# Detecção de códigos Java hostis na rede

Tantravahi Venkata Aditya[*], Antonio Montes[**]

Laboratório Associado de Computação e Matemática Aplicada
Instituto Nacional de Pesquisas Espaciais (INPE)

(*)Mestrado, Bolsa CAPES, e-mail: aditya@nucleo.inpe.br; (**) E-mail: montes@lac.inpe.br

**Resumo**

Este artigo propõe um método para a detecção de códigos Java hostis, visando uma maior segurança para usuários da Internet e de Java. Baseado neste método, é proposto o desenvolvimento de um módulo para ser incorporado num sistema de detecção de intrusão de redes.

*Palavras-Chave: Sistemas de detecção de intrusão, código Java hostil, segurança de Java*

## 1. Introdução

### 1.1. Problema da segurança de aplicações JAVA

A linguagem Java tem sido amplamente utilizada em vários tipos de aplicações principalmente para a Internet, além de ser uma linguagem multi–plataforma e por esta razão ser portável. Sua popularidade vem crescendo a cada ano e a tendência é que continue crescendo. Por isso, a segurança de Java tem se tornado um tema de grande importância.

Java foi criado com o intuito de resolver muitos problemas, um deles é a facilidade de programação em relação a outras linguagens, mas trouxe também vários problemas. Falhas na implementação de *browsers* que habilitam Java tem surgido com razoável freqüência. Essas falhas são tanto no âmbito do sistema, como no âmbito da aplicação. O resultado é uma série de vulnerabilidades que permitem que *applets* Java apaguem ou modifiquem arquivos, capturem informações importantes do sistema e corrompam o sistema do usuário.

A segurança do Java toma diferentes formas para diferentes usuários:

- **Usuários da Rede**: precisam conhecer os riscos aos quais estarão expostos ao deixar a opção Java habilitada em seus *browsers*.
- **Desenvolvedores de Aplicações Java**: os aplicativos para a Internet têm que ser projetados da maneira mais segura possível, visto que estes aplicativos vão estar acessando a máquina do usuário.
- **Administradores de Sistemas:** devem saber qual o impacto que Java terá em seus sistemas, e ter uma idéia exata dos prejuízos que o Java pode causar.

Muitos acreditam que a melhor solução é desabilitar o Java, mas existem aplicações onde o uso desta linguagem é viável e em muitos casos indispensável. Existem alternativas para estes casos, dentre elas a que é objeto deste trabalho que consiste em criar um detector para *applets* hostis baseado em vulnerabilidades conhecidas. Uma outra alternativa interessante seria bloquear os *applets* hostis no firewall[4].

Existem vários trabalhos sobre as vulnerabilidades do JAVA. Nas referências mencionadas no presente trabalho, são descritos vários tipos de vulnerabilidades e de *applets* maliciosos (discutido adiante). Assim, em [1] é mostrada uma tentativa de *DoS ( Denial of Service)* e descritas também as informações às quais os *applets* em JAVA tem acesso e outras vulnerabilidades que aparecem por falhas no DNS. Em [2] temos um exemplo de como se pode capturar informações importantes do sistema. Em [3] temos um outro tipo de *DoS*, e outros bons exemplos de vulnerabilidades. Em [5] temos um exemplo de uma vulnerabilidade que permite que se leia recursos protegidos do sistema de um usuário. E vários outras que podem ser encontrados em [11].

## 2. Fundamentação teórica

### 2.1. A linguagem Java

O Java é uma linguagem simples, distribuída, portável, e interpretada. Estas características resultaram em uma série de vulnerabilidades comprometendo a robustez da linguagem. Uma quantidade razoável dessas vulnerabilidades é dependente da arquitetura.

Para permitir que o Java seja executado é necessário que o *browser* da máquina esteja habilitado para isso. O código Java é executado da seguinte forma: o usuário baixa um arquivo que contém um Java *bytecode*, que em seguida é executado por uma máquina virtual Java que está instalada na máquina do usuário. Os programas Java que são executados em *browsers* são conhecidos como *applets*, que foram especialmente desenvolvidos para trabalhar com páginas de Internet.

Os *applets* podem ser considerados um sistema cliente/servidor tradicional da seguinte forma, o *Web Server* é o servidor do *applet*. Ele envia o *applet* para a máquina do usuário, onde ele será executado como um cliente.

### 2.2. Sandbox, a caixa de contenção do Java

O propósito da caixa de contenção do Java é fornecer um ambiente onde um programa possa ser executado, mas restringindo o acesso aos recursos do sistema. Esta caixa é quem dita as regras do modelo de segurança do Java. A caixa de contenção não é um modelo totalmente definido, visto que em alguns casos haverá a necessidade de dar privilégios a alguns applets. Por exemplo, é possível que em alguns casos os *applets* precisem acessar alguns arquivos de seus sistema. A caixa de contenção *default* pode ser dividida em três partes: o verificador de *bytecodes*, o carregador de classes, e o gerenciador de segurança.

### 2.3. Applets Hostis

São aqueles que prejudicam o sistema local de alguma forma. Existem dois tipos de *applets* hostis: os maliciosos e os de ataque.

Os *applets* maliciosos são aqueles que acessam o sistema local e provocam problemas considerados de baixa gravidade, podendo, por exemplo, forjar o correio eletrônico, roubar ciclos da CPU para realizar trabalhos pessoais ou sobrecarregar o sistema local usando o máximo possível de recursos.

Os *applets* maliciosos são encontrados na rede bem mais freqüentemente que os *applets* de ataque, estes últimos são mais perigosos e também mais difíceis de serem implementados. Os *applets* de ataque são aqueles que usam vulnerabilidades do JVM, ou da caixa de contenção, ou seja, vulnerabilidades no modelo de segurança. Estas incluem códigos corrompidos que ignoram o verificador, tentativas de forjar carregadores de classes, tentativas de comprometer o gerenciador de segurança, tentativas de infringir regras de *firewalls,* em resumo explorar erros de implementação. Pode-se até conseguir o controle com privilégios de administrador de uma máquina remota. Existe uma boa quantidade desses *applet*, felizmente poucos deles tem saído do laboratório onde foram criados.

## 3. Metodologia e Recursos

### 3.1. Sistemas de detecção de intrusão

Intrusão pode ser definida como o uso indevido ou impróprio de um sistema computacional. Detecção de intrusão é uma tecnologia de segurança que permite identificar e isolar intrusões em computadores ou redes. Na atualidade detecção de intrusão é uma ferramenta muito importante para a segurança de redes, mas não é a única e nem é o bastante. São necessárias outras ferramentas para garantir um bom nível de segurança, como por exemplo um *firewall*.

Sistemas de Detecção de Intrusão em Redes (*SDIR*), são sistemas que monitoram o tráfego de rede e permitem detectar determinados tipos de intrusões, analisando padrões suspeitos de atividade na rede.

### 3.2. Criação de um filtro para o SNORT

O SNORT [8] é um detector de intrusão, que realiza análise de tráfego em tempo real e registra pacotes trafegando sobre redes IP usando uma linguagem de descrição de regras que é simples, flexível e poderosa. Ele funciona em qualquer sistema que tenha o *libpcap* (ferramenta que permite fazer uma análise de trafego de rede em baixo nível). O SNORT registra pacotes no formato binário do *tcpdump* ou em seu próprio formato ASCII.

Propõe-se usar o SNORT para analisar o tráfego na rede fazendo uma varredura na porta 80 e em todas as outras que comunicam pelos protocolos *http* e *https*, capturando todos os arquivos de extensão *.class* e lendo o código binário deste arquivo. Em seguida será feita uma comparação com um conjunto de padrões de códigos hostis em Java conhecidos. Caso algum desses padrões coincida com algum código hostil Java conhecido o pacote será registrado e a execução deste código poderá ser interrompida.

Espera-se ao fim deste trabalho desenvolver um módulo confiável que detecte o código Java hostil de forma eficiente e eficaz, que seja o mais robusto possível.

## 4. Referências Bibliográficas

[1] Dean, D. ; Felten, E. W. ; Wallach D. S. ; *Java security: From Hotjava to Netscape and Beyond, IEEE Symposium on Security and Privacy, 1996*

[2] Oaks, S.; *Segurança de dados em JAVA , Editor ciência Moderna,1999*

[3] Felten, E. W.; *Securing Java, www.securingjava.com, 1999*
[4] Martin Jr, D. M., Rajagopalan , S., Rubin D. A.; *Blocking Java Applets at the Firewall, 1997*
[5] CERT Advisories, www.cert.org/advisories/CA-2000-15.html , 2000
[6] Writing Snort Rules, www.snort.org
[7] Cornell, G; Horstmann, C. S. ; Core Java 2, Prentice Hall, 1999
[8] von Doorn, L. ; *A Secure Java Virtual Machine,* 9th USENIX Security Symposium, 2000
[9] Roberts, S; *Complete Java 2 Certification Guide,* Sybex , 2000
[10] Harold, E. R. , Java Network Programming, O'Reilly, Second Edition, 2000
[11]Hostile Java www.megasecurity.org/Hostile_java