENDONZA, EDIT GRASSIANI LINO DE CAMPOS

BT/PCS/9701 - Uma Experiência na Construção de um Tradutor Inglês - Português - JORGE KINOSHITA, JOÃO JOSÉ NETO

BT/PCS/9702 - Combinando Análise de "Wavelet" e Análise Entrópica para Avaliar os Fenômenos de Difusão e Correlação - RUI CHUO HUEI CHIOU, MARIA ALICE G. V. FERREIRA

BT/PCS/9703 - Um Método para Desenvolvimento de Sistemas de Computacionais de Apoio a Projetos de Engenharia - JOSÉ EDUARDO ZINDEL DEBONI, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/9704 - O Sistema de Posicionamento Global (GPS) e suas Aplicações - SÉRGIO MIRANDA PAZ, CARLOS EDUARDO CUGNASCA

BT/PCS/9705 - METAMBI-OO - Um Ambiente de Apoio ao Aprendizado da Técnica Orientada a Objetos - JOÃO UMBERTO FURQUIM DE SOUZA, SELMA S. S. MELNIKOFF

BT/PCS/9706 - Um Ambiente Interativo para Visualização do Comportamento Dinâmico de Algoritmos - IZAURA CRISTINA ARAÚJO, JOÃO JOSÉ NETO

BT/PCS/9707 - Metodologia Orientada a Objetos e sua Aplicação em Sistemas de CAD Baseado em "Features" - CARLOS CÉSAR TANAKA, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9708 - Um Tutor Inteligente para Análise Orientada a Objetos - MARIA EMÍLIA GOMES SOBRAL, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/9709 - Metodologia para Seleção de Solução de Sistema de Aquisição de Dados para Aplicações de Pequeno Porte - MARCELO FINGUERMAN, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/9801 - Conexões Virtuais em Redes ATM e Escalabilidade de Sistemas de Transmissão de Dados sem Conexão - WAGNER LUIZ ZUCCHI, WILSON VICENTE RUGGIERO

BT/PCS/9802 - Estudo Comparativo dos Sistemas da Qualidade - EDISON SPINA, MOACYR MARTUCCI JR.

BT/PCS/9803 – The VIBRA Multi-Agent Architecture: Integrating Purposive Vision With Deliberative and Reactive Planning – REINALDO A. C. BIANCHI , ANNA H. REALI C. RILLO, LELIANE N. BARROS

BT/PCS/9901 – Metodologia ODP para o Desenvolvimento de Sistemas Abertos de Automação – JORGE LUIS RISCO BECCERRA, MOACYR MARTUCCI JUNIOR

BT/PCS/9902 – Especificação de Um Modelo de Dados Bitemporal Orientado a Objetos – SOLANGE NICE ALVES DE SOUZA, EDIT GRASSIANI LINO DE CAMPOS

BT/PCS/9903 – Implementação Paralela Distribuída da Dissecação Cartesiana Aninhada – HILTON GARCIA FERNANDES, LIRIA MATSUMOTO SATO

BT/PCS/9904 – Metodologia para Especificação e Implementação de Solução de Gerenciamento – SERGIO CLEMENTE, TEREZA CRISTINA MELO DE BRITO CARVALHO

BT/PCS/9905 – Modelagem de Ferramenta Hipermídia Aberta para a Produção de Tutoriais Interativos – LEILA HYODO, ROMERO TORI

BT/PCS/9906 – Métodos de Aplicações da Lógica Paraconsistente Anotada de Anotação com Dois Valores-LPA2v com Construção de Algoritmo e Implementação de Circuitos Eletrônicos – JOÃO I. DA SILVA FILHO, JAIR MINORO ABE

BT/PCS/9907 – Modelo Nebuloso de Confiabilidade Baseado no Modelo de Markov – PAULO SÉRGIO CUGNASCA, MARCO TÚLIO CARVALHO DE ANDRADE

BT/PCS/9908 – Uma Análise Comparativa do Fluxo de Mensagens entre os Modelos da Rede Contractual (RC) e Colisões Baseada em Dependências (CBD) – MÁRCIA ITO, JAIME SIMÃO SICHMAN

BT/PCS/9909 – Otimização de Processo de Inserção Automática de Componentes Eletrônicos Empregando a Técnica de Times Assíncronos – CESAR SCARPINI RABAK, JAIME SIMÃO SICHMAN

BT/PCS/9910 – MIISA – Uma Metodologia para Integração da Informação em Sistemas Abertos – HILDA CARVALHO DE OLIVEIRA, SELMA S. S. MELNICOFF

BT/PCS/9911 – Metodologia para Utilização de Componentes de Software: um estudo de Caso – KAZUTOSI TAKATA, SELMA S. S. MELNIKOFF

BT/PCS/0001 – Método para Engenharia de Requisitos Norteado por Necessidades de Informação – ARISTIDES NOVELLI FILHO, MARIA ALICE GRIGAS VARELLA FERREIRA

BT/PCS/0002 – Um Método de Escolha Automática de Soluções Usando Tecnologia Adaptativa – RICARDO LUIS DE AZEVEDO DA ROCHA, JOÃO JOSÉ NETO

BT/PCS/0101 – Gerenciamento Hierárquico de Falhas – JAMIL KALIL NAUFAL JR., JOÃO BATISTA CAMARGO JR.

BT/PCS/0102 – Um Método para a Construção de Analisadores Morfológicos, Aplicado à Língua Portuguesa, Baseado em Autômatos Adaptativos – CARLOS EDUARDO DANTAS DE MENEZES, JOÃO JOSÉ NETO

BT/PCS/0103 – Educação pela Web: Metodologia e Ferramenta de Elaboração de Cursos com Navegação Dinâmica – LUISA ALEYDA GARCIA GONZÁLEZ, WILSON VICENTE RUGGIERO

BT/PCS/0104 – O Desenvolvimento de Sistemas Baseados em Componentes a Partir da Visão de Objetos – RENATA EVANGELISTA ROMARIZ RECCO, JOÃO BATISTA CAMARGO JÚNIOR

BT/PCS/0105 – Introdução às Gramáticas Adaptativas – MARGARETE KEIKO IWAI, JOÃO JOSÉ NETO

BT/PCS/0106 – Automação dos Processos de Controle de Qualidade da Água e Esgoto em Laboratório de Controle Sanitário – JOSÉ BENEDITO DE ALMEIDA, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/01/07 – Um Mecanismo para Distribuição Segura de Vídeo MPEG – CÍNTIA BORGES MARGI, GRAÇA BESSAN, WILSON VICENTE RUGGIERO

BT/PCS/0108 – A Dependence-Based Model for Social Reasoning in Multi-Agent Systems – JAIME SIMÃO SICHMAN

BT/PCS/0109 – Ambiente Multilinguagem de Programação – Aspectos do Projeto e Implementação – APARECIDO VALDEMIR DE FREITAS, JOÃO JOSÉ NETO

BT/PCS/0110 – LETAC: Técnica para Análise de Tarefas e Especificação de Fluxo de Trabalho Cooperativo – MARCOS ROBERTO GREINER, LUCIA VILELA LEITE FILGUEIRAS

BT/PCS/0111 – Modelagem ODP para o Planejamento de Sistemas de Potência – ANIRIO SALLES FILHO, JOSÉ SIDNEI COLOMBO MARTINI

BT/PCS/0112 – Técnica para Ajuste dos Coeficientes de Quantização do Padrão MPEG em Tempo Real – REGINA M. SILVEIRA, WILSON V. RUGGIERO

BT/PCS/0113 – Segmentação de Imagens por Classificação de Cores: Uma Abordagem Neural – ALEXANDRE S. SIMÕES, ANNA REALI COSTA

BT/PCS/0114 - Uma Avaliação do Sistema DSM Nautilus –MARIO DONATO MARINO, GERALDO LINO DE CAMPOS

BT/PCS/0115 – Utilização de Redes Neurais Artificiais para Construção de Imagem em Câmara de Cintilação – LUIZ SÉRGIO DE SOUZA, EDITH RANZINI

BT/PCS/0116 – Simulação de Redes ATM – HSU CHIH WANG CHANG, WILSON VICENTE RUGGIERO

BT/PCS/0117 – Application of Monoprocessed Architecture for Safety Critical Control Systems – JOSÉ ANTONIO FONSECA, JORGE RADY DE ALMEIDA JR.

BT/PCS/0118 – WebBee – Um Sistema de Informação via WEB para Pesquisa de Abelhas sem Ferrão – RENATO SOUSA DA CUNHA, ANTONIO MOURA SARAIVA

BT/PCS/0119 – Parallel Processing Applied to Robot Manipulator Trajectory Planning – DENIS HAMILTON NOMIYAMA, LÍRIA MATSUMOTO SATO, ANDRÉ RIYUITI HIRAKAWA

BT/PCS/0120 – Utilização de Padrão de Arquitetura de Software para a Fase de Projeto Orientado a Objetos – CRISITINA MARIA FERREIRA DA SILVA, SELMA SHIN SHIMIZU MELNIKOFF

BT/PCS/0121 – Agilizando Aprendizagem por Reforço Através do uso de Conhecimento sobre o Domínio – RENÊ PEGORARO, ANNA H. REALI COSTA