Sistema de Detecção de Intrusões com Utilização de Redes Neurais

Flávio José Bianchetti[1], Michelli Marlane da Silva[1]

[1]Universidade Presidente Antônio Carlos – UNIPAC
Rua Palma Bageto Viol, s/n – CEP 36200-108 – Campolide – MG – Brasil
flavio.bianchetti@hotmail.com, michelli_marlane@hotmail.com

Abstract: This article aims at presenting an Intrusion Detection system called Neuro-One using Artificial Neural Networks. It also describes the operation of an Artificial Neural Network with Backpropagation algorithm using the problems encountered during development and testing performed. Although not produced satisfactory results, this article examines in detail the decisions taken to design the system, leaving its contents for future reference.

Resumo: Este artigo tem por objetivo apresentação de um Sistema de Detecção de Intrusões chamado Neuro-One que utiliza as Redes Neurais Artificiais. Descreve também o funcionamento de uma Rede Neural Artificial com utilização do algoritmo Backpropagation, os problemas encontrados durante o desenvolvimento e os testes efetuados. Apesar de não ter produzido resultados satisfatórios, este artigo retrata com detalhes as decisões tomadas para a concepção do sistema, deixando seu conteúdo para futuras referências.

1. Introdução

Com o constante aumento das informações e a grande difusão da internet, o termo *segurança* se tornou algo altamente necessário nos dias de hoje. A segurança das informações que trafegam em um computador conectado à internet e como garantir a segurança destas informações tanto dentro quanto fora dele, são questões que ainda geram muitas polêmicas. Existem muitas soluções já desenvolvidas, como por exemplo, o uso de criptografia para a segurança das informações, equipamentos que garantem proteger a rede de intrusos indesejados (como o firewall, por exemplo) e também dos chamados sistemas de detecção de intrusão (*IDS – Intrusion Detection System*)[5][6][7][8].

O *IDS* é um programa que faz a verificação do conteúdo dos pacotes que trafegam na rede, classificando-os como normal ou anômalo. Um *IDS* bem elaborado e configurado pode ajudar na detecção de inconsistências existentes na configuração do sistema e indícios de ataques e de pré-ataques[5][6].

Redes Neurais Artificiais (*RNA*) é um assunto antigo, porém, atualmente está bastante inserido em diversos setores de pesquisa para a resolução de problemas. Ela consiste em atribuir à máquina a capacidade de tomar decisões para a finalidade a qual foi concebida. A proposta de agregar ao *IDS* as *RNA* é permitir que após cada conjunto

de informações verificadas, o próprio sistema tenha condições de classificar a possível ameaça, e se errar, aprender com seus erros[5][6][7][8].

Este tema não é novo e já foi bastante discutido, porém existem problemas não solucionados, como por exemplo, na união destas técnicas é observado a ocorrência de muitos alarmes falsos, chamados de Falsos Positivos e Falsos Negativos[5][6][7][8]. A proposta deste artigo é a apresentação do desenvolvimento de um sistema de detecção de intrusões chamado de *Neuro-One* que utiliza técnicas de *RNA* para a detecção de anomalias em um computador ligado à internet. O foco deste trabalho é que o sistema *Neuro-One* consiga aprender com seus erros durante a fase de treinamento com o objetivo de diminuir o número de alarmes falsos de seus resultados quando comparado a outros sistemas já desenvolvidos.

Na segunda seção é descrito o funcionamento básico de um sistema de detecção de intrusões; na terceira seção temos a descrição básica de uma rede neural artificial; a quarta seção traz os experimentos efetuados; a quinta seção exibe algumas conclusões sobre o tema e a sexta seção descreve a bibliografia utilizada.

2. Detecção de Intrusão

2.1. O Sistema de Detecção de Intrusão (*IDS*)

Um *IDS* é um aplicativo que se caracteriza pelo escaneamento do trânsito de pacotes na rede em busca de comportamento estranho que podem permitir uma pessoa não autorizada ter acesso privilegiado ao sistema[4].

Um bom *IDS* deve monitorar as atividades do usuário e do sistema, avaliando sua integridade e dados de arquivos. Deve também reconhecer padrões de atividades que refletem ataques conhecidos, responder automaticamente à atividade detectada e reportar o resultados do processo de detecção. Normalmente, os *IDS* são divididos em duas grandes classes: os baseados em assinaturas e os baseados em anomalias[4][5].

Os *IDS* baseados em assinaturas tem a função de identificar ataques que seguem padrões já registrados em sua base de conhecimento. Porém esta técnica impossibilita a identificação de novos tipos de ataques ou variações de ataques já conhecidos, até que eles tenham sido incluídos na base de conhecimento[5].

Nos *IDS* baseados em anomalias são registrados o comportamento normal do sistema onde o *IDS* está implantado. Desta forma qualquer comportamento fora dos padrões registrados, o sistema gera uma sinalização anômala. Um ponto negativo deste método é que o número de alarmes falsos encontrados são maiores do que os encontrados nos baseados por assinaturas[5].

Uma questão importante para os *IDS* baseados em anomalias é a definição de como o sistema deverá aprender a distinguir o comportamento anômalo do comportamento normal. Para isso utilizaremos as técnicas de Redes Neurais Artificiais, descritas na próxima seção[5].

2.2. A Estrutura de um Pacote TCP/IP

As informações que trafegam pela rede são difundidas por meio de pacotes. A *Figura 1* exibe a estrutura de um pacote rede. Podemos comparar as informações de cada pacote recebido com as informações catalogadas e averiguar se caracteriza invasão ou não[3].

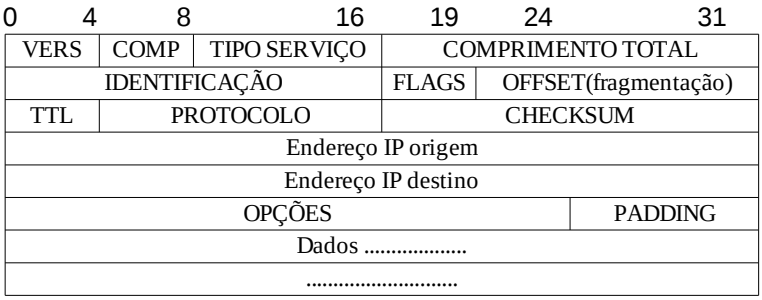

Figura 1 – Estrutura do pacote de rede
Fonte: [3]

2.3. IDS Baseado em Host

Os sistemas de detecção de intrusões podem ser classificados em três categorias, de acordo com a sua funcionalidade: Baseado em Rede (*NIDS*), Baseado em Host (*HIDS*) e Distribuído (*DIDS*). O sistema proposto neste trabalho adota o paradigma *HIDS*. A característica principal de um *IDS* Baseado em Host é que ele tem o objetivo de proteger somente o sistema onde se encontra instalado[4].

Uma das vantagens desta técnica é permitir uma definição de um conjunto de regras específicas para cada máquina existente em um rede. Desta forma, podemos configurar cada um dos *IDS* de uma rede para se adequar somente aos tipos de detecção que sejam pertinentes, e não adotar o modelo para toda a rede[4].

Possui também desvantagens, como por exemplo, quando o sistema é implantado em um host, ele adiciona uma carga adicional à máquina. Normalmente, não deve ser um problema quando instalado em um desktop de usuário, mas pode se tornar um grande problema quando instalado em um servidor. Outro problema é o desafio encontrado na manutenção de uma grande rede com muitos *HIDS* instalados. Sem uma gerência centralizada, o administrador pode perder muito tempo tentando acompanhar os alertas fornecidos pela rede[4].

3. Redes Neurais

Segundo [1], a primeira vez que cientistas reuniram-se em um esforço conjunto para estudar Inteligência Artificial (*IA*) foi no encontro no "*Darthmouth College*" em 1956 e, logo depois, no mesmo ano, foi publicado o livro com o título "*Automata Studies*" contendo o primeiro artigo de redes neurais como paradigma da arquitetura computacional.

As redes neurais aplicadas na arquitetura computacional consiste em, basicamente, atribuir à máquina a capacidade de tomar decisões. Consegue-se isto simulando o funcionamento de um neurônio do sistema biológico, apresentado na

Figura 2. Segundo [2], é recebido um estímulo pelos dendritos e enviado ao corpo para processamento. Após o processamento, o novo estímulo é enviado ao axônio, sendo repassado para os próximos neurônios conectados.

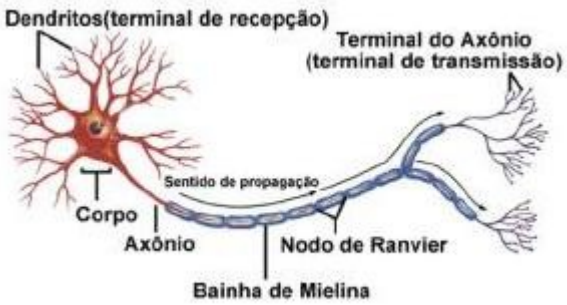

Figura 2 – Neurônio biológico

O funcionamento do neurônio artificial segue o mesmo paradigma do neurônio biológico. A *Figura 3* exibe um neurônio artificial onde os terminais de Entradas (*dendritos*) recebem os valores de x0, x1 e x2. Para simular o comportamento das sinapses (*estímulos*) são representados pelos pesos wj0, wj1,wj2. Cada entrada é multiplicada com o seu respectivo peso. Os valores são submetidos a uma Função de Soma ponderada que, se atingir um determinado valor, ativam a Função de Transferência e a Saída (*Axônio*)[1].

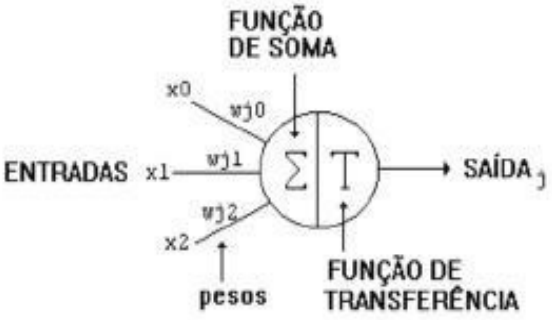

Figura 3 – Neurônio artificial

3.1. Backpropagation

O algoritmo *Backpropagation* utiliza técnicas de gradiente descendente interativo para minimizar uma função de custo igual à diferença da média quadrada entre a saída desejada e a saída real da *RNA*, em outras palavras, possui uma característica de permitir a modificação dos pesos das conexões das camadas internas em caso de erro em relação à saída desejada[4][13]. A *Figura 4* mostra o exemplo da rede *Backpropagation*.

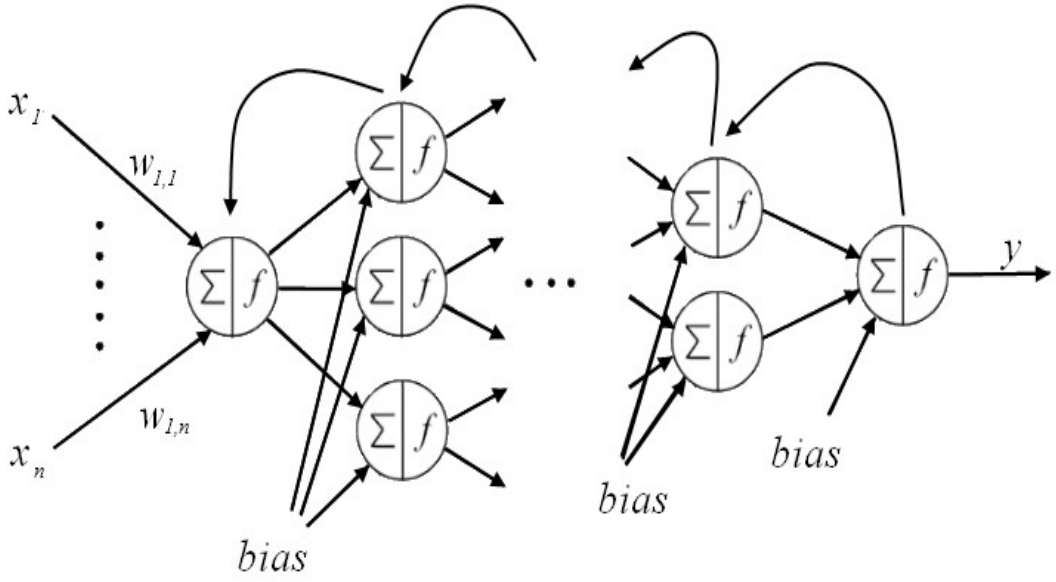

Figura 4 – Exemplo de uma rede Backpropagation

Uma rede *Backpropagation* utiliza “n” entradas, “m” saídas e possui “o” camadas ocultas. Inicialmente são atribuídos valores entre -1 e 1 a todos os pesos de entrada da rede neural (w_n) e para o Bias (b). O chamado Bias é um valor atribuído para aumentar os graus de liberdade de uma *RNA* melhorando a adaptação ao conhecimento adquirido[1][15].

Em seguida são apresentados os valores de entrada (x_n) para o treinamento da rede neural. Cada um dos valores é multiplicado pelos seus respectivos pesos. Depois é feita uma soma destes valores, adicionando também o Bias[1][15]:

$$u = \sum_{n=1}^{i} (x_n w_n) + b$$

O valor adquirido através da soma das entradas mais o Bias (u) é aplicado a uma Função Sigmoide (f) de acordo com a fórmula abaixo[1][15]:

$$f(u) = \frac{1}{1 - e^{-u}}$$

O valor adquirido pela Função Sigmoide é o valor de saída encontrado pelo neurônio. Este processo é propagado pela rede até que seja alcançado um resultado na camada de saída[1][15].

Durante o processo de treinamento a camada de saída produz um valor encontrado pela rede (y_j) para um determinado conjunto de entradas. Este valor é subtraído do valor desejado para a saída do treinamento (y'_j). O novo resultado é multiplicado pela primeira Derivada da Função Sigmoide ($f'(u)_j$) resultando em um valor de Erro (ξ_j)[1][15].

$$\xi_j = (y'_j - y_j) * f'(u)_j$$

O valor do Erro encontrado é retropropagado ajustando os pesos da camada de saída até os pesos da camada de entrada. Após os ajustes, uma nova entrada é apresentada à rede. O processo se repete até que o erro encontrado seja mínimo, resultando assim em uma aprendizagem de toda a rede neural[1][15].

4. Experimentos

Utilizando as técnicas de *RNA* aliadas a um *IDS* foi desenvolvido um sistema chamado de *Nero-One* com a utilização da linguagem de programação *JAVA*. É composto em sua primeira versão de dois módulos básicos: O Sistema de Treinamento da Rede Neural e o Sistema de Detecção de Intrusões.

4.1. O Equipamento Utilizado

Tanto para o desenvolvimento quanto para o treinamento foi utilizado um notebook POSITIVO com processador INTEL® Core™ 2 DUO com 1.73 Ghz, 1 GB de memória física, sistema operacional SLACKWARE na versão 13.0, ambiente gráfico KDE na versão 4.4.3 e NETBEANS IDE 6.9.

4.2. O Código Fonte Desenvolvido

Visando acelerar o desenvolvimento do sistema *Neuro-One* foi tomado como base de estudo o código do programa criado por [16] que utilizava as redes neurais para identificação de padrões em figuras. O programa desenvolvido possui poucos traços do código utilizado como base, sendo atribuídos os referidos créditos ao autor.

4.3. O Sistema de Treinamento da Rede Neural

Basicamente, o Sistema de Treinamento da Rede Neural desenvolvido retira as informações existentes nos arquivos de treinamento, entregando-as para a rede que fará as operações necessárias para o ajuste dos pesos. A *Figura 5* exibe a tela de interface com o usuário para treinamento da rede neural.

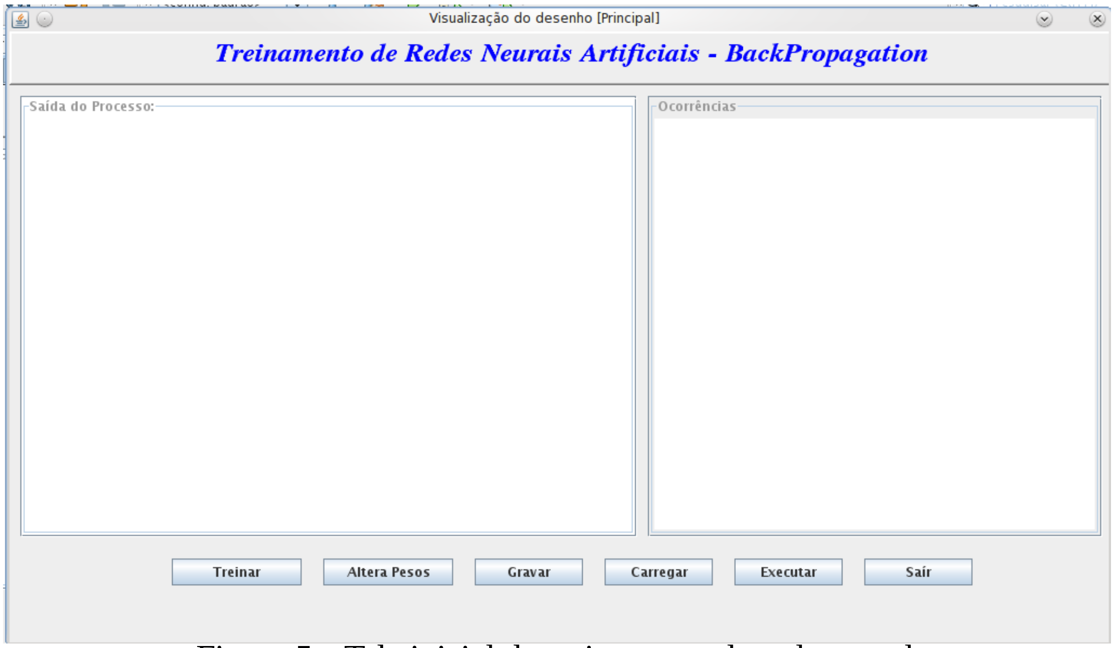

Figura 5 – Tela inicial do treinamento da rede neural

4.3.1. O Modelo dos Arquivos Utilizados

Para o treinamento da rede neural será adotada a base de dados utilizada na competição de anual *KDD CUP* de 1999, organizada pela *ACM Special Interest Group on Knowledge Discovery and Data Mining*. A base de dados adotada é um fragmento de nove semanas de captura de pacotes, com 311.029 registros de conexão[12]. A *Figura 6* exibe uma parte do conteúdo do arquivo de base de dados.

```
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,normal.
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,normal.
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,normal.
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,snmpgetattac
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.01,0.00,0.00,0.00,0.00,0.00,snmpgetattac
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.00,0.01,0.00,0.00,0.00,0.00,0.00,snmpgetattac
0,udp,domain_u,SF,29,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,1,0.00,0.00,0.00,0.00,0.50,1.00,0.00,10,3,0.30,0.30,0.30,0.00,0.00,0.00,0.00,0.00,normal.
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,253,0.99,0.01,0.00,0.00,0.00,0.00,0.00,0.00,normal.
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,snmpgetattac
0,tcp,http,SF,223,185,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,4,4,0.00,0.00,0.00,0.00,1.00,0.00,0.00,71,255,1.00,0.00,0.01,0.01,0.00,0.00,0.00,0.00,normal.
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.00,0.00,0.00,0.00,0.00,0.00,snmpgetattac
0,tcp,http,SF,230,260,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,19,0.00,0.00,0.00,0.00,1.00,0.00,0.11,3,255,1.00,0.00,0.33,0.07,0.33,0.00,0.00,0.00,normal.
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,254,1.00,0.01,0.01,0.00,0.00,0.00,0.00,0.00,normal.
0,udp,private,SF,105,146,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,252,0.99,0.01,0.00,0.00,0.00,0.00,0.00,0.00,snmpgetattac
1,tcp,smtp,SF,3170,329,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,2,0.00,0.00,0.00,0.00,1.00,0.00,1.00,54,39,0.72,0.11,0.02,0.00,0.02,0.00,0.09,0.13,normal.
0,tcp,http,SF,297,13787,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,177,255,1.00,0.00,0.01,0.01,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,291,3542,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,12,12,0.00,0.00,0.00,0.00,1.00,0.00,0.00,187,255,1.00,0.00,0.01,0.01,0.00,0.00,0.00,0.00,normal.
```

Figura 6 – Arquivo de base de dados

Os arquivos de treinamento fornecidos possuem 42 informações de cada registro divididas em duas classes: 41 informações retiradas dos fluxos de rede e 1 informação sobre a classificação das informações dos referidos fluxos. A *Tabela 1* exibe uma descrição das 41 informações retiradas de cada registro.

Tabela 1 – descrição das entradas

Nº	Campo	Tipo	Nº	Campo	Tipo
1	duration	continuous	22	is_guest_login	symbolic
2	protocol_type	symbolic	23	count	continuous
3	service	symbolic	24	srv_count	continuous
4	flag	symbolic	25	serror_rate	continuous
5	src_bytes	continuous	26	srv_serror_rate	continuous
6	dst_bytes	continuous	27	rerror_rate	continuous
7	land	symbolic	28	srv_rerror_rate	continuous
8	wrong_fragment	continuous	29	same_srv_rate	continuous
9	urgent	continuous	30	diff_srv_rate	continuous
10	hot	continuous	31	srv_diff_host_rate	continuous
11	num_failed_logins	continuous	32	dst_host_count	continuous
12	logged_in	symbolic	33	dst_host_srv_count	continuous
13	num_compromised	continuous	34	dst_host_same_srv_rate	continuous
14	root_shell	continuous	35	dst_host_diff_srv_rate	continuous
15	su_attempted	continuous	36	dst_host_same_src_port_rate	continuous
16	num_root	continuous	37	dst_host_srv_diff_host_rate	continuous
17	num_file_creations	continuous	38	dst_host_serror_rate	continuous
18	num_shells	continuous	39	dst_host_srv_serror_rate	continuous
19	num_access_files	continuous	40	dst_host_rerror_rate	continuous
20	num_outbound_cmds	continuous	41	dst_host_srv_rerror_rate	continuous
21	is_host_login	symbolic			

A base de dados utilizada para treinamento e testes possui um total de 24 tipos de ataques divididos em quatro classes, conforme a *Tabela 2*. As classes de ataques estão divididas em:

- U2R: acesso não autorizado aos privilégios de superusuário (*root*) locais;
- R2L: acesso não autorizado originado de um computador remoto;
- PROBE: varreduras de portas e outras sondagens; e
- DoS: negação de serviço.

Tabela 2 – descrição dos tipos de ataques

U2R	R2L	PROBE	DoS
buffer_overflow	ftp_write	ipsweep	back
loadmodule	guess_passwd	nmap	land
perl	imap	portsweep	neptune
rootkit	multihop	satan	pod
	phf		smurf
	spy		teardrop
	warezclient		
	warezmaster		

4.3.2. O Modelo de Rede Neural Adotado

A *RNA* adotada foi dividida em três grupos básicos: Camada de Entrada, Camadas Escondidas ou Ocultas e Camada de Saída.

A Camada de Entrada é composta pelas informações retiradas dos pacotes de rede que serão apresentadas à rede neural para o treinamento. As Camadas Escondidas ou Ocultas são camadas intermediárias com neurônios com a função de extrair informações das amostras. A Camada de Saída retorna a resultado encontrado pelo treinamento da rede neural[4].

Cada uma destas camadas pode possuir vários neurônios, conforme a necessidade do problema a ser resolvido. Inicialmente foi adotado seguinte modelo:

- 1 neurônio na primeira camada recebendo 46 entradas de informações do arquivo de treinamento;
- 5 camadas ocultas com 10 neurônios em cada uma; e
- 1 camada de saída com 1 neurônio.

De acordo com a divisão dos tipos de ataques em quatro classes foi proposto que o modelo de treinamento e o modelo de detecção de intrusões possuísse um conjunto de 5 redes neurais, um para cada tipo de conjunto de anomalias mais a classe de detecções classificadas como normais.

4.4. O Sistema de Detecção de Intrusões Neuro-One

4.4.1. O Módulo de Captura de Pacotes

Foi desenvolvido um módulo de captura de pacotes utilizando o ambiente de programação *JAVA* para atender as especificações do projeto. A *Figura 7* exibe a tela de interface com o usuário. Nela é escolhido o dispositivo de comunicação que fará a captura dos pacotes da rede para a detecção de anomalias.

Figura 7 – Tela inicial do IDS

4.4.2. A Biblioteca JPCAP

Os pacotes são capturados com ajuda da biblioteca *JPCAP* (*JAVA Package for Packet Capture*), desenvolvida para o ambiente *JAVA*. Esta biblioteca é composta de vários utilitários que auxiliam no envio e recebimento de pacotes na rede de computadores[10] [11].

A *JPCAP* pode ser utilizada em vários tipos de sistemas operacionais, dentre eles o Windows, com a utilização da ferramenta *WinPCAP*, e o Linux, com a ferramenta *LibPCAP*[11].

JPCAP suporta os protocolos Ethernet, *IPV4, IPV6, ARP/RARP, TCP, UDP* e *ICMPV4*. Outros tipos de pacotes são capturados como os pacotes primitivos, que contém todos os dados dos pacotes. Isto permite que aplicações do *JAVA* analisem pacotes com tipos não suportáveis[11].

4.5. Resultados

Para o treinamento e testes da rede neural foram utilizados os seguintes arquivos e configurações:

- a taxa de aprendizado da rede foi definida entre o intervalo 0.2 e 0.4;
- o número de épocas de treinamento foi definida entre o intervalo de 3000 e 10.000;
- o arquivo *corrected* foi utilizado para treinamento e testes da rede neural. Ele foi dividido em dois arquivos, com respectivamente, 217.720 para utilização de treinamento e 93.309 para a realização dos testes.

Durante o período de testes foi observado o real aprendizado da rede neural. O ajuste dos pesos da rede é feito confrontando a saída desejada com a saída encontrada, conforme o exibido pelo gráfico da *Figura 8*.

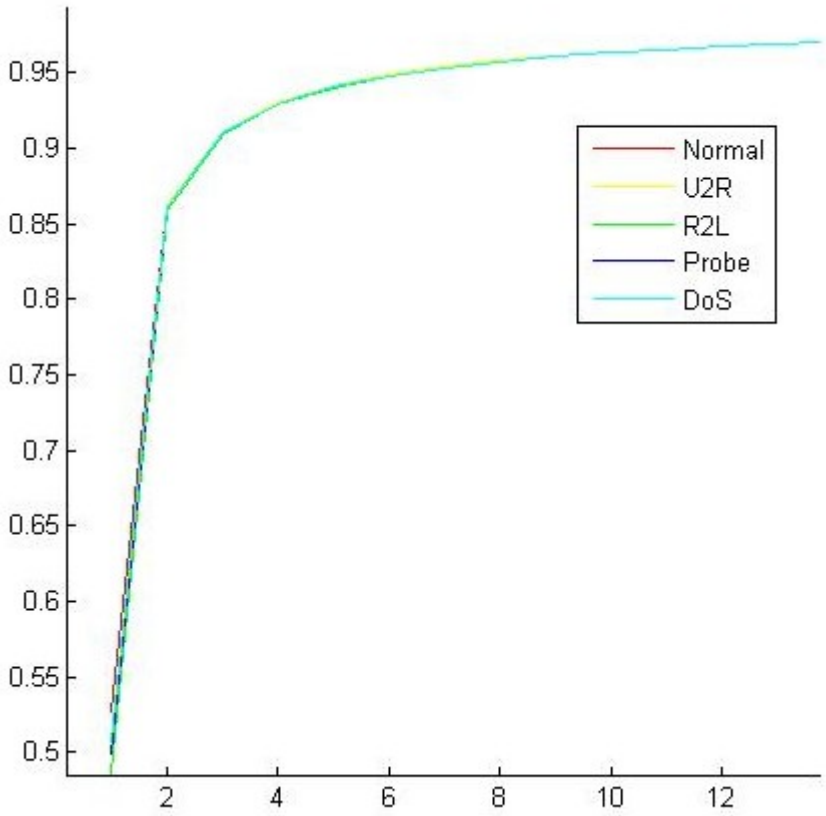

Figura 8 – Gráfico de treinamento da rede neural

Foram encontrados diversos problemas, principalmente no formato da rede neural e na base de dados para treinamento, obrigando mudanças na concepção do sistema proposto.

Utilizando a base de dados para treinamento e testes fornecida pela *KDD CUP* observou-se que a rede neural não produzia aprendizado. Sempre que a rede era submetida aos testes, todas as entradas apresentadas resultavam em apenas uma das classes de ataques.

De acordo com o retratado por [4] acreditava-se que o erro do treinamento estava na base de dados de entrada, sendo necessária a geração de entropia. Para isso, ele dividiu as 64 informações do campo *service* do arquivo utilizado no treinamento em 7 campos. Foi seguido o mesmo paradigma utilizado, apenas alterando o número de campos para 6 com o objetivo de confrontar os resultados alcançados nos dois paradigmas.

Visando aumentar o número de acertos adquiridos pela rede neural, toda a base de dados de treinamento foi transformada em números do tipo *double* e foi feita a sua normalização dentro do intervalo 0.0 e 1.0 [14]. Para a normalização foi utilizada a fórmula:

$$V = \frac{V_i - V_{min}}{V_{max} - V_{min}}$$

onde V_i é o valor atual da coluna a ser convertida; V_{min} e V_{max} são, respectivamente, o menor valor e o maior valor da coluna.

Após efetuadas as modificações anteriormente citadas observou-se que nos novos testes a rede ainda continuava a classificar como saída apenas uma das classes de ataques, obrigando a efetuar alterações no número de neurônios nas camadas escondidas.

Segundo [14], não é recomendada a utilização de um grande número de camadas escondidas. Utilizando muitas unidades pode levar a rede a memorizar os dados de treinamento (overfitting) ao invés de extrair as características gerais que permitirão a generalização. Utilizando poucas unidades faz com que a rede gaste tempo em excesso tentando encontrar uma representação ótima.

Ainda, segundo [14], existem várias propostas de como determinar a quantidade adequada de neurônios nas camadas escondidas de uma rede neural. As mais utilizadas são:

- Definir o número de neurônios em função da dimensão das camadas de entrada e saída da rede. Pode-se definir o número de neurônios na camada escondida como sendo a mé dia aritmética ou ainda como sendo a mé dia geométrica entre tamanho da entrada e da saída da rede[14].
- Utilizar um número de sinapses dez vezes menor que o número de exemplos disponíveis para treinamento. Se o número de exemplos for muito maior que o número de sinapses, overfitting é improvável, no entanto pode ocorrer underfitting (a rede não converge durante o seu treinamento)[14].

Utilizando este novo paradigma a rede neural foi ajustada para a seguinte configuração:

- 1 neurônio na primeira camada recebendo 46 entradas de informações do arquivo de treinamento;
- 2 camadas ocultas com 5 neurônios em cada uma; e
- 1 camada de saída com 1 neurônio.

Durante os novos testes observou-se que a rede começou a produzir resultados, porém o número de erros encontrados estava mais alto do que o número de acertos, conforme a *Tabela 3*.

Tabela 3 – resultado do primeiro teste encontrado

	Total de Registros no Arquivo	Registros Encontrados pelo Teste	Acertos	Erros
Normal	18208	10713	8750	1963
U2R	64	17789	32	17757
R2L	4361	801	185	616
PROBE	1250	63638	190	63448
DoS	69419	368	36	332

Desta forma foram feitos testes variando empiricamente o número de camadas escondidas e o número de neurônios existentes nestas camadas de forma a encontrar uma configuração adequada que produza as saídas desejadas. A melhor saída encontrada durante o período de treinamento utilizou a seguinte configuração:

- 5 neurônios na primeira camada recebendo 46 entradas de informações do arquivo de treinamento;
- 2 camadas ocultas com, respectivamente, 15 e 5 neurônios;
- 1 camada de saída com 1 neurônio; e
- Foram retirados 636 registros aleatórios de cada classe do arquivo utilizado no treinamento para a geração de um novo arquivo de treino.

Os resultados estão descritos na *Tabela 4*.

Tabela 4 – resultado do melhor teste encontrado.

	Total de Registros	Registros Encontrados	Acertos	Erros
Normal	18208	30899	15149	15750
U2R	64	7772	35	7737
R2L	4361	9886	1088	8798
PROBE	1250	46	45	1
DoS	69419	44706	44328	378

4.6 Trabalhos Futuros

De acordo com os resultados obtidos, sugere-se como trabalhos futuros os seguintes tópicos abaixo relacionados:

- atribuir diferentes números de camadas ocultas e número de neurônios nestas camadas para cada tipo de classe de ataques;
- incluir variáveis adicionais na rede neural, como por exemplo, momento e elasticidade, com o intuito de refinar a detecção das classes.
- efetuar testes com outras bases de dados, gerando entropias e normalizações mais eficientes.
- alterar o paradigma do sistema desenvolvido permitindo que ele possa aprender com seus erros em tempo real.
- elaborar estudo de caso sobre o software desenvolvido, sugerindo alterações para que se torne mais eficiente.
- incluir outras técnicas de redes neurais para confrontação de resultados e análise de suas eficiências.
- atribuir ao sistema a autonomia para executar bloqueios, efetuar contra-ataques e conseguir a localização do atacante.

5. Conclusão

Mesmo não encontrando resultados satisfatórios as técnicas de RNA demonstraram ser um grande artifício no aprendizado de sistemas computacionais. Quando corretamente configurada a rede aprende a distinguir as várias entradas convergindo para a saída desejada. Aliada a um sistemas de detecção de intrusões esta técnica pode oferecer um grande potencial para a detecção de anomalias em uma rede, tomando as decisões automaticamente e deixando usuários que porventura venham a utilizar do sistema livres para outras tarefas cotidianas.

Os resultados alcançados demonstram que a concepção de um sistema com este paradigma é muito difícil. A escolha do número de camadas escondidas e dos neurônios nestas camadas deve ser tomada com bastante cuidado para que o aprendizado da rede seja feito com sucesso. Também foi comprovada a importância da normalização da base de dados para uma maior eficiência dos resultados da rede.

6. Bibliografia

[1] BARRETO, Jorge M. Introdução às Redes Neurais, 2002. Disponível em: <www.inf.ufsc.br/~barreto/tutoriais/Survey.pdf > . Acesso em: 26 Fev. 2010.

[2] BRAGA, Antônio de Pádua; CARVALHO, André Ponce de Leon F. de; LAUDEMIR, Teresa Bernarda. Redes Neurais Artificiais – Teoria e Aplicações. LTC – Livros Técnicos e Científicos Editora S.A., 2007.

[3] FERRAZ, Tatiana Lopes; ALBUQUERQUE, Marcelo Portes, ALBUQUERQUE, Marcio Portes. Introdução ao Ping e Traceroute, 2002. Disponível em: <www.rederio.br/downloads/pdf/nt01002.pdf>. Acesso em: 7 jan. 2010.

[4] NETTO, Roberto Silva. Detecção de Intrusão Utilizando Redes Neurais Artificiais no Reconhecimento de Padrões de Ataques, 2006. Disponível em: <*http://adm-net-a.unifei.edu.br/phl/pdf/0031732.pdf*>. Acesso em: 28 ago. 2010.

[5] MAFRA, Paulo M.; FRAGA, Joni da Silva, MOLL, Vinícius; SANTIN, Altair Olivo. POLVO-IIDS: Um Sistema de Detecção de Intrusão Inteligente Baseado em Anomalias, 2008. Disponível em: <*sbseg2008.inf.ufrgs.br/resources/slides/ST2/apr_st02_04_artigo.pdf*>. Acesso em: 8 mai. 2010.

[6] JUNIOR, Arnaldo Cândido; CANSIAM, Adriano Mauro; SAUDE, Almir Moreira. Uso de Redes Neurais para Detecção de Anomalias em Fluxos de Dados, 2005. Disponível em: <*angel.acmesecurity.org/~adriano/papers/acme-artigo-sige-2005-arnalmadr.pdf*>. Acesso em: 8 mai. 2010.

[7] LIMA, Igor Vinícius Mussoi de. Uma Abordagem Simplificada de Detecção de Intrusão Baseada em Redes Neurais Artificiais, 2005. Disponível em: <*http://sbseg2008.inf.ufrgs.br/proceedings/data/pdf/st01_02_resumo.pdf*>. Acesso em: 8 mai. 2010.

[8] PISANI, Paulo Henrique; PEREIRA, Silvio do Lago. Detecção de Intrusões Baseada em User Profiling e Redes Neurais, 2009. Disponível em: <*bt.fatecsp.br/arquivos/bt_26/artigo79.pdf*>. Acesso em: 8 mai. 2010.

[9] Mecatrônica Atual – Automação Manual de Processos e Manufatura. Disponível em: <http://www.mecatronicaatual.com.br/secoes/leitura/553>. Acesso em 08 Abr. 2010.

[10] ESPÍNDULA, Maicon Machado de. Medição do Tráfego de Rede Baseado em Fluxos com JPCAP. Disponível em: <www.ulbra.tche.br/tc2/Maicon%20Machado%20de%20Espindula.pdf >. Acesso em: 21 jan. 2010.

[11] BRAGANÇA, Wanderson Carvalho. Sistema para Gerenciamento de Redes Baseado em Agentes Móveis, 2009. Disponível em: <http://www.ic.uff.br/~viviane.silva/2009.1/isma/util/trab1/trab1_wanderson.pdf >. Acesso em: 04 set. 2010.

[12] KDD CUP - ACM Special Interest Group on Knowledge Discovery and Data Mining, 1999. Disponível em: <http://www.sigkdd.org/kddcup/index.php?section=1999&method=info>. Acesso em: 13 ago. 2010.

[13] SILVA, Lilian de Sá. Uma Metodologia para Detecção de Ataques no Tráfego de Redes Baseada em Redes Neurais, 2008. Disponível em <http://mtc-m17.sid.inpe.br/col/sid.inpe.br/mtc-m17@80/2007/07.31.12.49/doc/publicacao.pdf>. Acesso em 08 mai. 2010.

[14] SILVA, Eugênio; OLIVEIRA, Anderson Canêdo de. Dicas para Configuração de Redes Neurais. Disponível em <http://equipe.nce.ufrj.br/thome/grad/nn/mat_didatico/dicas_configuracao_rna.pdf>. Acesso em 20 nov. 2010.

[15] TATIBANA, Cassia Yuri; KAETSU, Deisi Yuri. Uma Introdução às Redes Neurais. Disponível em <http://www.din.uem.br/ia/neurais/>. Acesso em 15 out. 2010.

[16] SZABÓ, Richárd. A Backpropagation Implementation in Java. Disponível em <http://www.freeweb.hu/jataka/rics/>. Acesso em 15 fev. 2010.