



**UNIVERSIDADE DO SUL DE SANTA CATARINA**  
**RÔMULLO FURTADO BELTRAME**

**SISTEMA DE DETECÇÃO DE INTRUSÃO EM REDES UTILIZANDO**  
**REDES NEURAIIS**

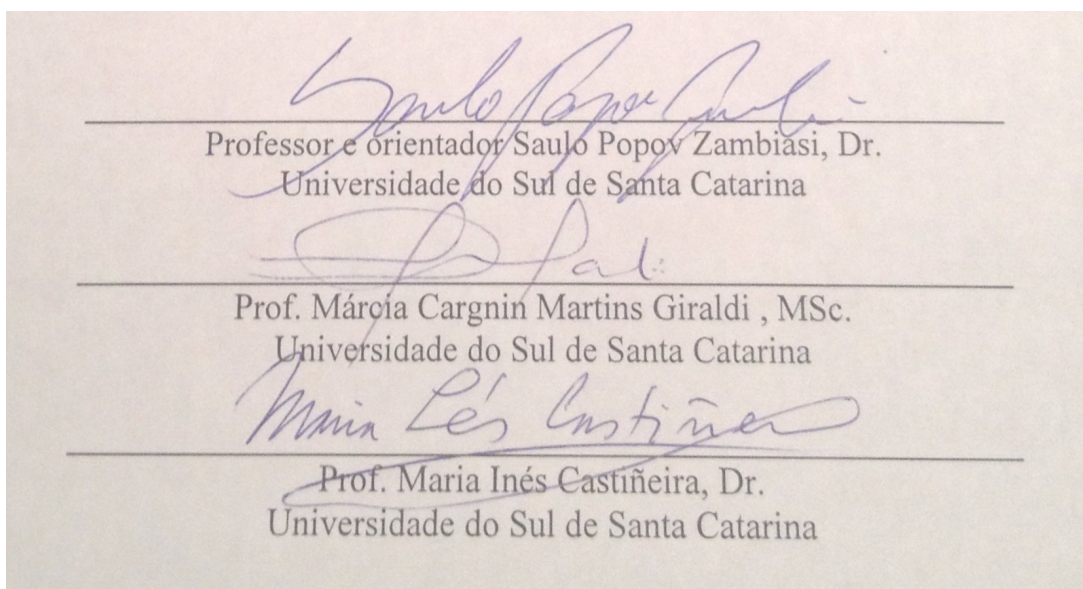
Florianópolis  
2015

**RÔMULLO FURTADO BELTRAME**

**SISTEMA DE DETECÇÃO DE INTRUSÃO EM REDES UTILIZANDO  
INTELIGÊNCIA ARTIFICIAL**

Este Trabalho de Conclusão de Curso foi julgado adequado à obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Graduação em Ciência da Computação da Universidade do Sul de Santa Catarina.

Florianópolis, 15 de Junho de 2015.



\_\_\_\_\_  
Professor e orientador Saulo Popov Zambiasi, Dr.  
Universidade do Sul de Santa Catarina

\_\_\_\_\_  
Prof. Márcia Cargnin Martins Giraldi , MSc.  
Universidade do Sul de Santa Catarina

\_\_\_\_\_  
Prof. Maria Inés Castiñeira, Dr.  
Universidade do Sul de Santa Catarina

Dedico este trabalho aos meus pais que sempre me deram seu melhor. Aos meus professores e mestres, cuja vocação é a mais bela deste universo. Aos meus amigos, que são minha família e meu alicerce. E, sobretudo, dedico este trabalho a todos que buscam o conhecimento com sabedoria, não há limites para o ser livre e sábio.

## **AGRADECIMENTOS**

Agradeço aos meus amigos R.K e aos Deuses que estiveram comigo por toda essa jornada e não deixaram sequer um segundo de me apoiar.

Aos meus pais, que sempre deram o melhor de si e o que há de melhor em mim agradeço a vocês.

A todos os professores, desde meus primeiros anos de vida até os aqui presente. Minha eterna gratidão àqueles que não medem esforços em ensinar e, muitas vezes, educar, vocês têm de mim a mais pura admiração por tão bela jornada e vocação.

Ao professor Saulo Z. Popov, meu orientador e professor paciente, que tão honestamente acreditou no potencial deste projeto e empreitou-o comigo.

Aos amigos que fiz na faculdade, Matheus e Douglas, que tão bravamente me aturaram em momentos de grande nervosismo, os DDG serão memoráveis sempre.

Ao meu caro amigo e chefe, Douglas, pelo apoio incondicional nos sonhos mais loucos e por topar grandes aventuras, inclusive empreender.

Agradeço à UNISUL e ao Governo Federal, por me concederem a possibilidade de estudar sem custos e abrindo oportunidade a tantos outros alunos bolsistas de estudar e graduar-se, buscando sempre evoluir.

“A educação é a arma mais poderosa que você pode usar para mudar o mundo”.

(Nelson Mandela, 2003)

## RESUMO

Este trabalho propõe uma solução para a problemática de detecção de intrusão em sistemas informatizados, utilizando a tecnologia de Redes Neurais Artificiais para generalizar informações e classificá-las de acordo com o risco de eminência de um ataque ou invasão. A revisão bibliográfica apresenta os principais conceitos e estudos relacionados aos assuntos no projeto abordados, possibilitando, assim, o desenvolvimento de um software que uniu os pontos positivos de diversas tecnologias a fim de melhor desenvolver a detecção de intrusão. Este trabalho se enquadra como pesquisa aplicada, exploratória, quantitativa e bibliográfica. Foi desenvolvido um software que monitora a rede a fim de encontrar padrões nos ataques, e detecta-os mesmo que estes ataques sejam novos. Os resultados obtidos com o protótipo chegaram a um sucesso de 99,79% de acerto, conseguindo detectar os *exploits* ensinados e alguns termos considerados de risco, considerados assim altamente satisfatórios e mostrando-se eficazes na detecção de intrusão.

Palavras-chave: Sistema de Detecção de Intrusão. Inteligência Artificial. SNORT. Network intrusion detection system. Invasão de sistemas. Inteligência artificial. Redes neurais artificiais. Python.

## **ABSTRACT**

This work presents a solution for intrusion detection problems in computer systems, using Artificial Neural Networks technology to generalize information and classify them according to the risk verge of an attack or invasion. It is proposed that these structures monitor the network in order to find patterns in the attacks, it can even detect new attacks. The literature review presents the main concepts and studies related to the issues addressed in the project, thus enabling the development of a software that united the strengths of various technologies in order to improve the intrusion detection today. This work fits as applied research, exploratory, qualitative and literature. The results obtained with the prototype were highly satisfactory, it reached a successful 99.78% accuracy, shown itself effective in detecting intrusion through payload analysis considering the trained exploits and the list of suspect terms.

**Keywords:** Intrusion Detection System. Artificial Intelligence. Neural Networks. SNORT. Network intrusion detection system. System Invasion. Python.

## LISTA DE ILUSTRAÇÕES

FIGURA 1 - FLUXO DA INFORMAÇÃO EM UM NEURÔNIO BIOLÓGICO.....	20
FIGURA 2 - ESTRUTURA DO NEURÔNIO ARTIFICIAL.....	22
FIGURA 3 - ESTRUTURA DA REDE MLP.....	25
FIGURA 4 - ARQUITETURA DE UM ATAQUE MITM.....	29
FIGURA 5 - ESTRUTURA DE UMA REDE COM IDS (NIDS).....	32
FIGURA 6 - ESTRUTURA DE FLUXO DE UM PACOTE A SER ANALISADO.....	35
FIGURA 7 - ESTRUTURA DE UMA REGRA DO SNORT (RULE).....	35
FIGURA 8 - EXEMPLOS DE REGRAS DO SNORT.....	36
FIGURA 9 - ARQUITETURA DA SOLUÇÃO PROPOSTA.....	40
FIGURA 10 - TOPOLOGIA DA RNA.....	44
FIGURA 11 - TOPOLOGIA E DISTRIBUIÇÃO DA CAMADA DE ENTRADA.....	45
FIGURA 12 - EXEMPLO DE NEURÔNIO DE TREINAMENTO.....	48
FIGURA 13 - FLUXOGRAMA DO MÓDULO DE TRATAMENTO DE DADOS.....	53
FIGURA 14 - PADRÃO DE LOGS.....	54
FIGURA 15 - EXEMPLO DE USO DO SISTEMA.....	57
FIGURA 16 - ARQUITETURA DO CENÁRIO DE VALIDAÇÃO.....	58



## LISTA DE QUADROS

Quadro 1: Modelos de redes neurais.....	24
Quadro 2: Pontos Positivos e Negativos do SNORT.....	34
Quadro 3: Modelos de redes neurais.....	49
Quadro 4: Dados do primeiro cenário de Avaliação do sistema.....	59
Quadro 5: Dados do segundo cenário de Avaliação do sistema.....	60
Quadro 6: Cronograma – 2014/2015.....	67

## LISTA DE ABREVIACÕES

<i>AM</i> .....	<i>APRENDIZADO DE MÁQUINA</i>
<i>IA</i> .....	<i>INTELIGÊNCIA ARTIFICIAL</i>
<i>IDS</i> .....	<i>INTRUSION DETECTION SYSTEM</i>
<i>I-NIDS</i> .....	<i>INTELLIGENT NETWORK INTRUSION DETECTION SYSTEM</i>
<i>HIDS</i> .....	<i>HOST INTRUSION DETECTION SYSTEM</i>
<i>MLP</i> .....	<i>MULTILAYER PERCEPTRON</i>
<i>NIDS</i> .....	<i>NETWORK INTRUSION DETECTION SYSTEM</i>
<i>NIDS</i> .....	<i>SISTEMA DE DETECÇÃO DE INTRUSÃO EM REDES</i>
<i>RNA</i> .....	<i>REDE NEURAL ARTIFICIAL</i>
<i>NNIDS</i> .....	<i>NEURAL NETWORK INTRUSION DETECTION SYSTEM</i>

## SUMÁRIO

<b>1 INTRODUÇÃO.....</b>	<b>13</b>
1.1 PROBLEMÁTICA.....	14
1.2 OBJETIVOS.....	14
1.2.1 Objetivo Geral.....	15
1.2.2 Objetivos Específicos.....	15
1.3 JUSTIFICATIVA.....	15
1.4 ESTRUTURA MONOGRÁFICA.....	16
<b>2 REVISÃO BIBLIOGRÁFICA.....</b>	<b>17</b>
2.1 INTELIGÊNCIA ARTIFICIAL.....	17
2.1.1 Agindo de forma humana: o teste de Turing.....	17
2.1.2 Subdivisões e áreas específicas.....	18
2.2 REDES NEURAIS ARTIFICIAIS.....	19
2.2.1 Estrutura.....	20
2.2.2 Neurônio de McCulloch e Pitts.....	22
2.2.3 Estudo dos Algoritmos.....	23
2.2.4 Perceptron Multicamadas ou Backpropagation.....	24
2.2.5 Momentum.....	26
2.2.6 Função de Ativação.....	26
2.3 SEGURANÇA DA INFORMAÇÃO.....	27
2.3.1 Tipos de ameaças mais comuns.....	28
2.3.2 Meios comuns de proteção.....	30
2.3.3 Sistemas de Detecção de intrusão (IDS).....	31
2.3.4 Sistemas de Detecção de intrusão em Redes (NIDS).....	32
2.4 SNORT.....	33
2.4.1 Características.....	33
2.4.2 Estrutura funcional.....	34
2.5 TRABALHOS CORRELATOS.....	36
<b>3 MÉTODO.....</b>	<b>38</b>
3.1 CARACTERIZAÇÃO DO TIPO DE PESQUISA.....	38
3.2 ETAPAS METODOLÓGICAS.....	38
3.3 SOLUÇÃO PROPOSTA.....	39
3.3.1 1ª Fase – Entrada dos dados.....	40
3.3.2 2ª Fase – Processamento de dados.....	40
3.3.3 3ª Etapa – Criação do Vetor de Entrada da RNA.....	41
3.3.4 4ª Fase – Classificação.....	41
3.3.5 5ª Fase – Definição dos limites de similaridade.....	41
3.3.6 6ª Fase – Gerar log.....	42
3.4 DELIMITAÇÕES.....	42
<b>4 DESENVOLVIMENTO.....</b>	<b>43</b>
4.1 DEFINIÇÃO DA ESTRUTURA DA REDE NEURAL.....	43
4.1.1 Estrutura e Topologia da RNA.....	44
4.1.2 Estrutura da camada de entrada.....	45
4.1.3 Aprendizado.....	46
4.1.4 Biblioteca PYBRAIN.....	47
4.2 DEFINIÇÃO DA BASE DE CONHECIMENTO PARA TREINAMENTO.....	47
4.2.1 Arquivo da Base de Conhecimento.....	48
4.2.2 Coleta dos dados Maliciosos.....	48

4.2.3 Lista de Palavras-chave.....	49
4.3 MÓDULO DE TRATAMENTO CAPTURA DE DADOS OU SNIFFER.....	50
4.3.1 Função sniff.....	50
4.3.2 Método Pkt_callback.....	51
4.4 MÓDULO DE TRATAMENTO DE DADOS.....	51
4.4.1 Conversão do Payload.....	51
4.4.2 Extração de informações do pacote.....	52
4.4.3 Fluxograma do módulo de tratamento de dados.....	53
4.5 MÓDULO DE CLASSIFICAÇÃO.....	53
4.6 MODELO DE SAÍDA DE DADOS.....	54
4.6.1 Padrão de log.....	54
4.7 TECNOLOGIAS UTILIZADAS.....	55
4.7.1 Python.....	55
4.7.2 Bibliotecas de apoio.....	55
4.7.3 Requisitos do sistema.....	56
4.8 APRESENTAÇÃO DO SISTEMA.....	57
4.8.1 Exemplos de inicialização.....	57
4.9 VALIDAÇÃO.....	58
4.9.1 Primeiro Cenário.....	59
4.9.2 Segundo Cenário.....	59
5 CONCLUSÕES E TRABALHOS FUTUROS.....	61
5.1 TRABALHOS FUTUROS.....	61

## 1 INTRODUÇÃO

Desde o início da utilização de computadores e equipamentos informatizados em ambientes empresariais e pessoais têm-se pensado em segurança. Segundo Nakamura e Geus (2007), a tecnologia da informação é um instrumento cada vez mais utilizado pelo homem, que busca incessantemente realizar seus trabalhos de modo mais fácil, mais rápido, mais eficiente e mais competitivo, produzindo assim os melhores resultados, tornando dessa maneira a confiabilidade, integridade e disponibilidade dessas estruturas (rede) essenciais para o bom andamento da organização.

Dentre as ameaças à rede de computadores, encontraremos a invasão, comumente realizada por um “hacker” com finalidades nocivas à saúde das informações que ali trafegam ou são armazenadas. Segundo Nakamura e Geus (2007), para monitorar o tráfego e identificar invasões, temos os “*Sistemas de Detecção de Intrusão*”, ou simplesmente IDS, quando configurados para monitorar servidores únicos ou hosts, são chamados de HIDS (“*Host Intrusion Detection System*”). Quando esses IDS’s são configurados para monitorarem todo o tráfego à rede, são chamados de NIDS (“*Network Intrusion Detection System*”), e será baseado, neste último, o protótipo proposto neste trabalho.

Nas últimas décadas têm-se estudado intensivamente a estruturação e funcionamento da inteligência humana, encontramos o denominado estudo das REDES NEURAIIS ARTIFICIAIS (também conhecidas como RNA's). De acordo com Russel e Norving, as redes neurais artificiais estudam as conexões neuronais e seu comportamento em redes, visando simular o aprendizado em computadores.

Não é meu objetivo surpreendê-los ou chocá-los — mas o modo mais simples de resumir tudo isso é dizer que agora existem no mundo máquinas que pensam, aprendem, criam. Além disso, sua capacidade de realizar essas atividades está crescendo rapidamente até o ponto — em um futuro visível — no qual a variedade de problemas com que elas possam lidar será correspondente à variedade de problemas com os quais lida a mente humana. (SIMON, 1957 apud RUSSEL, 2004, p. 22)

Considerando os temas abordados acima, imagina-se um sistema de detecção de intrusão em redes que aprenda utilizando os conceitos de redes neurais artificiais. A este propósito, é possível “utilizar suas características de reconhecimento de padrões e generalização para realizar a classificação dos eventos ocorridos em redes de computadores, classificando-os em normais ou intrusivos [...]” (LIMA, 2005, p. 2).

No cenário atual, de acordo com Nakamura e Geus (2007), os principais NIDS são *softwares* baseados em assinatura, ou seja, analisam o tráfego capturado e, em tempo real, comparam com os padrões presentes no banco de dados e regras, possibilitando, assim, que um invasor altere minimamente a forma de ataque e o NIDS não o identifique. Como solução para esta problemática, propõe-se neste trabalho um protótipo de NIDS que, após análise da assinatura, analisará padrões aproximados ou generalizados, através um módulo baseado em Redes Neurais, capazes de encontrar novos padrões com certo grau de similaridade em invasões, excluindo assim a possibilidade de invasões pouco diferentes do padrão comum passarem pelo NIDS.

## 1.1 PROBLEMÁTICA

No ano de 2014 o CERT<sup>1</sup> registrou 1.047.031 incidentes de segurança da informação, reportados no Brasil, maior valor registrado no país até os dias atuais, segundo pesquisa realizado pelo FBI/CSI em 2002, as perdas resultantes de ataques internos com intuito de roubar informações proprietárias totalizaram 170.827 mil dólares.

Estes são alguns dados que nos dão uma ideia a respeito da incidência e potencial de perda sobre a invasão de redes e computadores. Tendo em vista o crescimento da utilização das redes de computadores para tráfego de informações muito sensíveis (tais como senhas de bancos, e-mail, arquivos confidenciais, contratos, dentre outras do gênero), torna-se imprescindível a melhoria constante da segurança dessas redes de informação, utilizando-se de tecnologias mais eficazes e novos estudos na área.

## 1.2 OBJETIVOS

A seguir, serão apresentados os objetivos deste trabalho.

---

<sup>1</sup> CERT. Centro de Estudos, Resposta e Tratamento de incidentes em segurança da informação no Brasil. Disponível em: <<http://www.cert.br/stats/incidentes/>>. Acessado em: 01 setembro 2014.

### 1.2.1 Objetivo Geral

Este trabalho tem por objetivo o desenvolvimento de um sistema de detecção de intrusão por meio da utilização de Redes Neurais Artificiais.

### 1.2.2 Objetivos Específicos

Entre os objetivos específicos podem-se citar:

- desenvolver um algoritmo do tipo *Perceptron Multicamadas*<sup>2</sup> (também conhecido como *backpropagation*<sup>3</sup> ou em português, “retropropagação”) adaptado para análise de pacotes de rede;
- escolher os dados a serem considerados pela RNA;
- treinar a Rede Neural para detecção de intrusão, incluindo ajustes de nível de aproximação;
- testar o software desenvolvido;
- coletar os resultados obtidos e analisar os resultados.

## 1.3 JUSTIFICATIVA

Segundo Trilling (2005), é crescente o número de ataques cibernéticos mais sofisticados, isso é observado na evolução do perfil do cibercriminoso, refletindo no seu método de ataque, também evoluído.

Considerando o modelo de NIDS baseado unicamente em assinatura insuficiente para prover um nível consideravelmente alto de segurança para os dados, propõe-se com este trabalho a inclusão de um “analisador inteligente” dos pacotes que chegam na rede, utilizando

<sup>2</sup> Segundo RUSSEL (2007) Perceptron Multicamadas é um algoritmo de redes neurais com múltiplas camadas, amplamente difundido e robusto para aplicações com alto volume de tráfego de dados.

<sup>3</sup> Termo na língua inglesa para o Perceptron Multicamadas. Amplamente utilizado em literaturas técnicas.

RNA (Redes Neurais Artificiais), com intuito de ensinar à máquina detectar um nível mínimo de similaridade entre os padrões já aprendidos e os novos.

RNA's são capazes de criar, através de algoritmos matemáticos que simulam o neurônio humano, conexões similares à sinapse e ao reforço possibilitando à máquina aprender a identificar um padrão (RUSSEL, 2003, p. 13, tradução nossa).

Este sistema proposto, quando implementado, busca aumentar consideravelmente o nível da segurança de uma rede, uma vez que forneceria o fator “inteligente” à detecção dos padrões de ataques e invasões após o período de treinamento da RNA.

#### 1.4 ESTRUTURA MONOGRÁFICA

No primeiro capítulo, consta a introdução do trabalho com uma apresentação do tema, os objetivos gerais e específicos, a problemática e a justificativa.

No segundo capítulo, encontra-se a revisão bibliográfica, contendo todo o embasamento teórico necessário para o estudo e acompanhamento do presente trabalho, sendo estudados conceitos de segurança da informação, redes de computadores, intrusão em redes e redes neurais artificiais e principais técnicas e algoritmos disponíveis no mercado para implementação de um NIDS com apoio de uma RNA.

No terceiro capítulo, é apresentada a metodologia empregada neste projeto, desde as considerações sobre a literatura até a forma de pesquisa.

No quarto capítulo, é apresentado o desenvolvimento do protótipo, principais métodos, fluxogramas e de como foi elaborado. É mostrado o funcionamento do sistema em todos os aspectos e, ao final, é validado.

No quinto capítulo, conclui-se o presente trabalho com as conclusões do autor sobre os resultados obtidos e apresentados, no fim do capítulo quatro, após é apresentada uma breve descrição sobre possíveis trabalhos futuros.



## 2 REVISÃO BIBLIOGRÁFICA

Este capítulo visa a dar o embasamento teórico necessário sobre Sistemas de Detecção de Intrusão e Redes Neurais Artificiais, seu histórico, estrutura, características e desenvolvimento.

### 2.1 INTELIGÊNCIA ARTIFICIAL

A Inteligência Artificial (IA) é um ramo da ciência da computação ao mesmo tempo recente (sua criação data oficialmente de 1956) e muito antiga, pois foi construída a partir de ideias filosóficas e científicas herdadas de outras ciências, como por exemplo a lógica. O Objetivo central da IA é simultaneamente teórico – a criação de teorias e modelos para a capacidade cognitiva – e prático – a implementação de sistemas computacionais baseados nestes modelos. (BITTENCOURT, 2006, p. 20).

Segundo Russel (2004), a IA “é um campo universal, pois sistematiza e automatiza tarefas intelectuais. A IA tenta não apenas compreender, mas também construir entidades inteligentes.”

Para Feigenbaum (1992), IA é a parte da ciência da computação voltada para o desenvolvimento de sistemas de computadores inteligentes, isto é, sistemas que exibem características, às quais associam-se com a inteligência no comportamento humano – por exemplo: compreensão da linguagem, aprendizado, raciocínio, resolução de problemas [...] (FERNANDES, 2003, p. 2).

#### 2.1.1 Agindo de forma humana: o teste de Turing

O teste de Turing, proposto por Alan Turing (1950), foi projetado para fornecer uma definição operacional satisfatória de inteligência. A este propósito, Russel (2004, p. 4) escreve: “Em vez de propor uma lista longa e, talvez, controversa de qualificações exigidas para inteligência, ele sugeriu um teste baseado na impossibilidade de distinguir entre entidades inegavelmente inteligentes – os seres humanos[..]”.

Exemplificando, o teste de Turing consiste em uma pessoa conversar através de um *chat*, sem saber com quem ou o que está conversando, e tentar descobrir, através somente do diálogo digital se está conversando com um ser humano ou com um programa inteligente. Caso o programa inteligente consiga “enganar” a pessoa com quem está conversando, fazendo-a pensar que está conversando com outra pessoa, será considerado, inteligente.<sup>4</sup>

Desta forma, passou-se a considerar um sistema inteligente, todo aquele capaz de obter sucesso no teste de Turing.

### 2.1.2 Subdivisões e áreas específicas

A IA é uma ciência ampla e bastante abrangente, pois relaciona-se diretamente à demais áreas como, por exemplo, resolução de problemas de buscas ou, ainda, Mineração de dados, entre outros. Para Ganascia (1993 apud FERNANDES, 2003, p. 3), os principais modelos de inteligência artificial são:

- **Algoritmos Genéticos:** Modelo para aprendizagem de máquina, inspirado no livro *Origem das Espécies* escrito por Charles Darwin (1809-1882), criador da teoria evolucionista, segundo o qual somente os mais aptos sobrevivem. Os algoritmos genéticos criados por Holland (1975) visavam a emular operadores genéticos (específicos, como cruzamentos, mutação e reprodução) da mesma maneira que são observados na natureza. Isto é, criando-se dentro da máquina uma população representada por cromossomos e efetuando o processo de simulação da evolução, seleção e reprodução, gerando assim uma nova população.
- **Programação Evolutiva:** Campo da IA concebido por Fogel (1960), assemelha-se aos algoritmos genéticos, porém focando-se mais especificadamente na relação comportamental entre parentes e seus descendentes.
- **Lógica *Fuzzy*:** Também denominada lógica difusa ou nebulosa, foi estruturada por Lofti Zadeh em 1965. Criada para representar, manipular e

---

<sup>4</sup> DEPARTAMENTO DE INFORMÁTICA DA UNIVERSIDADE ESTADUAL DE MARINGÁ. **Teste de Turing**. Disponível em: <<http://www.din.uem.br/ia/maquinas/turing.htm>>. Acessado em: 14. out. 2014.

modelar informações incertas (princípio da incerteza). Este modelo tem por objetivo “permitir graduações na pertinência de um elemento a uma dada classe, ou seja, possibilitar a um elemento de ele pertencer com maior ou menor intensidade àquela classe.” (BITTENCOURT, 2006, p. 290).

- **Sistemas Baseados em Regras:** São sistemas que implementam comportamentos inteligentes de especialistas humanos. Este modelo também chamado de Sistema Especialista, tem por base, a utilização de expressões ou regras, e, através destas a criação de novos modelos de regras e resoluções para encontrar determinadas características que se adaptem à tais regras, um Sistema Especialista possui em sua arquitetura, de acordo com Bittencourt (2006, p. 261), “uma *base de regras*, uma *memória de trabalho* e um *motor de inferência*[..]”.
- **Raciocínio Baseado em Casos:** É o campo de estudo da IA que utiliza uma grande biblioteca de casos para consulta e resolução de problemas. Os problemas atuais são resolvidos através da recuperação e consulta de casos já solucionados e da consequente adaptação das soluções encontradas.
- **Redes Neurais:** Também conhecida como como Redes Neurais, Modelo Conexionista, Neurocomputação, Modelo de Processamento Paralelo Distribuído, Sistemas neuromórficos e Computadores Biológicos. São considerados uma classe de modelagem de prognóstico que trabalha por ajuste repetido de parâmetro. Estruturalmente, uma RNA (Rede Neural Artificial) consiste em um número de elementos interconectados (chamados Neurônios) organizados em camadas que aprendem pela modificação da conexão firmemente conectando as camadas.

Como este é o modelo no qual o presente trabalho foi embasado, será amplamente abordado na próxima seção.

## 2.2 REDES NEURAIS ARTIFICIAIS

Redes Neurais Artificiais ou RNA's são modelos matemáticos inspirados no funcionamento do cérebro humano, mais especificadamente na estrutural de neurônios e sua

forma de comunicação (sinapse), criando uma estrutura capaz de aprender. Para Kohonen (1972 apud FERNANDES, 2003, p. 57), As RNA's são definidas como redes massivamente paralelas e interconectadas de elementos simples, com organização hierárquica. Estes elementos devem interagir com objetivos do mundo real, da mesma maneira que o sistema nervoso biológico.

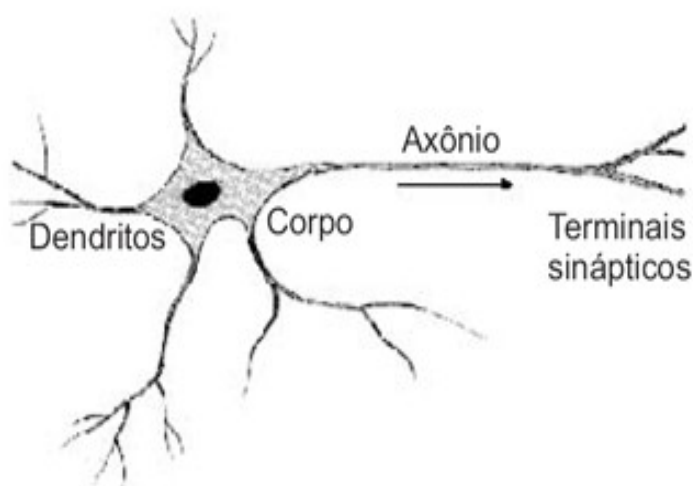
Segundo Loesch (1996), RNA's são sistemas computacionais de implementação em hardware e software, que imitam as habilidades computacionais do sistema nervoso biológico, utilizando para isso um grande número de neurônios artificiais interconectados. (FERNANDES, 2003, p. 57).

“As RNA's são capazes de resolver, basicamente, problemas de aproximação, predição, classificação, categorização e otimização”. (REZENDE, 2003, p. 145).

### 2.2.1 Estrutura

Para Russel (2004, p.713), “Um neurônio é uma célula cuja principal função é coletar, processar e disseminar sinais elétricos”. Abaixo, pode-se ver um modelo de neurônio biológico:

Figura 1 - Fluxo da informação em um neurônio biológico.



Fonte: Kovács (1997, p. 22)

A Figura 1 demonstra um modelo de neurônio biológico e o fluxo da informação no mesmo. De acordo com Kovács (1997) o neurônio é uma célula capaz de transmitir

informações através de um processo conhecido como *sinapse*. O neurônio é formado por Dendritos, Corpo ou Soma e Axônio, conforme descrito:

- **Axônio:** “Neurônio, em geral, possuem um único axônio que se caracteriza como uma porção filamentar mais alongada, cuja principal função é transmitir informação na forma de impulso elétrico.” (Kóvac, 1997, p.19).
- **Dendritos:** Normalmente chamados de árvore dendrital (pelo grande número de dendritos), são os receptores dos impulsos elétricos enviados pelos axônios no processo de sinapse. A este propósito, Kovács (1997) escreveu: “Sua característica mais importante é oferecer uma ampla área de contato para a recepção e informação”.
- **Corpo ou Soma:** Parte central que contém o núcleo e o pericário da célula, concentrando seus componentes vitais para funcionamento do metabolismo celular.

O contato e transmissão de informação entre neurônios acontece em estruturas conhecidas como sinapses, conforme Kovács (1997, p.21) escreveu:

Na sinapse, distinguem-se uma parte anterior composta pela membrana de um axônio pelo qual chega o pulso nervoso, chamada membrana pré-sináptica e uma parte posterior chamada membrana pós-sináptica, que receberá a informação. Essas duas partes [...] são separadas pela fenda sináptica.

No Neurônio biológico, o processo de transmissão de informação (sinapse), segundo Kovács (1997), se dá quando chega um pulso nervoso pelo axônio pré-sináptico, fazendo com que sua membrana libere neurotransmissores, absorvidos pela membrana do dendrito (pós-sináptica), provocando assim uma alteração na sua composição química e consequentemente elétrica. De acordo com o neurotransmissor liberado pela membrana pré-sináptica, o impulso poderá ser estimulante ou inibidor, fazendo com que sejam classificadas em sinapses inibitórias e sinapses excitatórias. O impulso elétrico no axônio e no dendrito ocorre, segundo Bittencourt (2006), através da concentração de íons  $K^+$  e  $Na^+$  em ambos os lados da membrana. Conforme descrito por Kovács (1997, p. 23):

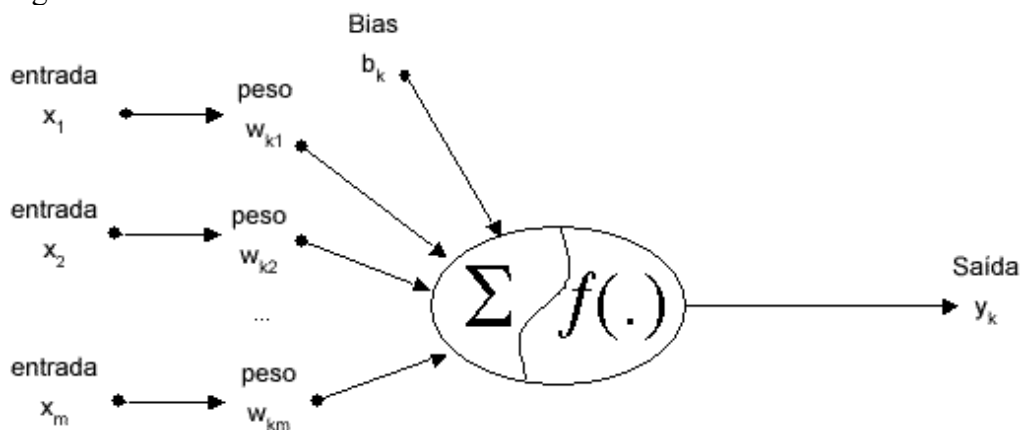
A formação de um potencial de ação em um ponto arbitrário ao longo da membrana axonal ocorre quando, por qualquer razão, esta sofre uma depolarização suficiente para que o potencial  $E$  cruze um valor conhecido como limiar de disparo. Neste ponto devido a um comportamento acentuadamente não linear e diferenciado da permeabilidade da membrana aos íons de sódio e potássio, provoca-se um influxo súbito de sódio, o que leva o potencial de membrana a um valor  $E > 0$ . Em seguida, o fluxo de sódio diminui e o fluxo de potássio se restabelece, reconduzindo a membrana a um potencial de repouso.

### 2.2.2 Neurônio de McCulloch e Pitts

Em 1943 o psiquiatra e neuroanatomista Warren S. McCulloch em conjunto com o matemático Walter Pitts propuseram em um artigo um modelo matemático de neurônio artificial, baseado no neurônio biológico e seu funcionamento, principalmente na comunicação e construção de redes neurais. Este modelo, porém limitava-se pela sua natureza binária. (BITTENCOURT, 2006, p. 310).

Descrevendo de maneira resumida e intuitiva o modelo criado, se a soma ponderada das entradas de um neurônio ultrapassar um determinado limite de disparo, então a saída será de valor 1, senão será 0. Assim como as saídas, as entradas também deverão ser binárias. Bose (1996, p. 23) descreve o processo de funcionamento do neurônio artificial na figura 2.

Figura 2 - Estrutura do Neurônio Artificial.



Fonte: BOSE (1996, p. 23)

Pode-se observar a entrada dos valores no vetor  $X$  ( $X_1, X_2, X_3 \dots$ ), posteriormente a multiplicação de cada valor pelo seu peso, representado por  $W_n$  e, por fim, a soma ponderada dos valores do vetor. Os valores que aplicados à função de ativação ultrapassarem o limiar de disparo, ou valor limite, segundo o modelo McCulloch e Pitts, resultarão em 1 (equivalente à sinapse de excitação), caso contrário, a saída terá valor 0 (equivalente à sinapse de inibição).

### 2.2.3 Estudo dos Algoritmos

Através dos anos os algoritmos que representavam RNA's foram evoluindo, novos modelos propostos e vertentes dentro desta promissora área foram sendo estudadas. Dentro do estudo das redes neurais artificiais, encontramos dois modos básicos para o treinamento e aprendizado, descritos abaixo, conforme Fernandes (2003):

- **Treinamento Supervisionado:** Neste modelo, durante a fase de treinamento (ou ensino) da rede neural, informa-se para a rede um vetor de entrada e associa-se esta entrada ao valor da saída desejada. O algoritmo assim ajusta seus pesos para que a entrada inserida obtenha a saída desejada. Desta maneira na fase de classificação, obtêm-se uma matriz de pesos equilibrada, de acordo com o que lhe foi ensinado.
- **Treinamento Não Supervisionado:** Neste modelo, os neurônios formam sua própria classificação de dados, para isso, os dados precisam partilhar de características comuns, a rede identificará tais características e irá se organizar. Uma forma de auto-organização é o aprendizado competitivo, onde os neurônios “competem” para serem ativados.

Pode-se visualizar no quadro 1, um comparativo para os principais modelos de RNA com suas características:

Quadro 1: Modelos de redes neurais

Perceptrons	
Aplicações	Reconhecimento de Caracteres
Vantagem	RNA mais antiga
Desvantagem	Não reconhecimento de padrões complexos, sensível à mudanças
Backpropagation	
Aplicações	Larga aplicação
Vantagem	Rede mais utilizada, simples e eficiente
Desvantagem	Treinamento supervisionado exige muitos exemplos
Counterpropagation	
Aplicações	Reconhecimento de padrões, análise estatística
Vantagem	Rapidez do treinamento
Desvantagem	Topologia complexa
Hopfield	
Aplicações	Recuperação de dados e fragmentos de imagem
Vantagem	Implementação em larga escala
Desvantagem	Sem aprendizado, pesos preestabelecidos
Bidirectional Associative Memories (BAM)	
Aplicações	Reconhecimento de padrões
Vantagem	Estável
Desvantagem	Pouco eficiente
Kohonen	
Aplicações	Reconhecimento de padrões
Vantagem	Estável
Desvantagem	Pouco eficiente

Fonte: BITTENCOURT (2006, p. 315)

Levando-se em conta tais informações e grande análise, optou-se pela utilização do modelo *Backpropagation* neste trabalho, levando em consideração principalmente sua eficiência ao lidar com grande volume de dados, capacidade de generalização e versatilidade.

## 2.2.4 Perceptron Multicamadas ou Backpropagation

O MLP (*Multilayer Perceptron* ou, ainda, Perceptron Multicamadas em tradução livre) é uma evolução do modelo Perceptron, com apenas uma camada, que se restringe a resolver problemas linearmente separáveis.

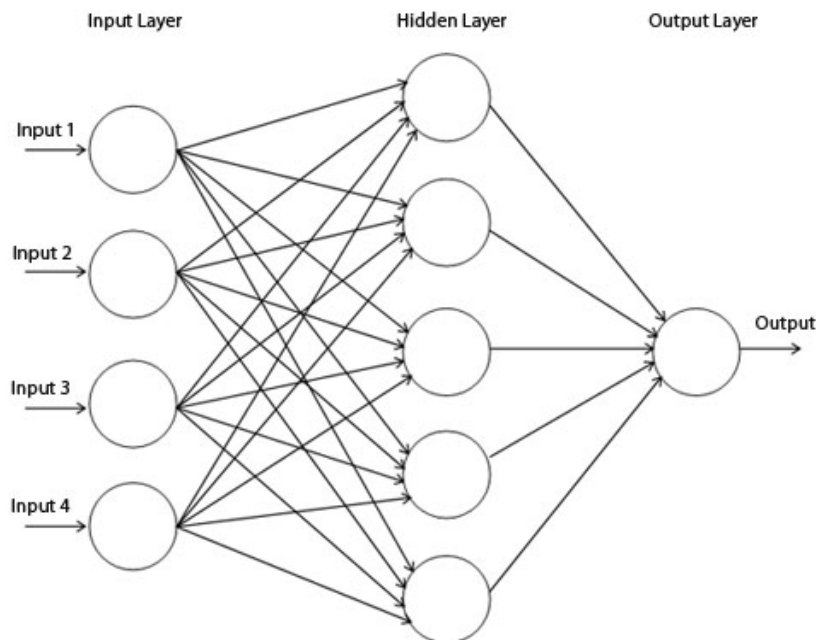


Minsky e Paper analisaram matematicamente o Perceptron e demonstraram que redes de uma camada não são capazes solucionar problemas que não sejam linearmente separáveis. Como não acreditavam na possibilidade de se construir um método de treinamento para redes com mais de uma camada, eles concluíram que as RNA's seriam sempre suscetíveis a essa limitação. (FAUSET, 1995 apud FERNANDES, 2003, p. 76).

Contudo, o estudo de um algoritmo com mais de uma camada, com treinamento supervisionado e de retropropagação mostrou que é possível treinar eficientemente estas redes e solucionar uma gama muito maior de problemas (incluindo os até então irresolúveis, linearmente separáveis). Criado por Rumelhart, Hilton e Williams em 1986, o MLP é amplamente utilizado nos dias de hoje com boa performance para grande volume de dados. (FERNANDES, 2003, p. 76).

Nessas redes, cada camada tem uma função específica. A camada de saída recebe os estímulos da camada intermediária e constrói o padrão que será a resposta. As camadas intermediárias funcionam como extratoras de características, seus pesos são uma codificação de características apresentadas nos padrões de entrada e permitem que a rede crie sua própria representação, mais rica e complexa, do problema. (FERNANDES, 2003, p. 76).

Figura 3 - Estrutura da Rede MLP



Fonte: RUSSEL (2004, p. 721)

Pode-se observar, na Figura 3, que o modelo MLP é mais complexo, apresentando a camada intermediária (por vezes chamada de camada oculta). Para Fernandes (2003), o treinamento deste modelo ocorre em dois passos. No primeiro, é inserido o vetor de entradas, este flui através da rede (conforme o modelo da figura 2) até que a resposta seja produzida

pela camada de saída. No segundo passo, a saída obtida no primeiro passo é comparada à saída correta (que deve ser inserida na rede de acordo com o modelo de ensino supervisionado, em que apresenta-se, na fase de treinamento, os valores de entrada e sua saída correta para, posteriormente, a rede reproduzir), se houver diferença entre a saída obtida e a correta, o erro é calculado. Então, inicia-se a etapa de retropropagação, da saída à entrada, os pesos são reajustados para eliminarem o erro calculado anteriormente.

### 2.2.5 *Momentum*

Segundo Severo (2010), O termo *momentum* tem por objetivo aumentar a velocidade de treinamento da RNA e reduzir o perigo de instabilidade. Este termo pode ou não ser utilizado durante o treinamento e seu valor varia de 0.0 (não utilização) a 1.0. A adição do termo *momentum* no processo de ensino da RNA reduz as oscilações em regiões próximas a mínimos locais na função de aprendizado.

### 2.2.6 *Função de Ativação*

Para Haykin (2001), a função de ativação é muito importante na estrutura da RNA pois ela é quem vai definir a gênero do impulso, se é um impulso excitatório, quando a conexão recebe um valor alto (normalmente 1), ou inibitório, quando a conexão não é estimulada e recebe um valor baixo (normalmente 0 ou -1). Dentre os diversos tipos de função de ativação, para o presente trabalho foi utilizada a função de ativação *Sigmoide tangente hiperbólica*:

A função *tangente hiperbólica* possui características similares a função logística, porém seus valores contínuos variam de -1 a 1, sendo comumente aplicada na construção de redes neurais artificiais [...] O fato de se permitir que uma função de ativação do tipo sigmoide assuma valores negativos traz benefícios analíticos e vantagens durante a fase de aprendizado. (Lima, 2005, p. 23).

## 2.3 SEGURANÇA DA INFORMAÇÃO

“Se você remover os termos tecnológicos e as interfaces gráficas com o usuário, o ciberespaço não é tão diferente do seu correspondente do mundo real, carne e osso, [...] e não bits.” (SCHNEIER, 2001, p. 27).

Baseado na afirmação acima, pensa-se na particularidade dos crimes cibernéticos, de acordo com Schneier (2001, p. 27), os ataques cibernéticos costumam imitar crimes ou ataques “no mundo real”. Partindo da máxima, na área de segurança da informação, que nenhum sistema é totalmente seguro, procura-se criar barreiras a fim de melhorar ao máximo a segurança de determinadas redes.

Existem duas formas de pensar segurança em rede. Na primeira, protegemos a rede de invasões e pessoas mal-intencionadas que estejam querendo invadir ou danificar seus dados. Na segunda, protegemos a rede de seus próprios usuários – e das intempéries do destino. (CARDOSO; GUTIERREZ, 2000, p. 180).

Segurança é um assunto abrangente e, normalmente, fala-se de maneira abstrata sobre o assunto. Tal sistema é seguro ou não há como roubarem meus dados, são colocações que falam sobre determinado sistema ou rede, porém não toma em consideração o contexto onde está incluso. Um sistema operacional pode estar seguro contra um ataque externo, mas se o invasor obtiver acesso físico ao servidor? Não se pode considerá-lo inútil por não proteger de uma ameaça interna. (SCHNEIER, 2001, p.24).

A segurança da informação visa a proporcionar, para Kurse e Ross (2001):

- **Integridade:** Garantia de que os dados não serão adulterados, corrompidos ou sofram alguma modificação indevida desde sua criação.
- **Confidencialidade:** É a garantia de que os dados não serão acessados por pessoas não autorizadas ou, ainda, que contas particulares sejam inacessíveis à quaisquer outros que não sejam os portadores da conta sem sua devida autorização. Este é um dos tópicos mais abrangentes, que envolve também a autenticidade na utilização de um cartão de crédito não fraudulento, por exemplo. A luta hoje pela privacidade, em especial, privacidade do próprio governo, cabe a este tópico garanti-la, ou buscá-la.
- **Disponibilidade:** Esta é a garantia de que os dados ou serviços, estejam operando ou disponíveis sempre que garantidos ou programados para estarem, protegendo contra a interrupção de determinado serviço causando indisponibilidade de acesso à informação quando se devia tê-la.

Compreende-se, então, segurança como uma prática que visa proporcionar ao máximo as características descritas acima, com intuito de proteger a informação de possíveis ameaças.

### 2.3.1 Tipos de ameaças mais comuns

Estar *online* requer sempre muito cuidado, a impessoalidade e a natureza das operações digitais propicia um ambiente fácil de se atacar, se não estiver bem protegido. Seguem abaixo, alguns dos gêneros mais comuns de ameaças digitais:

- **Negação de Serviço ou DDOS:** Termo genérico para as mais variadas formas de ataques que visam parar ou congelar serviços e ou tráfego de dados [atacando a disponibilidade dos dados]. Por exemplo, atacar o sítio eletrônico de uma loja virtual para que os servidores travem e os clientes não consigam acessá-los, logo o prejuízo será grande, pois, sem a disponibilidade param-se as vendas. O(s) atacante(s) enviam, em um dos tipos de ataque, solicitações do tipo SYN, onde o servidor deve responder e, após, aguardar confirmação da máquina que solicitou a conexão, imagine um servidor que recebe 50 destas conexões por segundo, e tenha *timeout*<sup>5</sup> de 75 segundos, provavelmente o grande volume de requisições fará o servidor travar. (SCHNEIER, 2001, p. 185).
- **Exploit e injeções de código:** Termo apropriado do inglês que caracteriza um ataque a determinado software ou tecnologia visando à exploração de uma falha [vulnerabilidade], podendo causar no sistema alvo a indisponibilidade do serviço, leitura e acesso a dados ou ainda uma invasão completa ao sistema via *backdoor*<sup>6</sup>. Este *software* pode explorar principalmente a falta de filtragem ou sanitização dos dados que podem ser inseridos pelo usuário do sistema. Incluem-se, nesta categoria, as injeções de código [*sqlinjection*]<sup>7</sup>, por exemplo] para fim de otimização da leitura, uma vez que estas inserções indevidas possuem a mesma natureza de explorar uma falha

<sup>5</sup> Termo utilizado para especificar o tempo em que o servidor esperará a resposta com a conexão ativa.

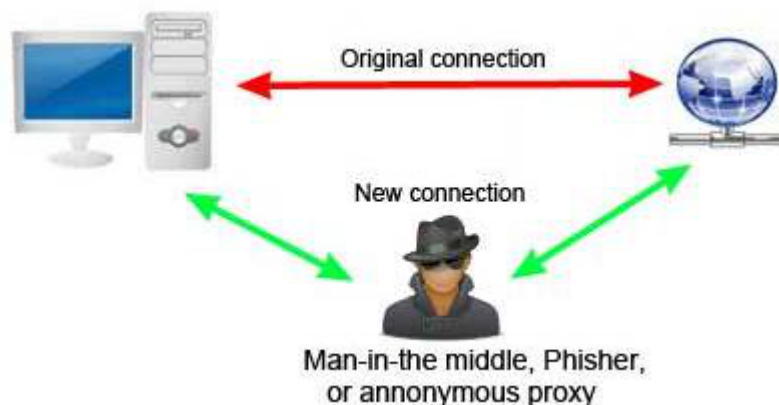
<sup>6</sup> *Backdoor* ou *back Orifice* [termos apropriados do inglês] são *softwares* de administração remota, utilizados por hackers para controlar e acessar um computador remotamente [invasão]. (SCHNEIER, 2001, p. 160).

<sup>7</sup> Nome dado à inserção de código SQL em uma variável disponível ao usuário, que será inserida diretamente no banco de dados e permitirá ao atacante extrair informações do mesmo.

não prevista na programação e desenvolvimento do sistema alvo. (SCHNEIER, 2001, p. 166).

- **Roubo de sessão e *Phishing*:** São técnicas que o atacante utiliza para roubar credenciais e fazer-se passar pela vítima e, ou enganar com páginas de acesso falsas, para captura das informações e credenciais da vítima. Por exemplo, a página de login idêntica à do seu banco, com o link enviado para o e-mail da vítima e esta acessa sem perceber que está entregando a um criminoso os seus dados de login e senha. (ALMEIDA, 2011, p. 2).
- **Interceptação de dados:** Este modelo de ataque também conhecido como MITM (*Men in the Middle*, ou homem no meio, em tradução nossa) no que o atacante faz-se passar por servidor (para o cliente) e passa-se por cliente (para o servidor). Ele age como um agente interceptando, copiando e reenviando as requisições, tornando a percepção do ataque mais difícil. (HWANG, 2008, p. 166). Conforme apresentado na figura 4.

Figura 4 - Arquitetura de um ataque MITM



Fonte: HWANG (2008, p. 166).

- **Engenharia Social:** Esta prática é comumente a mais propícia a estar vulnerável, apesar de estar fora do escopo deste trabalho, vale ressaltar que a arte de enganar pessoas é fortemente explorada por criminosos, uma vez que tiram proveito da inocência, falta de informação ou ingenuidade de outrem. Pode-se com a técnica explorar desde acesso físico à rede alvo, até a obtenção de credenciais para login em um sistema, passando-se (por exemplo, no telefone) por outra pessoa, uma vez que se possua algum dado da mesma. (CARDOSO, GUTIERREZ, 2000, p. 182).

### 2.3.2 Meios comuns de proteção

Existem diversas formas e níveis diferentes para se tratar segurança da informação, desde backups aos atuais RAID para cópia de dados, de *antivírus* à *anti-malwares* em tempo real para o escaneamento do computador em busca de *softwares* maliciosos. Como o foco deste trabalho é a proteção da rede e seu tráfego de dados, se dará enfoque aos principais meios de proteção para os dados trafegados na rede em tempo real. De acordo com Schneier (2001) são eles:

- **Firewall:** “No mundo digital, um *firewall* é um mecanismo que protege a rede interna de uma organização contra hackers maliciosos, [...] ele mantém os intrusos do lado de fora.” (SCHNEIER, 2001, p. 191). Estes módulos de proteção sofreram algumas variações ao longo dos tempos, com o intuito de separar redes (assim como as barreiras físicas), hoje funcionam basicamente como uma grande muralha que separa a internet de uma rede local, deixando acesso somente a portas e programas específicos e, principalmente, dentro de uma DMZ<sup>8</sup>. Permitindo que determinados pacotes passem pelo *firewall* e outros não, o administrador da rede consegue controlar quais as portas de entrada de agentes externos à sua rede interna. Porém, devido a essas características, o *firewall* é efetivo contra ataques a portas que não precisam estar abertas, mas não podem oferecer proteção contra pacotes legítimos que contenham, por exemplo em seu meio algum código malicioso disfarçado. (KUROSE; ROSS, 2010, p. 535).
- **Sistemas de Detecção de Intrusão:** “Para detectar muitos tipos de ataque, precisamos executar uma **inspeção profunda de pacote** [...]” (KUROSE; ROSS, 2010, p. 540). O sistema de detecção de intrusão tem essa tarefa, trabalhando em conjunto com um *firewall*, ele analisa os pacotes autorizados pelo mesmo, fazendo uma inspeção profunda em todo o pacote, cabeçalho e conteúdo, a fim de identificar a assinatura ou um padrão considerado pelo programa como malicioso. (KUROSE; ROSS, 2010)

Como o presente trabalho tem o foco em sistemas de detecção de intrusão, o tópico será melhor apresentado e estudado a seguir.

---

<sup>8</sup> DMZ ou *Demilitarized Zone*, segundo Schneier (2001), é uma área especificada pelo *firewall* para conter serviços públicos, uma área para o acesso proveniente da internet.

### 2.3.3 Sistemas de Detecção de intrusão (IDS)

Sistemas de detecção de Intrusão ou IDS (do inglês *Intrusion Detection System*) são importantes ferramentas na possível detecção de anomalias, atividades suspeitas ou inapropriadas, além da detecção, é função do IDS o alerta destes eventos. De acordo com Silva (2004, tradução nossa), há dois tipos clássicos de IDS, de acordo com o alvo do monitoramento:

- **Baseados em *hosts*:** São chamados de HIDS, utilizam de arquivos de logs e mecanismos (portas e protocolos) para monitoria de um sistema único (servidor ou PC), mudanças de privilégios, execução de programas e etc. Seu foco é o usuário final.
- **Baseado em redes:** Também conhecidos como NIDS (*Network Intrusion Detection System*) são sistemas alimentados através da captura de pacotes trafegados na rede, em que cada pacote é analisado profundamente e comparado a uma base de dados, contendo a assinatura de tráfego malicioso, regras de proibição ou ainda outra forma de detecção híbrida e mais elaborada.

Considerando o método de detecção, Silva (2004, tradução nossa) também escreveu:

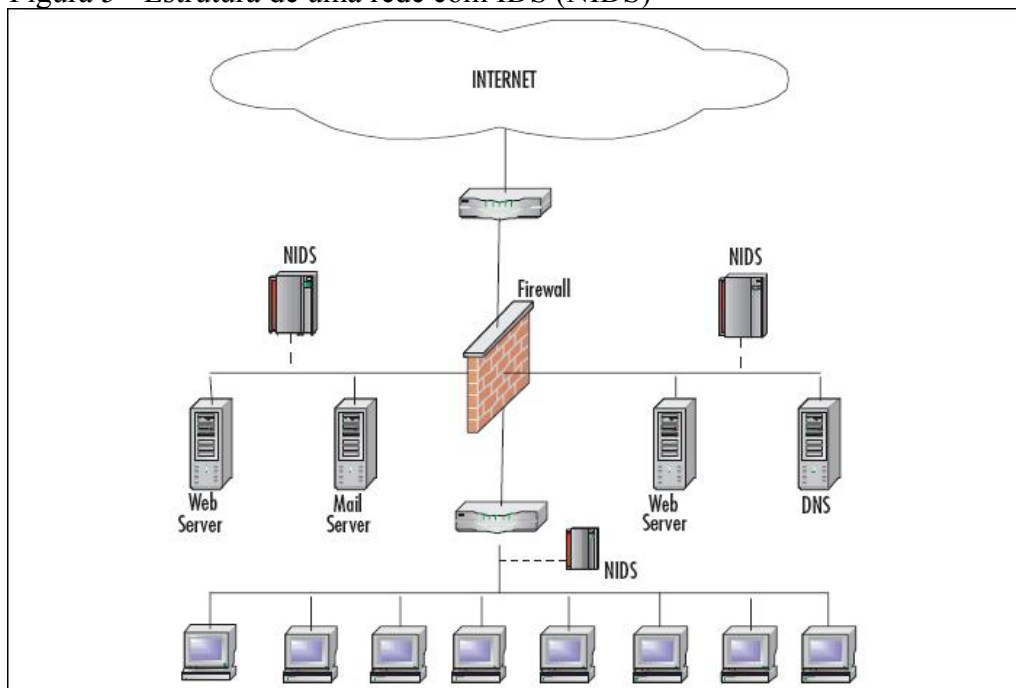
- **Baseado em Assinatura:** Método que compara as informações contidas nos pacotes ou seus padrões com uma base de dados, contendo as principais e documentadas formas de invasão.
- **Baseado em Comportamento:** Em que o IDS conhece a natureza de determinados eventos ou funcionalidades pré-definidas pelo administrador que podem se enquadrar como suspeitas.

Operacionalmente um IDS baseado em assinatura analisa cada pacote que passa, comparando cada pacote analisado com as assinaturas no banco de dados. Se um pacote (ou uma série deles) é compatível com uma assinatura no banco de dados, o IDS gera um alerta. O alerta pode ser enviado ao administrador da rede através de uma mensagem de e-mail, pode ser enviada ao sistema de administração da rede, ou pode simplesmente ser registrado para futuras inspeções. (KUROSE; ROSS, 2010, p. 543).

### 2.3.4 Sistemas de Detecção de intrusão em Redes (NIDS)

Pode-se visualizar o NIDS como “[...] detetives policiais autônomos que vagam pela cidade [leia-se rede]: Eles sabem quais são os comportamentos suspeitos [...] e mantêm um olho bem aberto para isso.” (SCHNEIER, 2001, p. 197). Schneier (loc. cit.) também esclarece sobre o polêmico e reincidente problema com “falso-positivos”<sup>9</sup>, o NIDS, seja ele *software* ou estrutura (quando envolve hardware também) precisa estar bem configurado e ajustado. Alguns *softwares* do mercado possuem melhor desempenho do que outros no tangente a detecção de ataques efetivos e seus alertas.

Figura 5 - Estrutura de uma rede com IDS (NIDS)



Fonte: (KUROSE; ROSS, 2010, p. 542)

Cada ataque envolve uma série de fatores, e cada um é particular, envolve um contexto inteiro. Conforme Schneier (op. cit., p. 29), os ataques cibernéticos possuem uma natureza mutável, uma vez que envolvem a escrita de códigos ou, comumente, podem ser alterados sem modificar seus fins. Esta situação faz com que os NIDS tornem-se pouco eficazes a ataques que sejam inteiramente novos ou, ainda não registrados em um banco de dados. É justamente neste ponto, ainda pouco explorado, que o presente trabalho visa estudar,

<sup>9</sup> Situação característica onde o IDS emite um alerta de invasão ou tentativa quando, na realidade esta não existe.



propondo uma estrutura auxiliar na análise de anomalias, ataques e invasões sobre redes de computadores, utilizando de RNA's para comparar a similaridade dos pacotes para com os grupos de pacotes considerados maliciosos, conseguindo, assim, identificar um ataque mesmo que este esteja com seu código com certa alteração.

## 2.4 SNORT

Considerado um dos melhores *softwares* para detecção de intrusão, este versátil programa pode atuar de maneira bastante eficiente, como NIDS OU HIDS, de acordo com como é configurado. “O SNORT é um código aberto IDS de domínio público com centenas de milhares de implementações existentes.” (Snort, 2007; Kozio, 2003 apud KUROSE; ROSS, 2010, p. 543).

O SNORT é uma ferramenta NIDS desenvolvido por Martin Roesch, *open-source*, bastante popular por sua flexibilidade nas configurações de regras e constante atualização frente as novas ferramentas de invasão. Outro ponto forte desta ferramenta é o fato de ter o maior cadastro de assinaturas, ser leve, pequeno, fazer escaneamento do micro e verificar anomalias dentro de toda a rede ao qual seu computador pertence. (SNORT, 2014).

### 2.4.1 Características

O SNORT é um *software* de código aberto, com ampla assistência da comunidade de desenvolvedores e usuários. Pode ser encontrado para as plataformas Linux, Windows e UNIX. Utiliza o módulo *libcap* para captura de pacotes, uma base de dados com as assinaturas e *rules* necessárias para a identificação das anomalias. (KUROSE; ROSS, 2010). No quadro 2, há um comparativo com pontos positivos e negativos do SNORT:

Quadro 2: Pontos Positivos e Negativos do SNORT

Pontos Positivos	
Extremamente flexível	Algoritmos de inspeção baseados em regras, sem falsos positivos inerentes, controle total do refinamento das regras.
Detecção Multidimensional	Anomalias no protocolo, assinatura (impressão digital) do ataque, anomalias no comportamento.
Imensa adoção	Dezenas de milhares de instalações (42 mil), algumas das maiores empresas do mundo.(Microsoft, Intel, PWC), milhares de contribuidores fazendo regras para novas vulnerabilidades.
Suporte da comunidade <i>open-source</i>	Rápida resposta à ameaças, velocidade de inovação, velocidade de refinamento.
Pontos Negativos	
Performance modesta	Menos de 30mbps, para redes de até 10Mbps.
Interface gráfica limitada	Configuração do sensor, gerenciamento de regras.
Sem suporte comercial	Implementação lenta e cansativa, capacidade analítica limitada, dependência de pessoas “capacitadas” e nem sempre estáveis, gastos significativos com recursos humanos.

Fonte: SNORT (2014)

A metodologia mais utilizada nos IDS atuais é baseada na ideia de que a análise ocorre através da comparação entre os eventos analisados e uma base de dados referente a ataques conhecidos. Este tipo de IDS funciona de forma semelhante a um antivírus tradicional: a base de dados contém assinaturas, que nada mais são do que indicativos de padrões existentes em cada tipo de ataque. A ideia é que qualquer ação que não se enquadrar nas regras definidas, são consideradas aceitáveis. Desta forma, para que um IDS baseado em assinatura seja eficiente, se faz necessária uma atualização constante de sua base de assinaturas. Tende a ser mais rápido que o método de Anomalia, além de não gerar tantos falsos positivos quando bem implementado, uma vez que consegue “entender” o tipo de ataque que está em andamento. Este mesmo fator é seu ponto fraco, visto que por ter que conhecer o ataque, não consegue detectar técnicas novas ou que não estejam devidamente atualizadas em sua base. (MACHADO; VOLTZ; DIAS, p. 3).

## 2.4.2 Estrutura funcional

O SNORT tem como princípio básico a captura de todos os pacotes trafegados no segmento da rede (previamente configurados) para, a partir deles, avaliar o que é ameaça ou não, conforme a documentação oficial disponível:

O Snort é um aplicativo que habilita a placa de rede do computador onde foi configurado, para modo promiscuo, ou seja, permite que todos os pacotes que trafegam pelo segmento de rede daquela máquina sejam capturados. Através de regras que são as assinaturas conhecidas dos ataques, é possível descobrir uma

variedade de ataques e sondagens, como *buffer overflow*, *port scans*, ataques *CGI*, verificação de *SMB*. O Snort com capacidade de alerta em tempo real, pode enviar os alertas a um arquivo de alerta individual ou a um meio externo como o *WinPopUp* (utilitário responsável por mandar mensagens de uma máquina para outra em uma rede). (SNORT, 2014)

Figura 6 - Estrutura de fluxo de um pacote a ser analisado

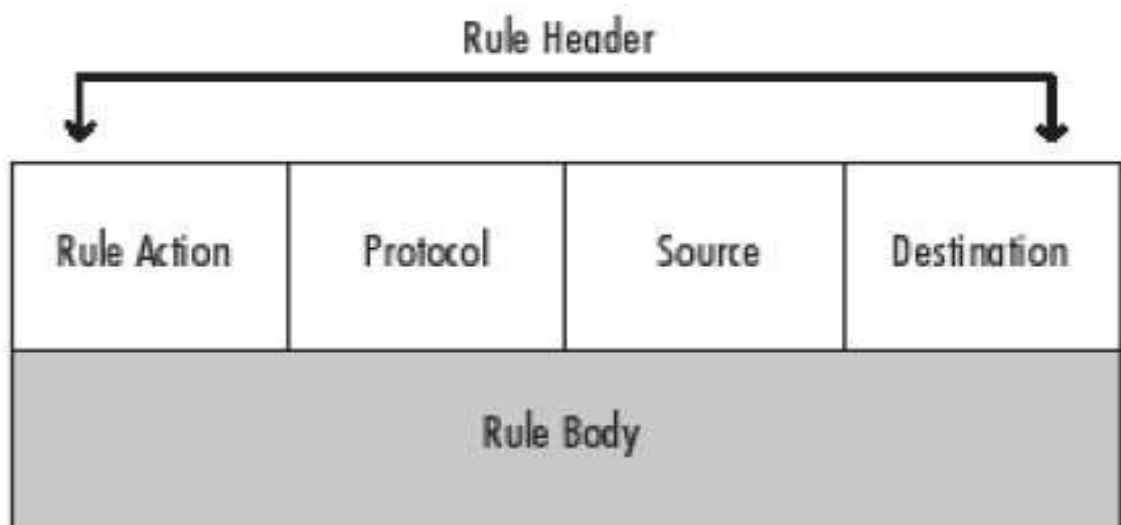


Fonte: (MACHADO; VOLTZ; DIAS, p. 4)

De acordo com a Figura 6, o pacote atravessa as estruturas do SNORT e, em tempo real, é checada, caso seja confirmado uma anomalia, invasão ou enquadre-se em alguma regra, o pacote segue seu fluxo normal e dispara o alerta. Caso o administrador da rede tenha configurado alguma regra diferenciada ou ação não padrão, o comportamento da ferramenta poderá variar, o que a torna tão versátil.

A Figura 7 apresenta um modelo estrutural de uma regra genérica do SNORT, seguida pela Figura 8 que mostra um exemplo de regra:

Figura 7 - Estrutura de uma Regra do SNORT (*rule*)



Fonte: (MACHADO; VOLTZ; DIAS, p. 4)

Figura 8 - Exemplos de regras do SNORT

Regra 1:

```
alert icmp any any -> any any (dsize:1048; content:"A"; msg: "Ping detectado";sid:1000001;)
```

Regra 2:

```
alert icmp any any -> any any (content:"A"; dsize:1048; msg: "Ping detectado";sid:1000001;)
```

Fonte: (MACHADO; VOLTZ; DIAS, p. 4)

As duas regras acima têm funções parecidas. Segundo Machado, Voltz e Dias (loc. cit.) elas criam alertas de qualquer origem e porta para qualquer origem e porta procurando por pacotes que tenham a letra "A" no *payload*<sup>10</sup> do pacote, que tenham tamanho de 1048 bytes. Caso algum pacote combine com essa regra, o SNORT gera a Mensagem "Ping detectado" com o id de identificação 1000001.

## 2.5 TRABALHOS CORRELATOS

Alguns trabalhos na área de IA e segurança da informação têm sido apresentados nos últimos anos. Nesta seção são apresentados e brevemente discutidos alguns destes projetos, bem como os diferenciais do presente trabalho para com seus correlatos.

Silva et al. (2004) descrevem um sistema de detecção de intrusão baseado em RNA utilizando a rede *Hamming* (uma RNA diferente da utilizada no presente projeto) e como fonte de dados para a rede utilizou-se regras do SNORT, ou seja, a RNA necessita do SNORT para a implementação total do IDS. Utilizando a técnica de *Extract Feature* (utilizada para extrair dados e padronizá-los para serem inseridos na RNA) o *payload* do pacote TCP é analisado e classificado.

Lima (2005) elaborou um modelo de IDS utilizando RNA para classificar os pacotes gerados pela rede. Utilizando o algoritmo MLP (*backpropagation*), criou um modelo de IDS com RNA e módulos distintos para realização da detecção, que leva em consideração

---

<sup>10</sup> Termo originário do inglês para o conteúdo propriamente dito de um pacote.

a RNA e a análise prévia de dados (diferentes etapas). Esta importante publicação contribuiu fundamentalmente na resolução de muitas dificuldades encontradas ao longo do desenvolvimento do presente projeto, como por exemplo, a utilização da representação binária e uso de palavras-chave para o conceito de detecção híbrida, apesar de na prática se diferenciarem pela forma com que os dados são inseridos na rede para análise e quais dados serão considerados. O modelo proposto pelo autor deste trabalho incorpora em um único vetor as informações a serem analisadas, compondo assim uma camada de entrada unificada.

Rocha (2006) apresentou em sua tese de mestrado a utilização de *Honeypots* (aplicações que visam simular um ambiente falho a fim de atrair hackers para a armadilha que gera *logs* ou replica um ambiente vulnerável) no treinamento de RNA para detecção de intrusão. Sua contribuição foi muito importante para a fundamentação teórica e analítica (foco da sua tese), principalmente nas considerações de quais algoritmos apresentaram melhores resultados, sendo o MLP o escolhido para o presente projeto.

Inspirado nestes exemplos citados acima, o presente projeto tem como principal diferencial a utilização de um conceito híbrido na coleta dos dados, extraindo as principais características relevantes à detecção de intrusão, sendo elas o tamanho do pacote, protocolo (TCP ou UDP), porta de destino, *flag* de fragmentação e uma lista de palavras-chave (com 16 palavras) que são inspecionadas dentro do *payload* do pacote. Este conjunto de variáveis é transformado em um vetor binário e enviado à RNA para classificação. Desta maneira, conseguiu-se utilizar os conceitos de assinatura e generalização da RNA pois, possui as características gerais somadas a uma lista de palavras-chave, que são consideradas indícios de atividade maliciosa.

Outro ponto diferencial estabelecido foi a adoção de licenças de software livre, linguagem de programação de fácil aprendizado e uma preocupação com a facilidade de compreensão do código-fonte como um todo, possibilitando a utilização de todo o material produzido neste projeto, por quaisquer interessados no assunto ou reutilizar em seus futuros projetos.

### 3 MÉTODO

Nesse capítulo, é abordado a metodologia utilizada para a elaboração deste projeto, a caracterização do tipo de pesquisa, as etapas metodológicas, a proposta de solução para chegar ao objetivo estabelecido e as delimitações do projeto.

#### 3.1 CARACTERIZAÇÃO DO TIPO DE PESQUISA

Para Galliano (1986, p. 6), “Método é um conjunto de etapas, ordenadamente dispostas, a serem vencidas na investigação da verdade, no estudo de uma ciência ou para alcançar determinado fim”. Pode-se ainda dizer:

[...] A própria significação da palavra 'método' indica que sua função é instrumental, ligando dois pólos [sic], a saber, um pólo [sic] de origem ou ponto de partida (estado de ignorância), outro pólo [sic] de destinação ou ponto de chegada (estado de conhecimento) [...]. O método corresponde ao grande empreendimento de construção do saber científico, da fase investigativa à fase expositiva [...]. O método se confunde com o processo por meio do qual se realiza a pesquisa científica. (BITTAR, 2003 apud LEONEL, 2007, p. 65).

Para Leonel (2007), a pesquisa é um processo que tem por objetivo investigar relações existentes entre características que englobem fenômenos, fatos, situações e coisas.

Considerando os critérios descritos por Leonel (2007), este trabalho é classificado como pesquisa aplicada, exploratória, quantitativa e bibliográfica, pois, através de incessante estudo de trabalhos relacionados, materiais de pesquisas e fontes de estudo, elabora-se um *software* e testa-se o mesmo, quantificando os dados e analisando o desempenho/resultado.

#### 3.2 ETAPAS METODOLÓGICAS

Este projeto possui as seguintes etapas, descritas na sequência, para alcançar seus objetivos:

- a) definir as regras de entrada e saída dos dados na rede à ser implementada;
- b) escolher o método de captura dos pacotes;
- c) implementar uma estrutura de tratamento e processamento dos dados capturados em dados estruturados para alimentar a RNA;
- d) definir os níveis de alerta e a classificação deles;
- e) implementar o algoritmo *backpropagation* adaptado à estrutura de dados que alimentará a RNA;
- f) treinar o algoritmo acima descrito, com exemplos de categorias de alerta previamente definidas.;
- g) testar o ambiente estruturado;
- h) analisar resultados obtidos.

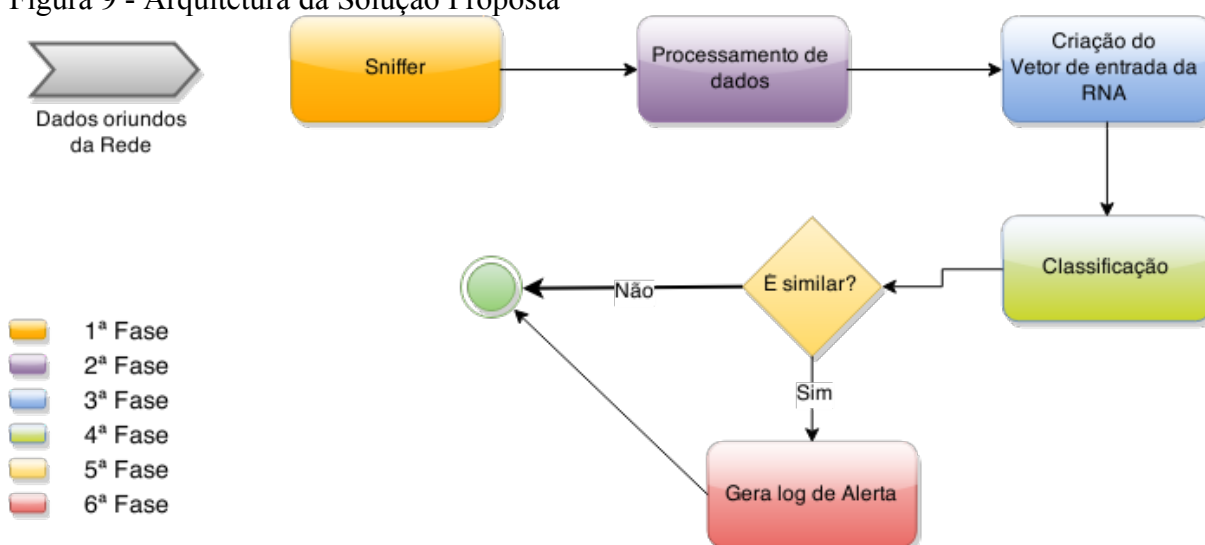
### 3.3 SOLUÇÃO PROPOSTA

Como solução proposta para este trabalho, criou-se um sistema de detecção de intrusão baseado em redes neurais (também chamado neste projeto de NNIDS) para generalização dos pacotes a fim de identificar, de acordo com os limites estabelecidos, pacotes similares aos treinados como maliciosos. Pode-se, assim, identificar ataques que não sejam totalmente idênticos aos conhecidos.

Propôs-se então a criação de um software que fará a captura do tráfego da rede, processamento desses dados capturados, tratamento dos dados e a classificação dos pacotes.

O diferencial de utilizar uma RNA treinada para identificar padrões de ataques, garante a percepção de pequenas alterações no código a fim de enganar os programas que utilizam assinatura para confirmação (como faz o SNORT). Para idealização deste projeto, o autor baseou-se nos princípios híbridos, considerando os pontos positivos da detecção por assinatura e tendo como exemplo o SNORT, somando às características de generalização estudadas e apresentadas nas RNA's. Na figura 9 encontra-se a arquitetura da solução proposta no presente trabalho.

Figura 9 - Arquitetura da Solução Proposta



Fonte: Elaboração do autor (2014)

### 3.3.1 1ª Fase – Entrada dos dados

Esta primeira etapa é composta pela entrada dos dados provenientes da interface e da porta, ambas definidas ao executar o sistema. Um método de captura de dados faz a captura e, caso o pacote seja válido para a análise (tenha *payload* e não seja criptografado), é então encaminhado para o próximo módulo.

É importante salientar que, se houver um *firewall* à frente do sistema, reduzirá drasticamente o volume de dados que o NNIDS deveria analisar, deixando seu desempenho mais eficiente e garantindo maior segurança com essa camada extra de proteção.

### 3.3.2 2ª Fase – Processamento de dados

Nesta etapa, o sistema faz uma triagem dos dados que entram para que o sistema possa compreendê-los. É feito a conversão do *payload* para hexadecimal e executada a limpeza do ruído, que é gerado durante o processo de conversão. O ruído é caracterizado neste



projeto pela presença do símbolo “/” no *payload*. Após a limpeza, os pacotes são encaminhados à próxima etapa.

### **3.3.3 3ª Etapa – Criação do Vetor de Entrada da RNA**

Nesta etapa, os dados previamente limpos são extraídos e utilizados para compor o vetor de 50 (cinquenta) posições (bits) que será lido pela RNA. Este vetor é composto pelos dados de tamanho do pacote, porta de destino, *flags*, protocolo e a lista de palavras-chave. Ao final deste processo, tem-se um vetor binário que é encaminhando à 4ª etapa.

### **3.3.4 4ª Fase – Classificação**

Os vetores de entrada são, então, inseridos para que a RNA faça a classificação, utilizando uma função de ativação “tangente hiperbólica”. A taxa de similaridade (resultado da classificação da RNA) é armazenada em uma variável, esta é encaminhada à próxima etapa.

### **3.3.5 5ª Fase – Definição dos limites de similaridade**

Nesta etapa, são definidos valores de aproximação, por exemplo, caso o valor de similaridade esteja entre 0,4 e 0,6 é classificado como “alerta de grau médio.” Se os valores estiverem entre 0,61 e 1,4, é classificado como “alerta de grau alto”. Valores fora desses limites são considerados tráfego normal ou não-maliciosos. Uma vez que a função de classificação calculará o valor de similaridade utilizando os valores dos pesos e os valores de entrada (não havendo limites de valores definidos), coube a esta fase a definição dos limites de alerta, todos os valores fora destes limites serão considerados pacotes normais ou limpos.

Os pacotes dentro dos limites de alerta são encaminhados à próxima etapa, os demais não.

### 3.3.6 6ª Fase – Gerar log

Nesta última etapa, são gerados os *logs* dos pacotes classificados como “alertas”. Cada gênero de alerta (alto e médio) tem um arquivo e é inserido neste, os alertas referentes, contendo os dados de origem, destino, porta e hora do pacote malicioso, bem como a taxa de similaridade.

## 3.4 DELIMITAÇÕES

Este projeto está delimitado a realizar:

- o desenvolvimento do algoritmo perceptron multicamadas (*backpropagation*), adaptado à estrutura necessária para classificar pacotes capturados;
- o treinamento da rede acima, considerando os níveis de alerta e tipos de pacotes;
- implementação da estrutura descrita na Figura 9;
- apenas alertará (IDS) em um arquivo de *logs*, cabendo ao administrador da rede utilizar o *software* que desejar para monitoramento dos *logs*.

Neste trabalho não será realizado:

- não garantirá 100% de acerto ou eficácia da rede;
- não analisará pacotes que sejam criptografados;
- não será implementado para outros sistemas operacionais diferentes do Linux;
- para fins puramente acadêmicos, a validação será realizada em um servidor Apache, porta 80 com média de 5 *exploits*, ensinados como maliciosos.

## 4 DESENVOLVIMENTO

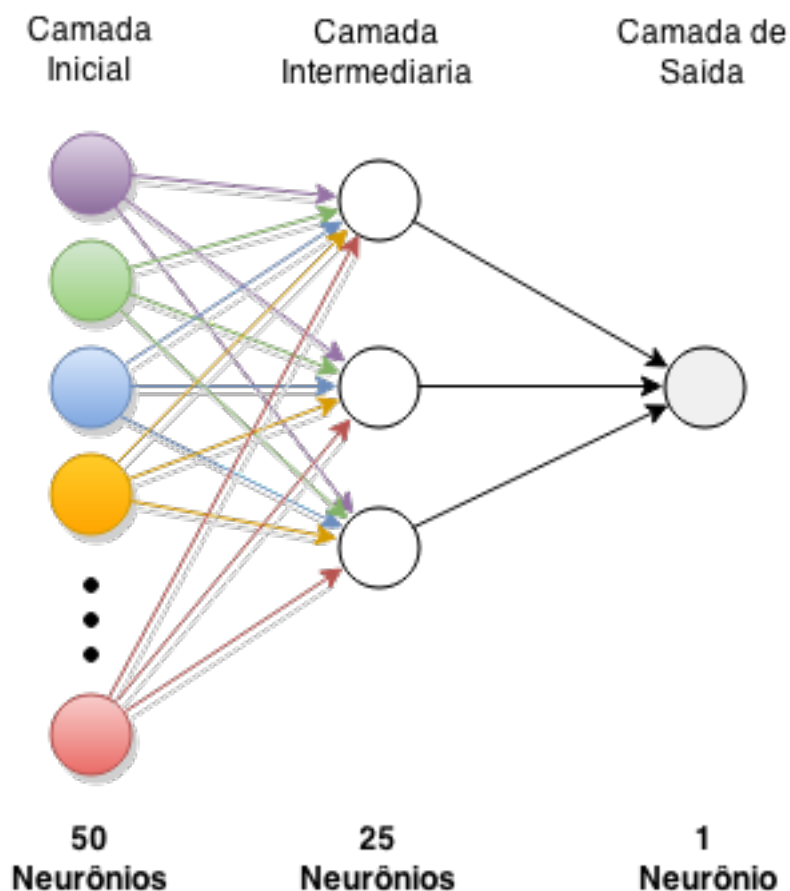
Neste capítulo, é apresentado o desenvolvimento do módulo de detecção de intrusão, utilizando redes neurais artificiais proposto para alcançar os objetivos deste trabalho. São relatados os problemas e dificuldades encontrados no decorrer do seu desenvolvimento, assim como as soluções e as técnicas adotadas para o projeto.

Primeiramente, é definida a estrutura da RNA, a base de treinamento com os exemplos ensinados à RNA e os módulos de tratamento e extração da informação para, por fim, ser classificada. Ao fim, são descritas as tecnologias utilizadas bem como as bibliotecas necessárias para a execução do software.

### 4.1 DEFINIÇÃO DA ESTRUTURA DA REDE NEURAL

Foi definido para este projeto a utilização de uma estrutura neural do tipo *Perceptron multicamadas* (tipo “*FeedForward*”) com treinamento, utilizando o algoritmo *backpropagation* e função *tangente hiperbólica* para ativação. Na figura 10 é apresentada a topologia ideológica da RNA:

Figura 10 - Topologia da RNA.



Fonte: Elaboração do autor (2015).

A Figura 10 apresenta a representação teórica das 3 camadas. A inicial com 50 neurônios (onde será inserido o vetor de entrada), uma camada intermediária ou “escondida” e a camada saída que será o resultado da rede.

#### 4.1.1 Estrutura e Topologia da RNA

Não há um número exato que seja ideal para a estruturação de uma rede neural, Segundo Russel (2004), é necessário entender o problema e encontrar um valor que satisfaça a necessidade da rede. Seguindo esta linha de raciocínio, utilizou-se cinquenta neurônios na camada de entrada, este valor confere à rede grande número de informações para computar, porém sem deixá-la extremamente “lenta” em seu funcionamento computacional.

Para a camada intermediária ou oculta, utilizou-se 25 neurônios. De acordo com Braga (et al., 2007, p. 55), uma camada intermediária seria suficiente para aproximar qualquer função contínua. O número de neurônios na camada intermediária depende, de acordo com:

- número de exemplos de treinamento;
- quantidade de ruído presente nos exemplos;
- complexidade da função a ser aprendida;
- distribuição estatística dos dados do treinamento.

Levando em conta os fatores citados acima, o projeto apresenta metade do número de neurônios da camada de entrada para a camada intermediária.

#### 4.1.2 Estrutura da camada de entrada

Esta estrutura foi pensada para melhor performance da RNA e melhor resultado na generalização dos dados. Esta camada possui 50 nodos ou neurônios de entrada, sendo assim distribuída, conforme figura 11:

Figura 11 - Topologia e distribuição da camada de entrada.



Fonte: Elaboração do autor (2015).

- **Protocolo:** Ocupa apenas 1 (um) neurônio binário, caso o protocolo seja UDP, o valor será 1, caso seja TCP, 0.

- **Flag fragmentação:** Ocupa apenas 1 (um) neurônio binário, caso a *flag* de fragmentação (MF – “*More Fragments*”) esteja ativa, o valor será 1; caso esteja inativado (ou seja, não há mais fragmentos), o valor será 0.
- **Porta de destino:** Ocupa 16 neurônios contendo o valor binário (16 bits) correspondente ao valor da porta de destino do pacote, ou seja, o valor da porta (exemplo, 80) é convertido para binário (16 bits) e inserido no vetor de entrada.
- **Tamanho do pacote:** Ocupa 16 neurônios e contém o valor binário (16 bits) correspondente à conversão do tamanho do pacote.
- **Lista de palavras-chave:** Ocupa 16 neurônios, cada um deles corresponde a uma palavra-chave em uma lista (que contém 16 palavras), caso alguma dessas palavras seja encontrada no payload do pacote, será ativado o neurônio com o valor 1 no local correspondente.

### 4.1.3 Aprendizado

As redes neurais do tipo MLP (*multilayer perceptron*) apresentam um poder computacional muito superior daqueles que não possuem camada intermediária, pois esta possibilita à rede resolver problemas linearmente não separáveis, ou seja, abrange quase todos os problemas matemáticos resolvíveis. (BRAGA et al. 2007).

O algoritmo utilizado para o treinamento da rede é o *backpropagation*, que trabalha em épocas, ao fim de cada época é calculado o erro e este valor é usado para o ajuste dos pesos, uma nova época então é iniciada com os pesos ajustados e, assim sucessivamente, até o fim das épocas ou a convergência da rede (erro zero). É o algoritmo mais utilizado, por ser robusto e muito eficiente.

A **taxa de Aprendizado:** neste trabalho, foi estipulada como **0.04** (zero ponto zero quatro). De acordo com Haykin (2001, p. 196), a taxa de aprendizado define a velocidade do aprendizado da RNA, ela é uma variável na fórmula de correção do erro na fase de “retropropagação”. Não há indicativos de valores exatos, porém, se muito alto ou baixa demais, pode danificar a capacidade da rede de aprender corretamente, por isso são necessários testes para definir o melhor valor, no caso do presente projeto, o valor **0.04** mostrou-se o mais eficiente.

Em face a um problema conhecido como “Mínimo Local”, utilizou-se no presente trabalho, o termo *momentum*.

Mínimos locais são pontos na superfície de erro que apresentam uma solução estável, embora não seja a saída correta. Algumas técnicas são utilizadas tanto para acelerar o algoritmo *backpropagation* quanto para reduzir a incidência de mínimos locais: [...] utilizar um termo *momentum*. (BRAGA et al., 2007, p. 67).

Como parâmetro do termo *momentum*, foi definido o valor **0.07** (zero ponto zero sete), também por apresentar melhor performance ao testar a RNA.

#### 4.1.4 Biblioteca *PYBRAIN*

Para a criação da RNA, utilizou-se da biblioteca externa PYBRAIN, importada para o sistema. Ela é responsável por todo o processo de criação, treinamento e ativação da RNA. Ela é um *framework* completo para auxílio e automação no desenvolvimento do sistema.

PYBRAIN é uma Biblioteca Aprendizado de Máquina modular para Python. Seu objetivo é oferecer algoritmos ainda poderosos e fáceis de usar contudo flexíveis para as tarefas de aprendizado de máquina e uma variedade de ambientes predefinidos para testar e comparar os seus algoritmos. (PYBRAIN, 2015, tradução nossa).

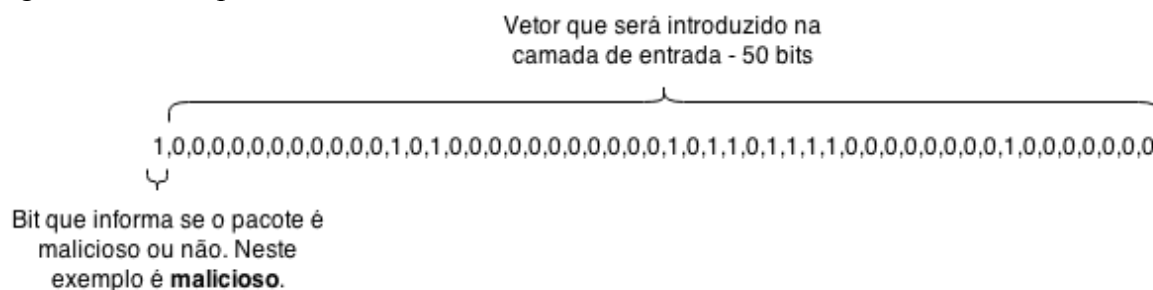
## 4.2 DEFINIÇÃO DA BASE DE CONHECIMENTO PARA TREINAMENTO

A base de conhecimento deste projeto consiste de exemplos de *exploits* e *payloads* mais comuns nos ataques hackers e, nos exemplos “limpos”, foram utilizados dados de navegação em *web sites* comuns e respostas padrões do servidor.

### 4.2.1 Arquivo da Base de Conhecimento

O *software* carrega o arquivo “training.txt”, disposto na mesma pasta do arquivo executável, todas as vezes que a função de treinamento é chamada. Este arquivo contém em cada linha um exemplo de vetor de entrada para a rede, acompanhado de um valor que é o valor da saída da entrada correspondente. O primeiro valor de cada linha pode ser -1 ou 1. O primeiro corresponde a um pacote “bom”, o valor 1 corresponde a um pacote classificado como malicioso. A Figura 12 ilustra este exemplo:

Figura 12 - Exemplo de Neurônio de treinamento.



Fonte: Elaboração do autor (2015).

O arquivo aceita também linhas de comentários, estas devem iniciar com o símbolo # (hashtag). Para um bom desempenho é aconselhável, no mínimo, 100 (cem) exemplos no arquivo de treinamento.

### 4.2.2 Coleta dos dados Maliciosos

Os dados ensinados à rede como maliciosos foram extraídos de modelos de *payloads* como, por exemplo, “*meterpreter*”, “*reverse shell*” e “*netcat*”. Segue a lista de *exploits* utilizada no treinamento:

- **Apache 2.2.11:** É um *exploit* publicado no dia 29 de Outubro de 2009 pela *sebug*. Sob o título “Apache 2.2.0 – 2.2.11 Remote exploit”, SSVD-ID: 12636.



- **Apache mod\_cgi - Remote Exploit (Shellshock)**<sup>11</sup>: *Exploit* publicado no dia 10 de Junho de 2014, referenciado sob EDB-ID: 34900.

Dados referentes à execução de *payloads* como “*meterpreter*”, “*shell reverso*” e “*netcat*” foram ensinados como maliciosos, possibilitando, assim, ao sistema detectar em tempo real a utilização destes meios de controle remoto e invasão de sistema.

### 4.2.3 Lista de Palavras-chave

Esta parte é composta de uma lista com dezesseis palavras ou trechos de código, convertidos para hexadecimal. Cada palavra possui uma posição fixa no vetor de entrada, por padrão, o valor é 0 (zero ou inexistente), no momento em que for detectada alguma palavra dentro do *payload* do pacote, é ativado na posição fixa desta palavra o valor 1 no vetor de entrada, confirmando a presença deste indicativo de alto grau, de uma possível invasão. Segue no quadro 3, as palavras cadastradas:

Quadro 3: Modelos de redes neurais

Posição	Conteúdo	Palavra a ser detectada
1	'2f62696e2f62617368'	/bin/bash
2	'2731202731273d27310d0a'	'1 '1'='1 (exemplo básico de injection)
3	'7831305a6668'	Trecho do exploit Apache 2.2.11
4	'7368682f62696e'	shh/bin
5	'61646d696e'	admin
6	'726f655f74'	root
7	'2f6367692d737973'	/cgi-sys
8	'6e63202d6c'	nc -l
9	'2f6574632f706173737764'	/etc/passwd
10	'6d65746572707265746572'	meterpreter
11	'7368656c6c5f65786563'	shell_exe
12	'6d656d6265724163636573735b22616c6c6f775374617469634d6574686f64416363'	memberAccess["allowStaticMethodAccess
13	'2f6574632f736861646f770d0a'	/etc/shadow

<sup>11</sup> <https://www.exploit-db.com/exploits/34900/>

14	'57838327863397865667838316e337862346f6e78636578666178643478'	parte do payload linux/x86/meterpreter/reverse_tcp
15	736f66742057696e646f7773205b56	parte do payload windows/shell_reverse_tcp
16	'2f6d616e616765722f736572766572696e666f20485454'	/manager/serverinfo HT do exploit/multi/http/tomcat_mgr_deploy

Fonte: Elaboração do autor (2015)

### 4.3 MÓDULO DE TRATAMENTO CAPTURA DE DADOS OU SNIFFER

Para a captura de dados do sistema, utilizou-se da biblioteca de apoio em Python, SCAPY.

SCAPY é uma poderosa ferramenta de manipulação interativa de pacotes. É capaz de forjar ou decodificar os pacotes de uma ampla série de protocolos, enviá-los, capturá-los, obter pedidos e as respostas, e muito mais. Ele pode lidar facilmente com a maioria das tarefas clássicas como a *scanning*, *tracerouting*, probing, testes unitários, os ataques ou a descoberta de rede[...]. (SCAPY, 2015, tradução nossa).

#### 4.3.1 Função *sniff*

A função “*sniff*” é nativa da biblioteca SCAPY e funciona como um *sniffer*, capturando cada pacote trafegando na rede que corresponda aos parâmetros definidos como filtro, este é responsável por separar o lixo da rede dos pacotes de interesse ao sistema. No filtro, é definido protocolo e porta(s), o presente sistema lerá tais dados da interface, ou seja, o usuário definirá as portas e a interface de rede que o sistema ficará “ouvindo”. Uma vez capturado o pacote, a função encaminhará o mesmo para um método chamado *pkt\_callback*.

#### 4.3.2 Método *Pkt\_callback*

Este simples método serve como intermediário entre a entrada dos pacotes e a extração de informações do mesmo. Serve para verificar se o pacote possui *payload*, caso não possua, ele descarta.

Uma vez verificada a existência do payload, o método verificará se o pacote não está encriptado (através da porta 443, HTTPS), caso seja, ele descartará o pacote, uma vez que o sistema não consegue interpretar nesta camada pacotes criptografados.

Ao fim das verificações, o pacote é encaminhado ao módulo de tratamento de dados.

### 4.4 MÓDULO DE TRATAMENTO DE DADOS

Nesta seção é explicado o processo de tratamento dos dados capturados pelo *sniffer* para um formato compreensível para a RNA. Esse processo consiste em:

- converter o *payload* para hexadecimal;
- limpar os dados obtidos na conversão para hexadecimal;
- extração dos dados do pacote;
- inserção da representação no vetor da camada de entrada.

#### 4.4.1 Conversão do Payload

Com o auxílio da biblioteca “*binascii*” importada no Python, é possível converter o payload extraído da camada *RAW* do pacote (*RAW* é a forma com que o dado trafega na rede), convertendo, assim, dados ilegíveis para uma representação passível de leitura.

É realizada então, a limpeza desta informação, retirando o ruído, considerado pela *string* “5c” (que em ASCII é equivalente ao símbolo “\”). Este valor, por fim, é armazenado em uma variável e utilizado na procura das palavras chaves.

#### 4.4.2 Extração de informações do pacote

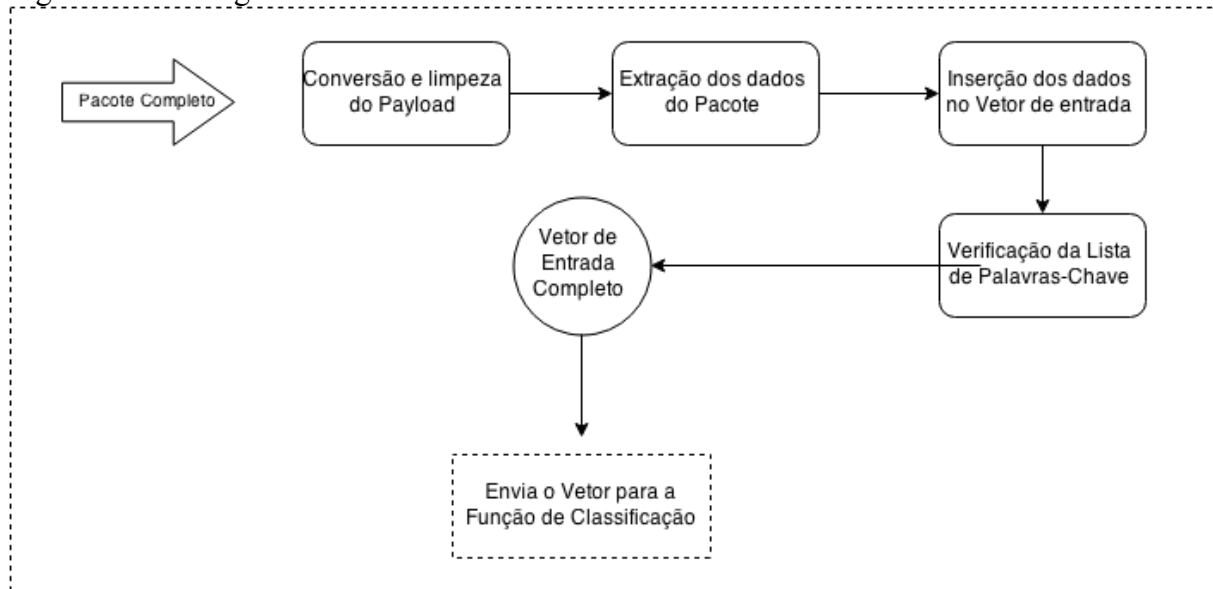
Nesta etapa, são extraídos os dados do pacote capturado pelo *sniffer* e inseridos em um vetor ordenado que, após completo, será inserido na rede neural para a classificação. São extraídos os seguintes valores:

- **Protocolo:** Quando o pacote é TCP, o primeiro campo do vetor recebe o valor 0 (zero), caso seja UDP, o campo receberá 1 (um). Corresponde a 2% do vetor.
- **Flag Desfragmentação:** Quando um pacote é dividido em vários pacotes menores, uma flag é sinalizada, contendo o valor MF ou *More Fragments* (tradução nossa). Caso esta flag esteja com o valor MF, o segundo campo do vetor receberá o valor 1 (um), caso contrário, permanecerá com o padrão 0 (zero). Corresponde a 2% do vetor.
- **Porta de Destino:** É extraído o número da porta para qual o pacote está destinado, este valor (inteiro) é, então, convertido para binário (de 16 bits) e cada valor inserido no vetor, sendo assim, este campo ocupa 16 posições do vetor de entrada (32% do vetor).
- **Tamanho do Pacote:** O valor com o tamanho do pacote é obtido e convertido para binário (de 16 bits), sendo este valor inserido no vetor em suas correspondentes posições, logo este campo também ocupa 16 posições (32% do vetor).
- **Lista de Palavras-Chave:** Nesta etapa o sistema possui uma lista com dezesseis palavras-chave convertidas para hexadecimal, cada uma corresponde a uma posição do vetor de entrada (as últimas dezesseis). É feita, então, a verificação se dentro do *payload* existe alguma destas palavras-chave, caso exista, o campo correspondente a palavra receberá o valor 1 (um), caso contrário, continuará com o valor padrão 0 (zero). Este campo corresponde a 32% do vetor.

#### 4.4.3 Fluxograma do módulo de tratamento de dados

Na Figura 13, podemos acompanhar o fluxograma, mostrando a sequência de fluxo dos dados:

Figura 13 - Fluxograma do módulo de tratamento de dados.



Fonte: Elaboração do autor (2015).

Conforme o fluxograma da figura 13, os pacotes entram através do *sniffer* e seguem para o processamento, limpeza e tratamento das informações, até estarem prontas para extração dessas informações e montagem do vetor que será analisado pela RNA.

#### 4.5 MÓDULO DE CLASSIFICAÇÃO

Este módulo recebe o vetor pronto para a inserção na RNA (composto por 50 neurônios com valores binários). Através da função de ativação da biblioteca *PyBrain* (utilizada para criar a RNA), o vetor é inserido e, após cálculo, retorna o valor de aproximação ou generalização da rede, onde “1” seria totalmente compatível com um pacote malicioso e “-1” seria um pacote totalmente idêntico a um exemplo “bom”.

Este módulo então verifica o percentual de proximidade, caso o valor retornado pela função de ativação esteja entre **0,3** e **0,6**, o sistema classificará o pacote com **Alerta Médio**. Caso o valor de similaridade esteja entre **0,61** e **1,4** o pacote emitirá um **Alerta Alto**.

#### 4.6 MODELO DE SAÍDA DE DADOS

Uma vez classificado o pacote, o presente módulo fará a inserção em um arquivo de *logs* os pacotes classificados como:

- **Alerta Médio:** É inserido no arquivo alerta\_medio.txt (presente na mesma pasta do arquivo Python).
- **Alerta Alto:** É inserido no arquivo alerta\_alto.txt (presente na mesma pasta do arquivo Python).

##### 4.6.1 Padrão de log

As informações a serem impressas no arquivo de *logs* correspondente ao gênero do alerta seguem o seguinte padrão:

Figura 14 - Padrão de *logs*.

---

```
Alerta de Nivel Alto - Taxa de Proximidade: 0.722943 - Data: Monday, 18. May 2015 08:20PM
IP de Origem: 192.168.243.128
IP de Destino: 192.168.243.1 Porta: 81
```

---

Fonte: Elaboração do autor (2015)

O *log* acima exemplifica um alerta de nível alto dia 18 de Maio de 2015 às 20:20, oriundas do ip 192.168.243.128, destinada à porta 81.

## 4.7 TECNOLOGIAS UTILIZADAS

A seguir, estão descritas as tecnologias utilizadas para o desenvolvimento do sistema NNIDS (*Neural Network Intrusion Detection System*). O sistema foi construído, utilizando a linguagem de programação Python, com auxílio de bibliotecas externas.

### 4.7.1 Python

De acordo com o site oficial da linguagem PYTHON (2015), Python é uma linguagem de programação que permite trabalhar rapidamente e integrar sistemas de forma muito eficaz. Foi escolhida para o presente projeto por tantas vantagens oferecidas:

- facilidade na escrita, programação e compreensão;
- ser poderosa, podendo alcançar alto grau de desempenho e escalabilidade;
- boa integração entre bibliotecas externas e internas;
- versatilidade no uso e execução de *scripts* e ou programas sem interface gráfica;
- possui ótima documentação;
- é fortemente *tipada*;
- é de alto nível, funcional, interpretada, orientada a objetos , imperativa, de *tipagem* dinâmica e forte.

Utilizou-se do Python na versão 2.7 por ser a mais preparada para suporte em relação às bibliotecas externas e estabilidade na execução.

### 4.7.2 Bibliotecas de apoio

Foram utilizadas diversas bibliotecas externas para auxílio na praticidade e versatilidade do software, sendo elas:

- PYBRAIN: Para a criação e execução da RNA, é versátil, possui excelente documentação e permite a alteração de diversos parâmetros suprimindo, assim, todas as necessidades do presente projeto com perfeição.
- SCAPY: Foi utilizada na captura e tratamento dos pacotes da rede, poderosa e dinâmica, esta biblioteca nos permite manusear e visualizar os pacotes com uma facilidade superior as demais.
- BINASCII: Biblioteca utilizada na conversão do payload, de binário para hexadecimal.
- OPTPARSE: Responsável por interpretar os parâmetros de entrada na execução do sistema, captura as variáveis inseridas pelo cliente como interface da rede a ser monitorada, porta e o modo de execução (monitoramento – 1 ou treinamento da rede – 2).
- DATETIME: Biblioteca responsável pela captura da hora e data exata e marcação destas no arquivo de *log*, quando houver incidência de alertas.

#### 4.7.3 Requisitos do sistema

Para executar o NNIDS (sistema descrito neste projeto), é necessário atender aos seguintes requisitos:

- sistema Operacional Linux;
- python na versão 2.7 instalado;
- ter instalado as bibliotecas PYBRAIN<sup>12</sup> e SCAPY<sup>13</sup>;
- executar em modo “root” ou Administrador;
- possuir na pasta raiz (mesmo diretório do arquivo executável) os arquivos: “brain.xml”, “training.txt”, “alerta\_alto.txt” e “alerta\_medio.txt”.

---

<sup>12</sup> <http://pybrain.org/docs/#installation>

<sup>13</sup> <http://www.secdev.org/projects/scapy/doc/installation.html>



## 4.8 APRESENTAÇÃO DO SISTEMA

O sistema é executado através do interpretador Python, preferencialmente em linha de comando e deve-se ater aos argumentos que precisam ser fornecidos ao sistema. São eles:

- **--mode ou -m:** Este argumento é referente ao modo de execução do sistema, se nesta execução deseja-se treinar a rede e atualizar os pesos (novo treinamento), deve-se escolher o valor 2. Caso deseja-se iniciar o NNIDS (monitoramento ou método principal), deve-se escolher o modo 1;
- **--interface ou -i:** Deve-se escolher a interface a ser monitorada pelo sistema;
- **--port ou -p:** Deve-se escolher a porta a ser monitorada pelo sistema.

### 4.8.1 Exemplos de inicialização

Na Figura 15 pode-se acompanhar alguns dos exemplos mais comuns de utilização (“*usage*”) do sistema:

Figura 15 - Exemplo de uso do sistema.

**Para iniciar o Sistema de Detecção (protótipo) monitorando a interface eth0 na porta 80:**

```
python nnids.py -m 1 -i eth0 -p 80
```

**Para iniciar o Sistema de Detecção (protótipo) monitorando a interface eth2 na porta 8080:**

```
python nnids.py -m 1 -i eth2 -p 8080
```

**Para treinar novamente a Rede Neural gerando novos pesos:**

```
python nnids.py --mode 2
```

Fonte: Elaboração do autor (2015)

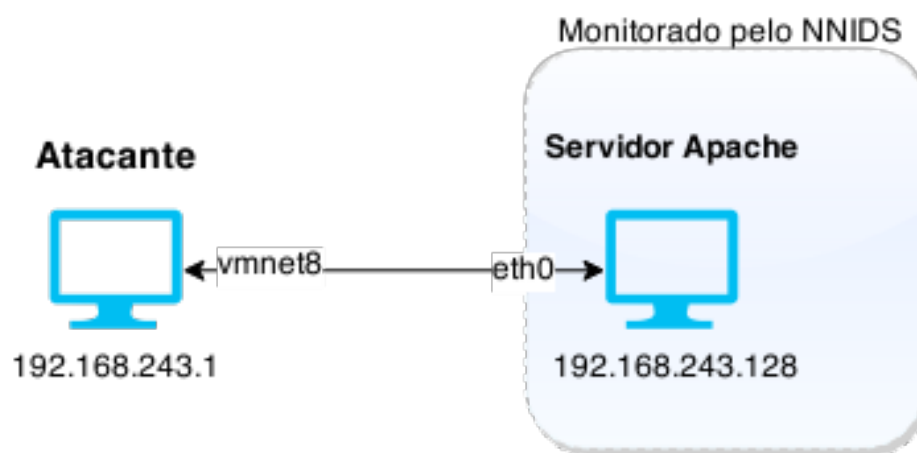
Estes exemplos mostram as principais formas de utilização do sistema, uma vez treinada a rede neural, a mesma só precisará ser treinada novamente, quando houver mudança na base de conhecimento (arquivo “training.txt”).

#### 4.9 VALIDAÇÃO

Na validação do sistema, foi averiguado se este atendeu aos objetivos definidos. Em um ambiente com o servidor Apache disponível na porta 80, o sistema diferenciou o tráfego normal do malicioso.

A validação foi realizada em um ambiente fictício, simulado com uma máquina virtual representando o cliente e a máquina real, representando o servidor Apache. Em uma média de 2000 pacotes, alguns poucos foram de *exploits* direcionados ao servidor Apache ou representando uma invasão (*shell* reverso ou *bind*). Na Figura 16 pode-se acompanhar a arquitetura dos cenários de validação:

Figura 16 - Arquitetura do Cenário de Validação.



Fonte: Elaboração do autor (2015)

#### 4.9.1 Primeiro Cenário

Neste primeiro cenário, criou-se, utilizando uma máquina virtual, um servidor apache com o NNIDS, monitorando a porta 80. Seguem os dados do cenário e validação:

Quadro 4: Dados do primeiro cenário de Avaliação do sistema

Quantidade total de pacotes analisados:	64
Tempo decorrido:	10 minutos
Comando executado:	Python nnids.py -m 1 -p 80 -i eth0
Porta monitorada no processo de validação:	80
Quantidade de alertas de Alto Grau	2
Quantidade de alertas de Grau Médio	1
Taxa de Acerto (NR. Pacotes detectados/maliciosos)	100%

Fonte: Elaboração do autor (2015)

Conforme pode-se observar, o sistema gerou 3 alertas (2 altos e 1 médio), confirmando com precisão de 100% a execução de 2 *exploits* (SSVD-ID 12636 e EDB-ID: 34900) e uma execução do software netcat (nc 192.168.243.128 80) com o argumento “root”, que o sistema classificou como possível ameaça. Os acessos normais foram feitos através de um *web site* fictício hospedado no servidor, com páginas de requisição, “index.html”, “index.htm”, acessos via “netcat”, páginas de erro e demais tipos de tráfego comum.

Totalizando, assim, neste primeiro cenário, 100% de acerto.

#### 4.9.2 Segundo Cenário

Neste segundo cenário, criou-se, utilizando uma máquina virtual, um servidor apache com o NNIDS, monitorando a porta 80, porém com páginas, contendo *backdoors* e diferentes *web sites* hospedados. Seguem os dados do cenário e validação:

Quadro 5: Dados do segundo cenário de Avaliação do sistema

Quantidade total de pacotes analisados:	243
Tempo decorrido:	20 minutos
Comando executado:	Python nnids.py -m 1 -p 80,4444 -i eth0
Porta monitorada no processo de validação:	80,4444
Quantidade de alertas de Alto Grau	2
Quantidade de alertas de Grau Médio	1
Falso-positivos ou falso-negativos?	1
Taxa de Acerto (NR. Alertas/maliciosos)	99,58%

Fonte: Elaboração do autor (2015)

No segundo cenário, foi detectado um falso-positivo, em que o sistema enquadrrou um pacote como alerta alto (taxa de proximidade igual à 1,33 sendo que a partir de 1,4 não é mais considerado alerta), que foi gerado a partir de um acesso a um web site no servidor (com uma porta alta). Nos demais casos, o sistema detectou corretamente a suspeita de um ataque ao entender “/etc/passwd” em um payload e corretamente um alerta alto oriundo de um acesso via meterpreter. Os acessos não-maliciosos ocorreram a um *web site* fictício, com vários arquivos, imagens e dinamismo, totalizando 243 pacotes transitados.

## 5 CONCLUSÕES E TRABALHOS FUTUROS

A primeira fase deste projeto consistiu no levantamento de projetos, literaturas, artigos e referências sobre os temas abordados. Com esta etapa de trabalho, concluiu-se que não há grande número de projetos que utilizem RNA's para auxílio na detecção de intrusão, principalmente utilizando conceitos híbridos para uma melhor performance. Com conceitos baseados no software SNORT, alguns *sniffers* e outros artigos teóricos, foi possível dar início ao desenvolvimento de um protótipo que visaria à identificação de pacotes, baseado em um aprendizado previamente realizado com pacotes maliciosos e não-maliciosos.

Este projeto tem como diferencial a união de diversos conceitos, como detecção por assinatura, generalização e extração de dados.

Durante o desenvolvimento do projeto, surgiram diversas dificuldades, como dificuldade para encontrar um tamanho fixo para o vetor de entrada, tipos de dados relevantes e o tratamento de dados.

Perante o que foi apresentado no trabalho, conclui-se que a utilização de redes neurais em auxílio à detecção de intrusão em sistemas é um caminho válido e apresenta resultados muito satisfatórios, porém é preciso percorrer um caminho pouco explorado.

Uma vez que o conhecimento é nosso maior bem, é justo que se compartilhe e ou disponibilize-o para todo(a) aquele(a) interessado em acessá-lo. Indo ao encontro com a afinidade do autor para com projetos de código e conhecimento livre, todo este projeto foi disponibilizado na internet através do GitHub, no link: <https://github.com/romullofb/nnids> sob licença GPL 3.0 que garante total liberdade.

### 5.1 TRABALHOS FUTUROS

A velocidade crescente no avanço tecnológico contribui e muito para a evolução dos sistemas, inclusive no aprimoramento de técnicas e projetos. Existe uma tecnologia ainda jovem, porém com forte potencial para a área de detecção de intrusão, é chamada de “FAST DATA”, que têm princípios similares aos do “*big data*”, porém voltada para dados imediatistas, ou seja, dados que têm um curto tempo de vida e precisam ser processados

rapidamente. Criando um sistema capaz de buscar, nesse “oceano” de informações, palavras-chave relacionadas à invasão, *exploit* e ou demais inserções (conforme se utilizou) neste projeto e, após a utilização de uma RNA para classificar o dado “descoberto”, poder-se-ia obter um poderoso NNIDS.

Pode-se ainda melhor desenvolver as formas de *output*, visual e geração de gráficos para aprimorar a experiência de uso do administrador da rede, considerando que este não é o foco no atual projeto, poder-se-ia incluir em uma futura versão deste protótipo.

É necessário manter-se sempre atualizado no que se refere às tecnologias envolvendo segurança da informação. Combinar tecnologias a fim de aumentar a segurança e performance, é um grande desafio.

## REFERÊNCIAS

- ALMEIDA, Igor N. S. de et al. Estudo da vulnerabilidade à coleta de informações por meio da técnica de phishing scam na FATEC Botucatu. **Revista Tekhne e Logos**, Botucatu, v. 2, n. 3, p. 1-3, jun. 2011. Disponível em: <<http://www.fatecbt.edu.br/seer/index.php/tl/article/view/23/80>>. Acesso em: 18 out. 2014.
- BIGUS, Joseph P. **Data Mining with Neural Networks**: Solving bussiness problems from application developement to decision support. New York: The McGraw-Hill, 1996.
- BITTENCOURT, Guilherme. **Inteligência Artificial**: Ferramentas e Teorias. 3. ed. Florianópolis: Editora da UFSC, 2006.
- BOSE, Nirmal K; LIANG, P. **Neural Network Fundamentals whith Graphs, Algorithms and Applications**. Singapore: McGraw-Hill, 1996.
- BRAGA, Antônio de Pádua; CARVALHO, André Ponce de Leon F; LUDEMIR, Teresa Bernarda. **Redes Neurais Artificiais**: Teoria e Aplicações. 2. ed. Rio de Janeiro: LTC, 2007.
- CARDOSO, Carlos; GUTIERREZ, Marcos Antônio. **Redes**. Rio de Janeiro: Axcel Books do Brasil, 2000.
- CERT. **Estatísticas dos Incidentes Reportados ao CERT.br**. Disponível em: <<http://www.cert.br/stats/incidentes/>>. Acesso em: 01 set. 2014.
- FERNANDES, Anita Maria da Rocha. **Inteligência Artificial**: Noções Gerais. Florianópolis: Visual Books, 2003.
- GALLIANO, A. G. **O método científico teoria e prática**. São Paulo: HARBRA Ltda, 1986.
- GERHARDT, Tatiana Engel; SILVEIRA, Denise Tolfo. **Métodos de Pesquisa**. Porto Alegre: Editora da UFRGS 2009.
- HAYKIN, Simon. **Redes Neurais**: Princípios e práticas. 2. ed. Porto Alegre: Bookman, 2001.
- HWANG, Hyunuk et al. A study on MITM (Man in the Middle) vulnerability in wireless network using 802.1 X and EAP. In: **Information Science and Security, 2008. ICISS. International Conference on**. IEEE, 2008. p. 164-170. Disponível em: <<http://nslab.kaist.ac.kr/courses/2012/test/paperlist/2-6.pdf>>. Acesso em: 19 out. 2014.
- KOHONEN, Teuvo. **Self-Organizing Maps**. 2. ed. Springer-Verlag Berlin Heidelberg New York, 1997.
- KOVÁCS, Zsolt Lászio. **O Cérebro e a sua mente**: Uma introdução à neurociência computacional. São Paulo: Edição Acadêmica, 1997.
- KUROSE, James F; ROSS, Keith W. **Redes de Computadores e a Internet**: Uma abordagem top-down. 5. ed. São Paulo: Addison Wesley, 2010.
- LEONEL, Vilson. **Ciência e Pesquisa**: Livro Didático. 2. ed. Palhoça: UnisulVirtual, 2007.

LIMA, Igor Vinícius Mussoi de. **Uma abordagem simplificada de detecção de intrusão baseada em redes neurais artificiais**. 2005. 94 f. Dissertação (Mestrado em Ciência da Computação)-Universidade Federal de Santa Catarina, Florianópolis, 2005. Disponível em: <<http://www.inf.ufsc.br/~bosco/grupo/MestradoIgor.pdf>>. Acesso em: 23 setembro 2014.

LUGER, George F. **Inteligência Artificial**: Estruturas e estratégias para a resolução de problemas complexos. 4. ed. Porto Alegre: Bookman, 2004.

MACHADO; Fernando, VOLTZ; Jonas, DIAS; Vagner. **Análise e otimização de assinatura SNORT**. São Leopoldo. Disponível em: <[http://www.230voltz.com.br/files/assinaturas\\_snort.pdf](http://www.230voltz.com.br/files/assinaturas_snort.pdf)>. Acesso em: 19 out. 2014.

NAKAMURA, Emilio Tissato; GEUS, Paulo Lício de. **Segurança de redes em ambientes corporativos**. São Paulo: Novatec Editora, 2007.

PYBRAIN. **Site Oficial Pybrain**. Disponível em: <<http://pybrain.org/>>. Acesso em: 2 fev. 2015.

PYTHON. **Site Oficial Pybrain**. Disponível em: <<https://www.python.org/>>. Acesso em: 1 jan. 2015.

REZENDE, Solange Oliveira. **Sistemas Inteligentes**: Fundamentos e Aplicações. Barueri: Manoele, 2003.

ROCHA, Daniel Lyra. **Utilização de um ambiente de honeypot no treinamento de redes neurais artificiais para detecção de intrusão**. 2006. 193 f. Dissertação (Mestrado em Engenharia de Redes de Comunicação – Universidade de Brasília. Brasília, 2006.

RUSSEL, Stuart; NORVING, Peter. **Artificial intelligence**: A modern approach. 2. ed. New Jersey: Prentice Hall, 2003.

\_\_\_\_\_. **Inteligência Artificial**: Referência Completa para Cursos de Computação. 2. ed. Rio de Janeiro: Elsevier, 2004.

SCHNEIER, Bruce. **Segurança.com**: Segredos e mentiras sobre a proteção na vida digital. Rio de Janeiro: Campus, 2001.

SEVERO, Diogo da Silva. **Otimização Global em redes neurais artificiais**. 2010. 53 f. Monografia (Graduação em Ciência da Computação)-Universidade Federal de Pernambuco, Recife, 2010.

SILVA, Denise Ranghetti Pilar da; STEIN, Lilian Milnitsky. Segurança da Informação: Uma reflexão sobre o componente humano. **Ciência e Cognição**, Porto Alegre, v. 10, p. 46-53, mar. 2007. Disponível em: <<http://www.cienciasecognicao.org/pdf/v10/m346130.pdf>>. Acesso em: 19 out. 2014.

SILVA, Lília de Sá et al. A Neural Network Application for Attack Detection in Computer Networks. **Proceedings, 2004 IEEE International Joint Conference on**. v2. Disponível em: <[http://mtc-m16.sid.inpe.br/col/sid.inpe.br/marciana/2004/12.03.10.53/doc/Lilia%201572\\_ijcnn2004.PDF](http://mtc-m16.sid.inpe.br/col/sid.inpe.br/marciana/2004/12.03.10.53/doc/Lilia%201572_ijcnn2004.PDF)>. Acesso em: 14 out. 2014.



SCAPY. **Site Oficial**. Disponível em: <<http://www.secdev.org/projects/scapy/>>. Acesso em: 19 mai. 2015.

SNORT. **Site Oficial Brasileiro**. Disponível em: <<http://www.snort.org.br>>. Acesso em: 19 out. 2014.

\_\_\_\_\_. **Manual de Instalação do Snort em Português**. 2002. Disponível em: <<http://www.snort.org.br/documentacao/ManualInstalacaoSnort-port.pdf>>. Acesso em: 19 out. 2014.

TRILLING, Steve. **As Mil Faces dos Ataques Cibernéticos**. Disponível em: <[https://www.symantec.com/region/br/enterprisesecurity/content/expert/BR\\_2305.html](https://www.symantec.com/region/br/enterprisesecurity/content/expert/BR_2305.html)>. Acesso em: 01 set. 2014.

WADLOW, Thomas A. **Segurança de Redes**: Projeto e gerenciamento de redes seguras. Rio de Janeiro: Campus, 2000.

## **APÊNDICES**

## APÊNDICE A – Cronograma

Quadro 6: Cronograma – 2014/2015

Etapa	Nov	Dez	Jan	Fev	Mar	Abril	Mai	Jun
Definir Regras de Entrada		x	x					
Escolher e definir regras SNORT		x	x					
Definir Níveis de alerta	x							
Implementar o algoritmo		x	x	x				
Treinar o algoritmo criado					x	x		
Testar o ambiente estruturado						x		
Ajustes finais e revisão							x	
Preparar apresentação para banca							x	
Entrega do TCC e Apresentação								x

Fonte: Elaboração do autor (2014).

## APÊNDICE B – Arquivo “Readme” do projeto

### Pré-requisitos do Sistema

É necessário ter instalado ou operante:

- Python na versão 2.7;
- A biblioteca PyBrain e seus dependentes;
- A biblioteca Scapy e seus dependentes.

### Inicialização

Para iniciar o sistema é necessário conferir se os seguintes arquivos estão dispostos na mesma pasta que o executável Python:

- **nnids.py** – Arquivo principal (executável Python);
- **brain.xml** – Arquivo contendo os pesos da Rede Neural;
- **training.txt** – Arquivo contendo a base de conhecimento;
- **alerta\_medio.txt** – Arquivo de output dos alertas de nível médio;
- **alerta\_alto.txt** – Arquivo de output dos alertas de nível alto.

### Modo de Uso ou Usage

Para iniciar o sistema de monitoramento, deve-se definir 3 argumentos: modo de operação, interface e porta. Exemplos:

#### **Para monitorar a porta 80 da interface 'eth0'**

- `python nnids.py -m 1 -p 80 -i eth0`

#### **Para monitorar a porta 80, 8080 e 4444 da interface 'eth0'**

- `python nnids.py -m 1 -p 8080,80,4444 -i eth0`

O modo de operação 1 é o principal, representando o monitoramento de intrusão.

### Treinamento da Rede Neural

Caso haja uma atualização da base de conhecimento ou modificação no código original, será necessário efetuar um novo treinamento da rede neural, este treinamento pode ser feito de maneira rápida através do comando:

- `python nnids.py -m 2` ou
- `python nnids.py --mode 2`

O modo de operação 2 treina a rede novamente.



