JACSON RODRIGUES CORREIA DA SILVA

SISTEMAS DE DETECÇÃO DE INTRUSÃO COM TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA
MINAS GERAIS – BRASIL
2011

JACSON RODRIGUES CORREIA DA SILVA

SISTEMAS DE DETECÇÃO DE INTRUSÃO COM TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 25 de fevereiro de 2011.

Alcione de Paiva Oliveira
(Co-orientador)

Luiz Henrique Andrade Correia

Carlos de Castro Goulart
(Orientador)

"Não tenha medo, pois eu estou com você. Não precisa olhar com desconfiança, pois eu sou o seu Deus. Eu fortaleço você, eu o ajudo e o sustento com minha direita vitoriosa."
Isaías 41, 10

AGRADECIMENTOS

- Agradeço à Deus, que sempre me acompanhou em todos os momentos de minha vida, que sempre me mostrou luz em caminhos escuros, que sempre despertou a esperança em meu ser e que sempre mostrou-se amigo e companheiro.
- À minha esposa, que me acompanhou nos momentos bons e ruins, que soube e/ou sempre tentou entender meus pensamentos. Que me fez quebrar a rotina em momentos necessários e que sempre esteve ao meu lado quando o apoio era necessário.
- Agradeço também ao meu orientador, que sempre foi paciente, que sempre confiou em meu trabalho e que soube acreditar que eu seria capaz, mesmo quando devida afirmação não parecia aflorar. Agradeço aos momentos que dispôs tempo para me ajudar com meu trabalho, mesmo abrindo mão de horas com sua família. Agradeço também aos momentos que pronunciou palavras de apoio e palavras que me fizeram crescer.
- A toda minha família, avó, pais, irmãs, cunhados e sobrinhas, que sempre souberam me entender quando eu dissera não ter tempo para estar com eles. Que sempre souberam me educar, que sempre demonstraram amor, carinho, companheirismo em tudo de minha vida.
- Agradeço a todos que me fizeram crescer, que ajudaram a construir o que eu sou nos dias atuais. A todos que me deram oportunidades, que tiveram paciência em me ensinar, em mostrar como se deve pesquisar e que quiseram compartilhar além de conhecimento, amizade. Que além de tudo isso, sempre demonstraram confiança em minha pessoa, em meu trabalho e em meu aprender.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ACRÔNIMOS

AG: Algoritmo Genético

CE: Computação Evolutiva

IA: Inteligência Artificial

IRC: *Internet Relay Chat*

MCC: *Multi-class Classifier*

MLFF: *Feedforward* Multicamada

MLP: *Multi-Layer Perceptron*

PDI: Política de Detecção de Intrusão

RNA: Rede Neural Artificial

SDI: Sistemas de Detecção de Intrusão

SDID: Sistemas de Detecção Distribuídos

SDIH: Sistemas de Detecção Baseados em Host

SDIR: Sistemas de Detecção Baseados em Rede

SIA: Sistemas Imunológicos Artificiais

SIH: Sistemas Inteligentes Híbridos

SN: Sistema Nebuloso

SPI: Sistemas de Prevenção de Intrusão

RESUMO

SILVA, Jacson Rodrigues Correia da, M.Sc., Universidade Federal de Viçosa, fevereiro de 2011. **Sistemas de detecção de intrusão com técnicas de inteligência artificial**. Orientador: Carlos de Castro Goulart. Co-Orientadores: Alcione de Paiva Oliveira e Mauro Nacif Rocha.

Devido ao aumento da quantidade de informações importantes sobre as Redes de Computadores, a segurança torna-se fundamental para garantir a integridade, a confidencialidade e a disponibilidade dos dados trafegados. Para melhorar a segurança, utilizam-se ferramentas, como Firewalls e Sistemas de Detecção de Intrusão (SDI). Atualmente, métodos de Inteligência Artificial (IA) são utilizados para melhorar tais ferramentas. Esta dissertação propõe, então, avaliar a melhoria das taxas de acerto dos Sistemas de Detecção de Intrusão utilizando algumas técnicas de Inteligência Artificial. São apresentados os conceitos sobre segurança e as medidas necessárias para aplicá-los, além de informações sobre invasores de sistemas computacionais. Também são apresentados os principais conceitos sobre os Sistemas de Detecção de Intrusão e algumas técnicas de Inteligência Artificial e uma revisão bibliográfica sobre SDIs implementados com técnicas de IA. O desenvolvimento desse trabalho iniciou-se com a implementação de uma Rede Neural Artificial, que teve suas características alteradas através de Algoritmos Genéticos para obter melhores taxas de acerto sobre conexões normais e anormais de uma rede de computadores. Também foi implementado um Sistema Nebuloso, utilizado posteriormente em junção com as Redes Neurais Artificiais para formar um Sistema Híbrido Inteligente. Após as implementações, as taxas de acerto para detecção de invasões e de tráfego normal foram obtidas e comparadas. Elas apresentaram, em todos os métodos de IA, taxas de acerto maiores que o sistema inicial utilizado. Foram apresentados, para cada método implementado, suas taxas de acerto em gráficos comparativos com os métodos implementados anteriormente e uma discussão dos resultados encontrados. A classificação correta dos ataques e do tráfego normal através das Redes Neurais Artificiais aumentou em até 17,6% com a utilização do Algoritmo Genético. Já o Sistema Nebuloso implementado apresentou um ganho modesto, da ordem de 5% na taxa de acerto de ataques e tráfego normal. Porém, com a junção das Redes Neurais ao Sistema Nebuloso, formando o sistema Neuro-Fuzzy, obteve-se um ganho nas taxas de acerto da ordem de 30% em relação ao trabalho original

implementado. Ao final, também são apresentadas as conclusões desse trabalho e algumas possibilidades de trabalhos futuros.

ABSTRACT

SILVA, Jacson Rodrigues Correia da, M.Sc., Universidade Federal de Viçosa, February of 2011. **Intrusion detection systems with artificial inteligence technics**. Adviser: Carlos de Castro Goulart. Co-Advisers: Alcione de Paiva Oliveira and Mauro Nacif Rocha.

Due the increase of the amount of important information on computer networks, security has became primordial to ensure the integrity, confidentiality and availability of data traffic. To improve security, there are useful tools such as Firewalls and Intrusion Detection Systems (IDS). Currently, methods of Artificial Intelligence (AI) are used to improve these tools. This work proposes to evaluate the improvement of the hit rates of Intrusion Detection Systems using some Artificial Intelligence techniques. They are presented the concepts of security and measures required to implement it, as well as some information about computer systems intruders. It is also presented the main concepts abow Intrusion Detection Systems and some Artificial Intelligence techniques and a review about IDS implemented with AI techniques. The development of this work has begun with the implementation of an Artificial Neural Network, which had its characteristics modified by Genetic Algorithms to improve its hit rates on normal and abnormal connections of a computer network. It was also implemented a Fuzzy System, which was then combined with the Artificial Neural Networks to create a Hybrid Intelligent System. After the implementations, the hit rates for detected intrusions and normal traffic were obtained and compared. The results showed that we have obtained higher hit rates for all methods when compared with the initial system. For each implemented method, the results were presented considering their hit rates using comparative charts with the previously implemented methods and a discussion about each new result. The correct classification of attacks and normal traffic by the Artificial Neural Networks increased up to 17.6% using the Genetic Algorithm. The Fuzzy System presented a slight gain of about 5% on hit rate of attacks and normal traffic. However, when Neural Networks and Fuzzy System were combined, forming the Neuro-Fuzzy System, we have obtained a gain around 30% on hit rate, when compared to the original work. Then, we present some conclusions of this work and some possible future work.

Capítulo 1

Introdução

As civilizações nunca foram tão interligadas, globalizadas, como nos dias atuais. A comunicação sempre foi um requisito básico do ser humano, que não só fala, mas também escuta, compreende, aprende e manipula as informações geradas por uma pessoa, um jornal, ou outro dos diversos meios encontrados. Sob toda esta comunicação, encontramos as Redes de Computadores, sempre recebendo e transmitindo dados para que essas informações possam ultrapassar os limites e deixar duas pessoas, ou mesmo máquinas, se comunicarem.

Devido à necessidade crescente de trocar, armazenar e manipular informações, milhares de dados passam pelas redes de comunicação que nos cercam e a cada minuto obtém-se um número incalculável de transações confidenciais e importantes, na qual a segurança existente às vezes é pouca, ou pior, engana aqueles que se sentem seguros.

O extravio e a má utilização de informações confidenciais é um ato comum, que existe, mas às vezes, é invisível aos olhos de um usuário. Vários são os fatores para que tal ato aconteça e os que procuram formas de deter esses ataques enfrentam desafios todos os dias, devido a diversidade e ao avanço dos ataques, como citado por Joseph Migga Kizza (KIZZA, 2005).

Assim como qualquer sistema evolui, as técnicas de invasão também evoluem (KIZZA, 2005) e não há como uma pessoa ou grupo de pessoas analisar manualmente e em tempo hábil o tráfego de uma rede de computadores, pela enorme quantidade de dados que possui. A análise desses dados deve sempre ser facilitada, devendo-se fazer

uma filtragem correta do conteúdo normal dos tráfegos das conexões. O gerenciamento de redes liga-se diretamente à coleta, filtragem e análise dos dados originados dos dispositivos de rede monitorados (BRACKMANN, 2006) e é necessário para a manutenção e bom funcionamento dos diferenciados sistemas em uma rede (KEMPFER, 2006).

Através desses dados, é possível gerar o conhecimento sobre o mal funcionamento, as tentativas de ataques, etc., do dispositivo, sendo esta informação um componente chave que deve ser gerenciado sistematicamente para a sobrevivência e o melhoramento do sistema (HOUARI, 2004).

Figura 1: O valor dos dados organizados (adaptado de (HOUARI, 2004)

Também em (HOUARI, 2004), foi proposto desenvolver um sistema de suporte a decisão capaz de lidar com uma grande parte de dados disponíveis distribuídos em um ciclo de vida de um projeto desenvolvido por uma empresa. Eles mostraram como uma grande quantidade de dados fornece meios para uma decisão. Acompanhando a Figura 1, pode-se ver que é necessário organizar os dados de forma coerente, transformando-os em conjuntos mais úteis. Cada conjunto é uma informação, que por sua vez, deve ser processada a ponto de relacionar sua causa, seu efeito e sua correlação, tornando-se então um conhecimento, que deve ser útil para a tomada de decisões.

Em estruturas de redes maiores e mais complexas, a quantidade de informações é enorme. A velocidade de análise e resposta dos gerentes pode não ser tão rápida quanto a necessária, sem comprometer a sua eficiência (KEMPFER, 2006). A falta de redes de

computadores em bom funcionamento pode levar a problemas como paradas de serviço, bloqueio na obtenção de informações importantes, falta de agilidade de operações, dentre outras.

Para garantir que os serviços em uma rede de computadores possam ser fornecidos e utilizados de forma a garantir o correto funcionamento, é necessário implementar diversos métodos que trarão consigo a segurança.

1.1 O Problema e sua importância

A informação manifesta-se na sociedade atual principalmente através das redes de computadores, apresentando um conjunto de dados úteis e significativos que necessitam cada vez mais de proteção (MAIOR et al., 2006).

Há algum tempo atrás, a invasão às informações internas e confidenciais era feita por técnicos com conhecimento sobre o sistema. Porém, nos dias atuais, a obtenção do conhecimento para se conseguir acesso às informações privadas, sem permissão, tem sido facilitada pela Internet. Vários usuários com pouco conhecimento podem obter e utilizar cada vez mais facilmente ferramentas e manuais de invasão disponibilizados na Internet.

Embora sejam inúmeras as formas de atacar, as invasões em computadores geralmente apresentam modificações sobre padrões de ataques, muitas vezes já conhecidos e que podem ser classificados e mapeados através de assinaturas ou modelos. Apesar das técnicas atuais apresentarem métodos para utilização de assinaturas, os sistemas de detecção de intrusão, principalmente os proativos, não são tecnicamente avançados o suficiente para detectar ataques sofisticados de profissionais treinados. Além disso, falsos alarmes de Sistemas de Detecção de Intrusão (SDI) são problemáticos, persistentes e preponderantes, causando perdas financeiras às organizações (DEBAR, 1992).

Os Sistemas de Detecção de Intrusão possuem soluções que dependem do conhecimento prévio do padrão de determinada invasão (SILVA e MAIA, 2004), não sendo capazes de aprender com novas formas de ataque.

Com o intuito de melhorar o reconhecimento de padrões, tornando-o capaz de

aprender novas informações e de poder ser atualizado com novas descrições de invasão ou tentativas de invasões ao sistema, iniciou-se a utilização de métodos de Inteligência Artificial (IA) na implementação dos Sistemas de Detecção de Intrusão (SDI). Porém, mesmo que a integração de IA aos SDI possa aprimorar as técnicas de defesa em sistemas de computadores, são poucos os sistemas atuais que adotam tal solução. Mesmo que essas técnicas possam demonstrar capacidade de melhorar a segurança da informação de empresas, instituições, etc., elas são pouco abordadas e têm pouca utilização. Seriam importantes estudos capazes de comprovar a capacidade de métodos de IA fornecerem aos SDIs a característica de aprendizado sobre novos tipos de ataques, tornando-os capazes de adaptar-se ao ambiente e de detectar mais ataques do que as ferramentas atualmente implementadas.

Outro problema é encontrar ferramentas que forneçam aos estudantes e à comunidade acadêmica a capacidade de aprendizado através da visualização total do código fonte do programa gerado. Tal característica demonstraria como o programa foi criado e implicaria na conclusão de como continuar de forma correta a pesquisa na área.

1.2 Hipótese

A hipótese desse trabalho é a de que a utilização de técnicas de Inteligência Artificial na implementação dos Sistemas de Detecção de Intrusão pode aumentar a eficiência na tarefa de detecção de ataques.

1.3 Objetivos

O objetivo principal deste trabalho é avaliar a utilização de algumas técnicas de Inteligência Artificial na melhoria das taxas de acerto dos Sistemas de Detecção de Intrusão. Para isso, é necessário aplicar algumas técnicas de IA sobre dados de invasão e analisar as taxas de acerto encontradas e compará-las com intuito de verificar o ganho de desempenho em cada abordagem.

Para se chegar ao objetivo traçado, foram adotadas as seguintes etapas:

- implementação de uma Rede Neural Artificial (RNA) para analisar os

dados de invasão;

- implementação de Algoritmos Genéticos (AG) para aumentar as taxas de acerto das Redes Neurais Artificiais;
- criar um Sistema Nebuloso (SN) para analisar os dados de invasão;
- unir os sistemas implementados, formando um Sistema Inteligente Híbrido, para verificar se poderiam alcançar taxas de acerto maiores na análise dos dados de invasão.

1.4 Resultados Alcançados

Após as implementações, seguindo os passos descritos na seção 1.3, as taxas de acerto quanto à detecção de invasões e de tráfego normal foram obtidas e comparadas. Elas apresentaram, em todos os métodos de IA, taxas de acerto maiores que o sistema inicial utilizado, implementado de acordo com o que foi descrito em Tavallaee (TAVALLAEE, 2009). Dentre os métodos de IA implementados, as RNAs obtidas pelos algoritmos genéticos apresentaram um maior acerto nas detecções de ataque e um menor erro no tráfego normal mal identificado. Já ao considerar uma taxa geral de acerto, os Sistemas Inteligentes Híbridos obtiveram a maior taxa de acerto. Foram apresentados, para cada método implementado, seus resultados em gráficos comparativos com os métodos implementados anteriormente e uma discussão dos resultados encontrados.

1.5 Organização do texto

O restante do texto possui a seguinte estrutura: no Capítulo 2, são apresentados os conceitos sobre segurança e as medidas necessárias para aplicá-la, além de informações sobre invasores de sistemas computacionais; o Capítulo 3 apresenta os principais conceitos sobre os Sistemas de Detecção de Intrusão (SDI), exibindo sua classificação por arquitetura e por métodos de detecção; no Capítulo 4 descritos alguns métodos de Inteligência Artificial (IA) e suas aplicações em SDIs; o Capítulo 5 apresenta as implementações de alguns métodos de IA aplicados aos SDIs, iniciando com uma Rede Neural Artificial (RNA) e sua melhoria através de Algoritmos Genéticos, passando por

um Sistema Nebuloso e terminando em um Sistema Híbrido Inteligente; os resultados dos testes efetuados sobre os métodos implementados são apresentados no Capítulo 6; e, finalmente, no Capítulo 7 são apresentadas as conclusões desse trabalho e algumas possibilidades de trabalhos futuros.

Capítulo 2

Segurança em Redes de Computadores

Várias tarefas rotineiras apresentaram uma crescente utilização e tornaram-se comuns através das facilidades oferecidas pelos computadores ligados em rede. Transações financeiras bancárias ou em lojas virtuais, comunicação através de correio eletrônico (e-mail) ou bate-papos, armazenamento de dados pessoais ou empresariais, dentre várias outras atividades circulam pelas redes de computadores (CERT.BR, 2010).

O volume desses dados valiosos são cada vez maiores e as redes devem trazer segurança para as pessoas que a utilizam. De acordo com Jie Wang, o objetivo da segurança é dar as pessoas a liberdade de desfrutar as redes de computadores sem medo ou preocupação de comprometer seus direitos e interesses (WANG, 2009). Segurança é um processo contínuo que visa proteger um alvo de ataque, sendo este uma pessoa, uma organização ou um sistema de computador (KIZZA, 2005).

A ideia básica entre todas as comunicações envolve o compartilhamento de alguma informação entre duas entidades, emissor e receptor, através de um canal que segue um conjunto de regras e protocolos (KIZZA, 2005).

A segurança envolve os seguintes elementos (CERT.BR, 2010; KIZZA, 2005; TANENBAUM, 2003):

- Confidencialidade: evitar a divulgação não autorizada de informações para terceiros, protegendo, por exemplo, informações bancárias ou de identificação, dentre outras que um usuário possui. O modo mais comum e

forte de manter a confidencialidade é através da criptografia;

- Integridade: impedir a modificação não autorizada de recursos, mantendo-os íntegros, protegendo que alguém possa, por exemplo, modificar a conta de destino no momento que o usuário fizer uma transferência bancária;
- Disponibilidade: disponibilizar os recursos com autorização a quem necessita, fornecendo somente a quem deve, o acesso aos dados e/ou recursos de um servidor, evitando, por exemplo, que a sobrecarga de dados causada por ataques impossibilite os clientes de acessar um recurso bancário.

Para aumentar a prevenção de um acesso não autorizado, os serviços de segurança devem abrigar mecanismos para fornecer os elementos descritos e possuir os seguintes procedimentos (CERT.BR, 2010; KIZZA, 2005; TANENBAUM, 2003):

- autenticação: processo de determinar com quem o usuário está se comunicando antes de revelar informações sigilosas ou entrar em uma transação comercial. Os métodos mais comuns para autenticação são as senhas de uso único, PGP (*Pretty Good Privacy*) e dispositivos de autenticação baseados em *tokens*;
- autorização: se refere a conceder privilégios no sistema aos processos e/ou aos usuários já providos de uma identificação. Este procedimento ocorre após a identificação segura de um usuário, concedendo-o privilégios, direitos e permissões internas no sistema;
- não-repúdio: trata de assinaturas, como, por exemplo, para provar que o cliente fez um pedido por um valor X, quando ele mais tarde afirmar que fez por um preço menor Y.

Nos dias atuais, devido à quantidade de computadores conectados a rede e a diversidade de *softwares*, a segurança é prejudicada pelas vulnerabilidades existentes. Segundo Joseph Migga Kizza (KIZZA, 2005), um sistema vulnerável possui uma condição, ou fraqueza, ou uma ausência de procedimentos de segurança.

As vulnerabilidades de um sistema são fraquezas encontradas no software ou no hardware de um servidor ou de um cliente que podem ser exploradas por alguém para ganhar acesso ou para derrubar uma rede. Existem não só no hardware ou no software que constitui um sistema computacional, mas também nas políticas e nos meios de segurança que são utilizados em uma rede e nos usuários e empregados que a utilizam (KIZZA, 2005).

Entre as fontes mais citadas de problemas de vulnerabilidade de segurança estão (KIZZA, 2005):

- as falhas no projeto, com três fatores principais:
  - fatores humanos: devido a memória do programador em esquecer de corrigir um erro, ao tempo curto para terminar o projeto, ao excesso de confiança, ao uso de algoritmos não padronizados ou não testados, a inserção de código malicioso e a utilização de códigos prontos, já utilizados em outros sistemas;
  - complexidade do software: devido a quantidade de entradas que podem existir e na dificuldade de nunca ter um conjunto completo de testes. Outro fator também é a facilidade de programação por levar muitos que, mesmo com a falta de conhecimento das especificações básicas de projetos, insistem em implementar programas que acabam apresentando muitos erros; e
  - a falta de utilização de *softwares* de fonte confiável: maioria das vezes são adotados *softwares* quaisquer, sem a preocupação de verificar sua fonte e sua qualidade. Outro prolema também é relacionado a popularidade do *software*, pois quanto mais popular, menos os consumidores aplicam testes para verificar sua segurança. Até mesmo o *software* livre pode ser perigoso, pois como o código é aberto, ele pode ser estudado para encontrar erros para um possível ataque, além disso, algumas pessoas já chegaram a inserir códigos maliciosos em códigos fontes, afetando os usuários que não testaram corretamente os

fontes obtidos;

- gestão de segurança ruim: acontece quando os responsáveis tem pouco controle sobre a implementação, administração e monitoramento da segurança em uma empresa. Isso fornece um leque grande de possibilidades de ataque, pois assim não podem ser identificadas novas ameaças na rede para correção, não se pode definir novas políticas de segurança e nem criar padrões para encontrar e corrigir erros, dentre outros problemas como a reação sobre um ataque;
- implantação incorreta de serviços: geralmente é o resultado de implantar sistemas incompatíveis. Dois módulos só podem ser implantados para trabalhar juntos se forem compatíveis. Assim, um módulo deve ser um aditivo para melhorar e não para causar falhas e/ou mal comportamento no sistema já implantado. Os sistemas incompatíveis geram problemas, devido a muitos detalhes que surgem, a não compreensão dos parâmetros mais intrínsecos do sistema, a má comunicação durante o projeto de implantação, a seleção de outros *softwares* e *hardwares* antes de entender o *software* que será implantado, a falta de preocupação com as questões de integração e o erro em entradas manuais;
- vulnerabilidade tecnológica na Internet: as falhas no projeto refletem na Internet, apresentando um índice de vulnerabilidades maior na última década (CERT, 2010). Várias vulnerabilidades são descobertas e, por muitas razões, não são relatadas. Elas podem ser colocadas em quatro categorias: vulnerabilidades do Sistema Operacional; vulnerabilidades de serviços (portas); *softwares* de aplicação baseados em erros; e *software* de protocolo de sistema;
- aprimoramento dos métodos e tecnologias de um invasor: a tecnologia utilizada por eles está em evolução assim como as outras tecnologias existentes. Existem implementações de vírus muito mais modernos que podem contaminar o mundo inteiro em tempo pequeno. Assim com a obtenção de materiais, ferramentas, etc., pelos aprendizes é muito mais

fácil nos dias atuais através de vários sites, além disso os invasores são curiosos e seus estudos caminham com dedicação;

- dificuldade de consertar sistemas vulneráveis: a dificuldade vem do tempo que uma correção de um sistema leva para ser liberada. O tempo de implementação das correções pode ser crítico, pois quanto mais demora para implementar, mais tempo o sistema permanece vulnerável. Além dessas atualizações demoradas, ainda tem-se o problema do administrador que costuma não atualizar seu *software* para corrigir os erros;
- pouca eficácia nas soluções de segurança: com o aprimoramento dos invasores, tem-se que sempre pesquisar e desenvolver técnicas melhores também para responder aos ataques, mas isso torna-se difícil devido ao número de vulnerabilidades que existem em produtos comerciais (que só podem ser corrigidas pela empresa que os fez), a quantidade de computadores conectados à Internet, aos métodos automatizados de ataque e a dependência dos usuários de utilizar mais a Internet, levando a utilização de mais e mais programas que podem ter falhas;
- engenharia social: é o invasor utilizar truques psicológicos para ganhar informações necessárias em uma empresa para invadi-la. São conversas bem convincentes que fornecem informações suficientes para invadir o sistema.

Todas essas vulnerabilidades estão presentes no cotidiano de cada pessoa, ou empresa, conectada às redes de computadores. O combate é feito através de um processo de segurança e do conhecimento das características do intruso.

Para garantir mais a segurança, deve-se adotar uma política de segurança para proteger o ambiente tecnológico e humano que podem ser afetados por invasões. A política de segurança é definida por um conjunto de regras, diretrizes e instruções que retratam os objetivos de segurança de uma empresa ou organização, determinados pelos seguintes itens (FRASER, 1997):

- Serviços Oferecidos e Segurança fornecida: envolve escolher utilizar ou

não um serviço baseando-se nos riscos de segurança que o envolvem;

- Facilidade de uso e Segurança: envolve escolher formas de acesso mais rígidas, assim mais seguras, ou mais fracas, assim mais fáceis de uso, como por exemplo, a não utilização de senhas;
- Custo da segurança e Risco de perda: ponderar custos monetários, de desempenho e de facilidade de uso com riscos de perda de privacidade, de perda de dados e de perda de serviços.

A política de segurança é importante e seu conjunto de regras deve ser seguido por cada pessoa envolvida no processo, tendo o propósito de informar aos usuários, equipe e administradores suas obrigações para a proteção da tecnologia e das informações da organização.

Para garantir uma boa política de segurança, seus procedimentos devem originar-se da administração, com a publicação de procedimentos aceitáveis, além da exigência do cumprimento das regras, até mesmo com penalizações, e da utilização de ferramentas de segurança. Deve ser clara a responsabilidade dos usuários, dos administradores, dos gerentes e de todos os envolvidos na empresa ou organização.

Além dessas características, uma política de segurança deve ser flexível, ou seja, não ser dependente de hardware ou software específicos e deve apresentar de forma clara os mecanismos de sua atualização. Deve ter um plano amplo, que defina, por exemplo, quais os serviços que serão fornecidos e quais suas áreas, quais os usuários que o poderão acessar, como será provido o acesso e quem o administrará.

O problema maior encontrado em uma política de segurança é sua implementação, que enfrenta: a falta de interesse das organizações ou empresas de investir recursos; e na educação dos usuários no cumprimento das normas definidas, o que pode transformar a política em um processo inoperante e sem eficácia (FRASER, 1997).

2.1 O Processo de Segurança

Segurança é um processo e não um estado. Envolve quatro passos, sendo estes o de avaliação, o de proteção, o de detecção e o de resposta ao ocorrido. Esta seção e a seção

2.2 baseiam-se no trabalho de Bejtlich (BEJTLICH, 2004), sendo citados explicitamente os demais autores em suas referentes contribuições.

2.1.1 Avaliação

A avaliação é o passo inicial e o mais importante por preparar para os outros passos do processo. Envolve determinar medidas que possam garantir a probabilidade de sucesso ao defender uma empresa ou instituição.

Nesta etapa, devem-se definir uma política de segurança e os serviços que estarão disponíveis em uma empresa. Este passo é muito importante, pois assim, pode-se definir como será o tráfego da instituição, tornando-se então uma tarefa mais fácil a detecção de tráfego suspeito na rede. Se a política definida for realmente rigorosa, qualquer outro tráfego não conhecido é um incidente na rede, sendo ou não um ataque.

Os incidentes originam também de tentativas de usuários internos de escaparem das regras definidas pela instituição para acesso ou de utilização da rede. Até que ponto isso pode ser considerado como invasão ou somente violação da política é avaliação da equipe de gerência interna da rede, baseando-se na política de segurança interna adotada (FRASER, 1997).

Esta definição de tráfego é muito útil e é mais visível através de um exemplo: uma empresa, ao definir que fornecerá somente o serviço de FTP aos seus clientes pode considerar que qualquer outro tráfego não relacionado, ou tentativa de acesso a uma porta diferente do serviço seja algo suspeito e que necessite ser monitorado e avaliado para verificar a possível tentativa ou real invasão.

2.1.2 Proteção

A proteção é a aplicação das medidas defensivas para reduzir a probabilidade comprometimento. Em Sistemas de Detecção de Intrusão (SDI) ou de monitoramento, este passo não é incluído, pois o objetivo é detectar a invasão caso a prevenção realmente falhe.

Os sistemas responsáveis pelo passo de proteção são os Sistemas de Prevenção de

Intrusão (SPI), que podem, até mesmo, controlar um dispositivo ou um *firewall*. Mesmo que sistemas SDI não sejam uma forma de prevenção, essa etapa ajuda na eficiência de tais sistemas, tornando-os mais efetivos.

Um problema crítico na auditoria de tráfego de uma rede é a quantidade de tráfego que deve ser monitorada, ao reduzir a quantidade de tráfego, ou as formas de acesso, tem-se um tráfego menor para monitorar e analisar. Afinal, o tráfego rejeitado não pode ferir uma empresa, e sim o tráfego aceito. Tem-se somente uma exceção, referente aos ataques de negação de serviço.

Alguns itens muito úteis na proteção de um sistema são:

- Controle de Acesso: um tráfego desconhecido na rede, que foge da política do controle de acesso, pode ser uma má configuração, ao invés de uma ação maliciosa. Quando uma ferramenta de monitoramento trabalha em conjunção com uma política de segurança bem definida e com um controle de acesso reforçado e apropriado, ele oferece a forma mais pura de auditoria da rede. Desvios da política são fáceis de identificar e resolver. Com políticas apropriadas e bem implementadas, os invasores tem poucos vetores de ataque e os responsáveis pelo monitoramento e segurança da rede podem ficar intencionalmente analisando somente o tráfego em porções limitadas;
- Afinação de Tráfego: muitos SDI gastam maior parte do seu tempo analisando pacotes replicados. A afinação do tráfego que garante um conteúdo com os passos de invasão é importante para agilizar e melhorar o desempenho de sistemas de identificação de intrusão;
- *Proxies*: Esses sistemas, inseridos entre os cliente e servidores, são importantes por razões de segurança, monitoramento, ou desempenho, já que um cliente não tem como se conectar ao servidor sem antes se conectar ao *proxy*.

2.1.3 Detecção

A detecção é o processo de identificar os intrusos, coletando, identificando, validando e escalonando eventos suspeitos. As invasões são violações da política ou acidentes na segurança dos computadores de uma rede.

A detecção requer quatro passos:

- Coletar: envolve acessar o tráfego com intenção de inspecionar e armazenar informações úteis. Dentre os erros mais comuns de coleta, podem ser citados a má configuração ou má aplicação de filtros ou regras para evitar eventos indesejáveis, a implantação de enlaces com a capacidade superior à capacidade do sensor de detecção e a combinação de equipamentos sem o conhecimento básico de sua tecnologia;
- Identificar: uma vez que todo o tráfego é transformado em um tráfego observável, é hora de obter informações dele, reconhecendo pacotes que não são comuns à rede. O tráfego observado pode ser caracterizado em:
  i. tráfego normal : é tudo que é esperado na rede de uma empresa;
  ii. tráfego suspeito: é o tráfego que trás características estranhas, incomuns, mas não causam danos aos bens da empresa;
  iii. tráfego malicioso: é qualquer informação que possa trazer um impacto negativo a postura de segurança da empresa, se enquadrando os ataques de todos os tipos realizados com sucesso à empresa;
- Validar: atribuir aos eventos uma categoria de incidente preliminar. Essa categoria de incidentes classifica os eventos como advertências ou, às vezes, simplesmente como indicadores, que são evidências do ataque ou, pelo menos, algo que necessite de uma investigação adicional, sempre alertando aos analistas que houve algo malicioso ou uma falha de segurança.

  Os eventos podem ser assimilados em uma das categorias:

  i. acesso sem autorização como administrador: ocorre quando uma

ferramenta ou alguém adquire sem autorização o controle de administração de outra parte do sistema;

ii. acesso de usuário sem autorização: ocorre quando uma uma ferramenta ou alguém sem autorização ganha controle de uma conta de usuário que não seja de administrador;

iii. tentativas de acesso sem autorização: ocorre quando uma ferramenta ou alguém sem autorização tenta obter acesso de administrador ou usuário do computador;

iv. ataque de negação de serviço com sucesso: ocorre quando o adversário (pessoa que ainda não invadiu o sistema) consegue ter ações prejudicais aos serviços e processos de uma máquina alvo ou de uma rede. Esses ataques podem consumir ciclos de CPU, banda, espaço em disco, tempo de usuário e vários outros recursos;

v. prática de segurança ruim ou violação da política: ocorre quando uma operação de monitoramento detecta uma condição que expõe a todos informações que permitem explorar os clientes, ou quando o cliente infringe a política, utilizando, por exemplo, serviços sem autorização;

vi. reconhecimento/provas/explorações: ocorre quando um adversário tenta aprender sobre uma rede ou um sistema alvo, com a intenção presumida de mais tarde comprometer este sistema ou rede;

vii. infecção por vírus: ocorre quando um sistema é infectado por um vírus ou *worm*. Um vírus depende da interação do humano para propagar-se e/ou pode se acoplar em um arquivo hospedeiro, como em um e-mail, um documento ou uma página da Internet. Já os *worms* são capazes de propagar por si só sem a capacidade de interação de homens ou de arquivos hospedeiros.

Essas categorias são indicações de atividade maliciosa, contudo classificar um evento na categoria i ou ii requer um grau mais alto de confiança nos dados que foram coletados no evento, pois os invasores podem ter

efetuado mudanças antes da coleta desses dados;

- escalonamento: é o processo de encaminhar os eventos obtidos às pessoas que tomarão decisões sobre os mesmos, sendo estes os clientes ou os supervisores de uma organização. Mesmo nessa fase, nem todos as indicações e advertências devem ser encaminhadas aos clientes, pois podem ser casos em que o incidente ainda não chegou a acontecer.

2.1.4 Resposta

A resposta é o processo de validar os frutos da detecção e tomar medidas para remediar as invasões. Existem dois processos de resposta:

- contenção do incidente em curto prazo: consiste em impedir que o incidente continue na rede. Geralmente os passos tomados são impedir fisicamente o acesso à máquina alvo, ou instalar uma nova regra no roteador de filtro ou no *firewall* para proibir o tráfego de entrar ou sair do alvo; e
- monitoramento de emergência: consiste em capturar todos os dados sobre o IP invasor. Quando todo o tráfego já é armazenado, isso não é necessário.

2.2 Características do intruso

Segundo Bejtlich (BEJTLICH, 2004), é importante ter em mente os passos que correspondem a um ataque, mas também é importante sempre lembrar que alguns invasores são mais inteligentes que a pessoa que implementou a segurança, que vários invasores são imprevisíveis e que a prevenção eventualmente falha. Os passos de um atacante envolvem o reconhecimento, a exploração, o reforço, a consolidação e o saque. Somente efetuando esses passos é que o invasor pode tirar vantagem de uma vítima.

O reconhecimento é o processo de validar a conectividade, enumerar os serviços e verificar por vulnerabilidade de aplicações. Para obter informações de um servidor, o atacante precisa fornecer pacotes válidos TCP ou UDP, pois precisa comunicar com o

servidor utilizando sua linguagem e suas regras. Existem exceções como, por exemplo, fazendo uma busca detalhada de informações sobre o alvo (*footprinting*).

A exploração é o processo de violação, que corrompe ou penetra os serviços de um alvo. Alguns exemplos de violação são: acessar o servidor por algum serviço utilizando um acesso legítimo ou não, por exemplo, através de um nome e senha roubado; criar uma execução de um serviço não prevista por seus programadores; e acessar um sistema, ganhando privilégios sem interromper um serviço.

O reforço é o estágio quando os intrusos aproveitam o acesso sem autorização para ganhar capacidades adicionais no alvo. De modo mais significativo, os invasores geralmente instalam meios de comunicar com o mundo externo, como colocar túneis de acesso mais complicados de serem encontrados, os chamados *backdoors.*

A consolidação já ocorre quando o invasor se comunica com seu alvo através de um *backdoor*. Há três casos de sua consolidação, o primeiro consiste em abrir uma porta por onde um invasor pode se conectar ao programa do *backdoor*, o segundo consiste no programa do *backdoor* se conectar ao IP do invasor e o terceiro consiste no programa do *backdoor* se conectar a um *Internet Relay Chat* (IRC) para receber as instruções de funcionamento.

O saque envolve roubar informações do alvo, construir uma base de ataques ou fazer qualquer outra coisa que o invasor deseje. As chances de detecção estão presentes em cada um desses passos de ataque:

- no reconhecimento, a probabilidade de detecção é numa faixa de média para alta, pois para catalogar as máquinas, os serviços e as aplicações, os atacantes devem executar serviços de descoberta durante um longo tempo, utilizando padrões normais de tráfego. Nesse momento, os atacantes revelam-se pelas diferenças entre seu tráfego e o tráfego legítimo de um usuário;
- na exploração, a probabilidade de detecção é alta, pois para efetuar tentativas de acesso, os atacantes utilizam *exploits* nos serviços oferecidos ou ofuscam o tráfego dos *exploits*. Essas ferramentas não apresentam um tráfego legítimo, assim um SDI pode ter várias assinaturas para detectar diferentes ataques;

- no reforço, a probabilidade de detecção também é alta, pois as ferramentas utilizadas pelos atacantes para obter mais privilégios ou para disfarçar a invasão causam uma atividade suspeita nos servidores que pode ser facilmente acompanhada e identificada;
- na consolidação, a probabilidade de detecção é numa faixa de baixa para média, pois o atacante tem o controle total através da comunicação de sua máquina com a máquina alvo, seus limites são impostos somente pelo controle de acesso e de tráfego dos dispositivos da rede. O perfil do tráfego é o único que pode identificar padrões desconhecidos que correspondem a utilização do *backdoor* pelo atacante;
- e no saque , a probabilidade de detecção também é numa faixa baixa para média, pois o tráfego do atacante vem de uma máquina "de confiança". Uma forma possível para detecção seria conhecer o trabalho dos sistemas internos, para assim informar divergências.

Agora que os conceitos necessários sobre segurança foram abordados, o Capítulo seguinte apresentará os conceitos sobre Sistemas de Detecção de Intrusão e como os mesmos podem ser utilizadas para melhorar a segurança de uma Rede de Computadores.

Capítulo 3

Sistemas de Detecção de Intrusão

Existem várias tecnologias que protegem o perímetro das redes de computadores contra invasores e ameaças externas. Porém, esses sistemas não podem parar invasores que obtém acesso autenticado e usufruem do sistema como se fossem usuários legítimos. Devido a isso, torna-se importante monitorar as informações contidas nos pacotes que atravessam o *firewall* para analisar como os usuários, legítimos ou não, utilizam seus computadores, monitorando assim suas atividades e permitindo a detecção das ações de invasão (WANG, 2009).

Segundo Costa (COSTA, 2007), os primeiros conceitos sobre sistemas capazes de detectar intrusões foram propostos por James P. Anderson em 1980 (ANDERSON, 1980). Segundo Wang (WANG, 2009), os sistemas para detectar invasões iniciaram-se com Dorothy Denning e Peter Neumann também em meados de 1980. Na época, eles observaram que os invasores agiam de forma diferente dos usuários normais do sistema e que as diferenças poderiam ser medidas, tornando possível a análise quantitativa do ataque. Para identificar os eventos anormais nos sistemas, torna-se necessária a criação de ferramentas automatizadas que se baseiam nas operações ocorridas nos sistemas de administração, nos protocolos de rede, nas estatísticas computacionais e na mineração de dados.

Essas ferramentas denominam-se Sistemas de Detecção de Intrusão (SDI), sendo sistemas automatizados que procuram por indícios de intrusão direta ou indireta, incluindo invasões que já aconteceram ou que estão atualmente em curso, e que notificam

os administradores para que possam agir apropriadamente. Essas ferramentas automatizadas para detectar invasões tem o objetivo de identificar as atividades de invasão que ocorreram ou que estejam acontecendo em uma rede. Quanto mais rápida for a detecção, menores serão os prejuízos causados pela invasão (WANG, 2009).

Com uma abordagem preemptiva ou evolucionária, os SDI podem monitorar uma rede e agir assim que observarem uma atividade suspeita, ou podem se basear em *logs*, agindo com segundos de atraso à invasão (ANÔNIMO, 2000). Alguns sistemas de detecção de intrusão podem também tomar medidas contra as invasões, esses sistemas recebem o nome de Sistemas de Prevenção de Intrusão (SPI). Um SPI pode mudar regras de perímetro, isolar os sistemas afetados ou finalizar seus serviços (WANG, 2009).

A metodologia básica de detectar invasores é através da auditoria do sistema, que possui dois tipos: a que trabalha com as informações estáticas de configuração, também chamada de perfis de segurança; e a que trabalha com os eventos dinâmicos do sistema. O primeiro define conjuntos de valores pré-configurados de parâmetros de segurança, como, por exemplo, qual o máximo de logins sem sucesso, qual o tamanho mínimo e o tempo de vida de uma senha, dentre vários outras medidas de configuração. Já o segundo trata da análise e gravação de eventos que ocorrem no sistema, onde as informações registradas devem conter no mínimo os campos (WANG, 2009):

- o sujeito: autor do evento;
- a ação: atividades do sujeito, como a execução de um comando, o ato de abrir um arquivo, etc.;
- um objeto: informação armazenada somente quando existe. É um item envolvido em uma ação, como por exemplo, um arquivo aberto;
- uma exceção: informação armazenada somente quando ocorre. Pode ser, por exemplo, uma falha ao abrir um arquivo ou ao executar um programa;
- o recurso utilizado: processamento, memória, etc; e
- a marcação de tempo.

Através das informações adquiridas do sistema, os SDI podem analisar os eventos

e comportamentos para detectar a invasão e produzir os alarmes correspondentes. Dentre os sistemas de detecção, as etapas comuns que possuem são (WANG, 2009):

- avaliação: que verifica as necessidades de segurança necessárias de um sistema e produz seu perfil de segurança;
- detecção: que coleta os eventos da utilização do sistema e analisa-os para detectar atividades de invasão. Também pode analisar os comportamentos de alguns programas, para verificar se suas atividades são comuns. O objetivo dessa etapa é identificar anomalias, baseando-se em regras ou/e em medidas quantitativas;
- alarme: que emite alertas quando um atacante consegue acesso ao sistema. Pode também classificar os alarmes e especificar como responder ao ataque.

Um SDI também pode conter uma Política de Detecção de Intrusão (PDI) que são utilizadas para identificar as atividades de invasão, especificando quais dados devem ser protegidos e como eles devem ser protegidos. Elas também podem especificar quais os tipos de atividades que são consideradas como invasões e como responder quando uma atividade suspeita é identificada (WANG, 2009).

Um PDI ideal deve ser simples, efetivo e fácil de implementar, com o mínimo de alarmes falsos. Isto é, um PDI não pode permitir que um SDI indique algo normal como anormal (Falso Positivo) ou algo anormal como normal (Falso Negativo). Os Falso Positivos e os Falso Negativos são complementares e a redução de um, pode causar o crescimento de outro. Pode-se, por exemplo, aceitar mais tipos de atividades como normal, reduzindo assim os Falso Positivos e aumentando os Falso Negativos. Pode-se também, classificar mais tipos de atividades como anormais, o que diminuiria os Falso Negativos e aumentaria os Falso Positivos (WANG, 2009).

Quando um usuário acessa um computador de uma rede e utiliza seus recursos, suas atividades podem ser interpretadas como eventos, que incluem em utilizar o sistema operacional, os aplicativos do sistema operacional ou os aplicativos de sua produção. Quando agrupam-se uma sequência de eventos ou uma coleção de várias sequências de

eventos, tem-se o denominado comportamento, que é aceito quando um usuário legítimo segue a política de segurança, ou não é aceito quando possui uma sequência de eventos que violam a política de segurança, sendo assim denominado mal comportamento (WANG, 2009).

Os SDI podem classificar os comportamentos em níveis com alta, média e baixa prioridade: os de baixa prioridade são os comportamentos normais aceitos pelo sistema; os de alta prioridade tem um comportamento anormal que deve ser rejeitado pelo sistema; e os de média prioridade são comportamentos que o sistema não pode classificar com as informações que possui até o momento, requerendo que o sistema obtenha mais informações para fazer uma classificação melhor (WANG, 2009).

Além dessas características, dependendo de quando as detecções são realizadas, os SDI podem ser divididos pelo momento em que fazem a detecção das invasões, que podem ser (WANG, 2009):

- detecções em tempo real: analisam os dados quando eles chegam;
- detecções em lote: analisam os dados quando um conjunto de dados coletados chegou em um tamanho específico; e
- detecções periódicas: analisam os dados periodicamente, em tempos pré-determinados.

Os SDI também podem ter diferentes locais para agir, sua arquitetura define se atuam distribuídos na rede, em um máquina específica, ou das duas formas.

3.1 Arquitetura dos SDIs

A visão de um SDI depende de sua localização, da forma e de onde obtém suas informações. Quanto a sua funcionalidade, podem ser classificados em:

- baseados em Rede (SDIR);
- baseados em *Host* (SDIH);
- distribuídos (SDID).

Suas coletas de informações podem ser específicas da aplicação, do *host*, ou da sub-rede, através da captura e análise de pacotes, da análise dos arquivos de *logs*, do monitoramento das chamadas do sistema e da observação do sistema de arquivos (BEALE, 2004).

3.1.1 SDI baseado em Rede

Os Sistemas de Detecção Baseados em Rede (Figura 3.1.1) monitoram um segmento de rede inteiro ou uma sub-rede através do modo promíscuo de uma interface de rede, identificando quais pacotes tem comportamento de nível alto ou médio e enviam mensagens de aviso ao gerente de redes responsável por responder aos alarmes. Podem também armazenar os pacotes em um local específico para uma possível análise futura, ou para garantir provas contra as invasões correntes (CERT.BR, 2010; WANG, 2009).

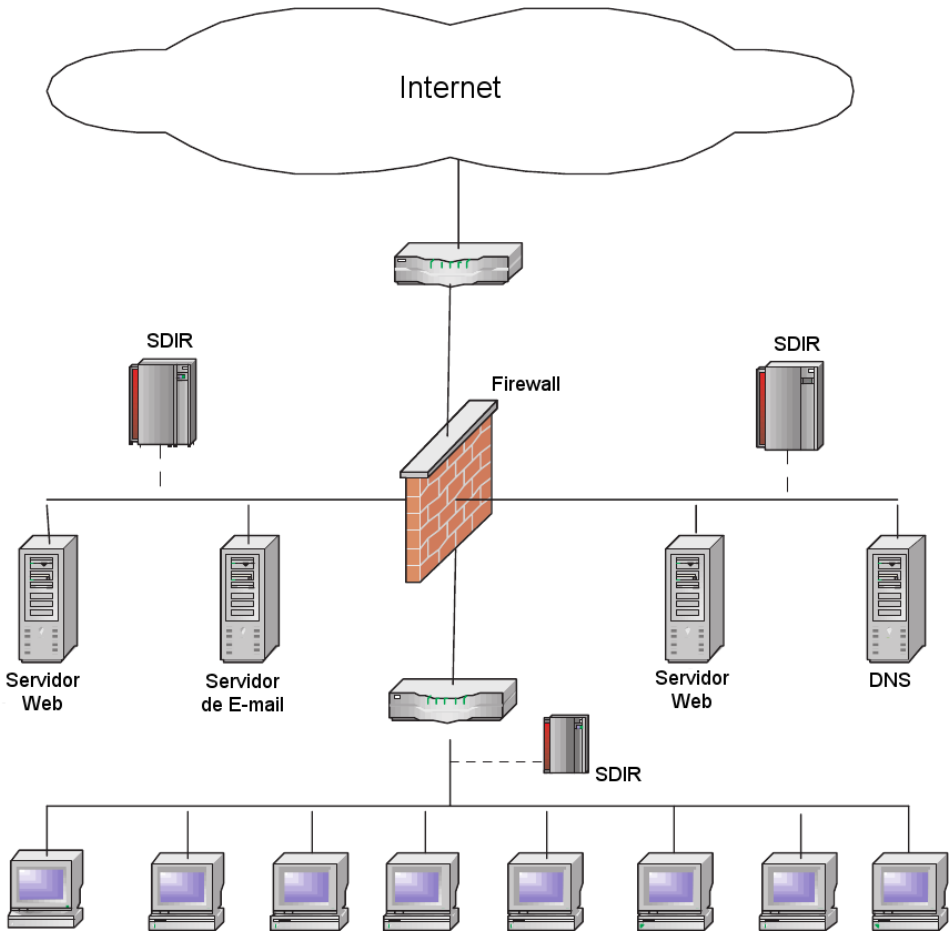


Figura 3.1.1: Sistemas de Detecção Baseados em Rede. Adaptado de (BEALE, 2004)

Tipicamente possui dois componentes principais, os mecanismos de captura permanente (promíscuos), responsáveis por recolher informações em pontos selecionados da rede, e o mecanismo de detecção, responsável por analisar os pacotes e enviar alarmes quando necessário (WANG, 2009).

Mesmo em modo promíscuo, um SDI pode não capturar todos os pacotes de uma rede. Para garantir que isso ocorra, o dispositivo de rede superior deve ser configurado para enviar todos os pacotes a máquina que possui a instalação do SDI. Em caso de um *hub*, por exemplo, isso não é necessário, mas no caso de um *switch* ou similar, as opções da porta referente ao SDI devem ser modificadas para trabalhar em modo de monitoramento para receber todos os pacotes da rede e não somente os referentes a sua interface (BEALE, 2004).

Há dois tipos desse SDI: os baseados em um nó da rede, que monitoram os pacotes que entram e saem de um computador específico, e o baseado em um sensor da rede, que é posicionado em uma localização específica, checando os pacotes que passam pelo local (WANG, 2009). Um SDI de rede possui a vantagem de não atrapalhar as demais máquina da rede que ele está monitorando, além de poder passar desapercebido por um invasor que esteja na rede e de ter seus serviços desabilitados para evitar a possibilidade de um ataque (BEALE, 2004).

3.1.2 SDI baseado em *Host*

Os Sistemas de Detecção Baseados em *Host* (Figura 3.1.2) protegem somente o sistema que residem, sendo capazes de identificar comportamentos estranhos através da análise dos *logs* do sistema, que possuem os comportamentos registrados, ou através da análise dos pacotes do *Host*, com a interface no modo não promíscuo. Também podem criar registros com informações ou podem verificar as configurações do sistema para identificar comportamentos de invasão (BEALE, 2004; WANG, 2009). Ele pode ser vantajoso quando a placa não aceita modo promíscuo ou quando a máquina não tem um processamento capaz de suportar o modo promíscuo (BEALE, 2004).

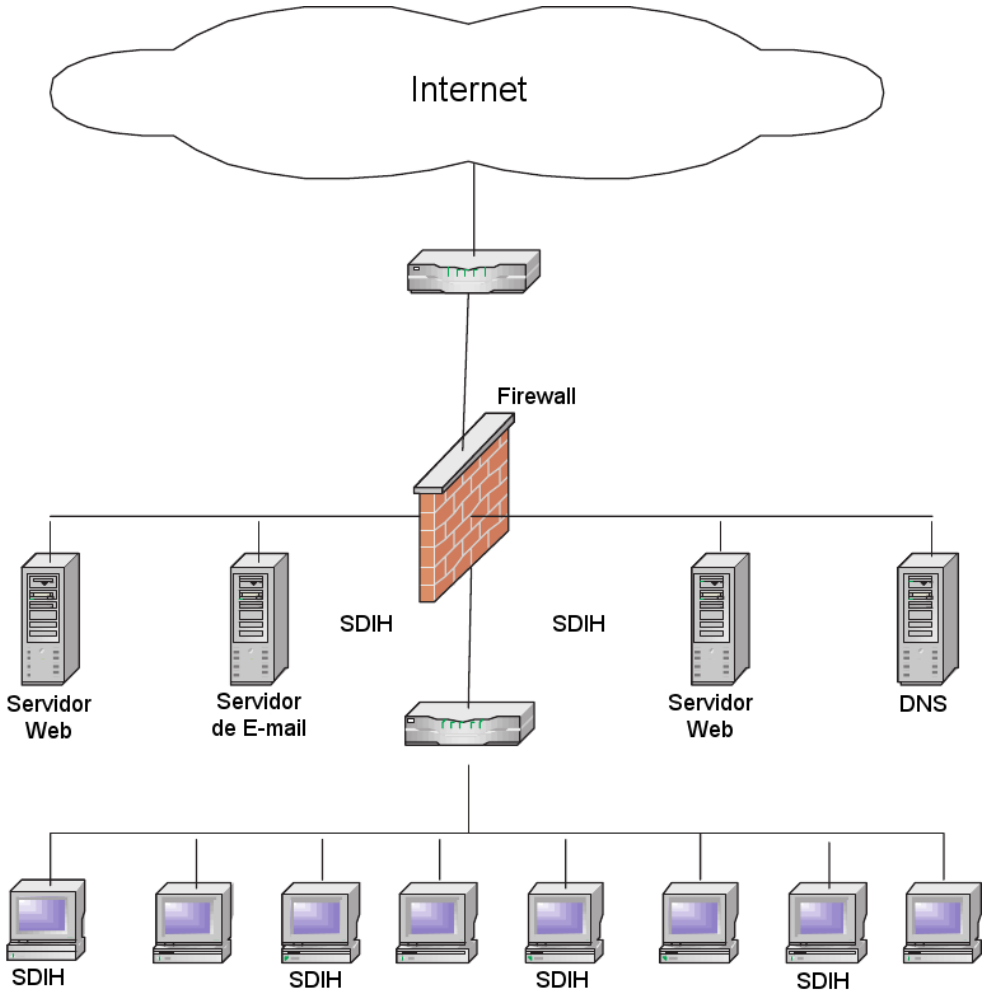


Figura 3.1.2: Sistemas de Detecção Baseado em Host. Adaptado de (BEALE, 2004)

Outra vantagem é de possuir uma PDI específica para o computador no qual está instalado e poder monitorar modificações específicas nos arquivos e no sistema operacional, modificações que poderiam passar desapercebidas. Como desvantagens, a escolha de um SDI baseado em *Host* é mais complicada, pois deve se adequar ao sistema instalado, pois consume mais recursos da máquina em sua execução e, caso instalado em vários *hosts*, trás dificuldade de atualização das normas da PDI (BEALE, 2004).

3.1.3 SDI Distribuídos

Um Sistema de Detecção Distribuído (Figura 3.1.3) é a combinação de sensores de Sistemas Baseados em Rede ou de sensores de Sistemas Baseados em *Hosts*, ou de ambos

distribuídos na empresa, todos ligados a um sistema central. As informações de ataque são então enviadas continuamente ou periodicamente para um servidor central, onde podem ser armazenados em um banco de dados central. Os sensores podem também obter novas políticas disponibilizadas no servidor central. Diferentes tipos de sensores podem ter diferentes servidores e suas políticas podem ser adaptadas para cada *host* (BEALE, 2004).

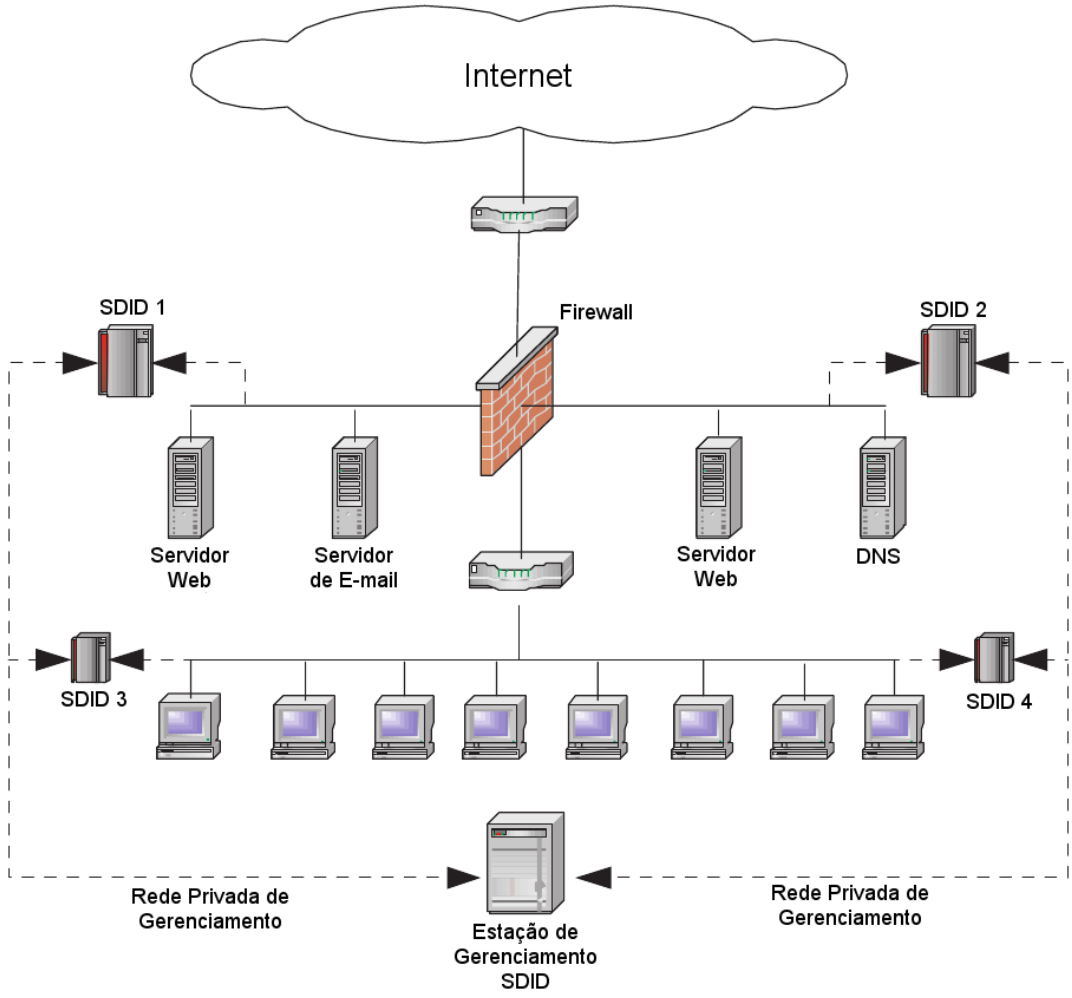


Figura 3.1.3: Sistemas de Detecção Distribuídos. Adaptado de (BEALE, 2004)

3.2 Métodos de Detecção

Os modelos de detecção de invasão mais comuns são os baseados em anomalias e os baseados em abuso, também conhecidos como baseados em assinaturas devido ao seu método de detecção ser realizado sobre assinaturas, ou seja, sobre trechos de caracteres

pertencentes a uma transação de ataque. Ainda tem-se os sistemas que utilizam características desses dois métodos para melhorar sua taxa de acerto.

Tem-se a seguir a descrição dos métodos mencionados. O tópico 3.2.1 descreve os modelos baseados em assinatura, o tópico 3.2.2 descreve os modelos baseados em anomalia e o tópico 3.2.3 descreve os sistemas híbridos.

3.2.1 Métodos de Detecção baseados em Assinatura (Abuso)

Os sistemas baseados em assinaturas (Figura 3.2.1) costumam ser os preferidos por necessitarem de pouco custo computacional e por causarem pequeno comprometimento de desempenho (CANSIAN, 1997). Utilizam técnicas de casamento de padrões, combinando os dados analisados, da rede ou dos registros do sistema, com sua base de assinaturas de ataques conhecidos para decidir se são aceitáveis. Ao conseguir sucesso na combinação, este SDI emite um alarme ao administrador responsável (PIETRO, 2008; WANG, 2009).

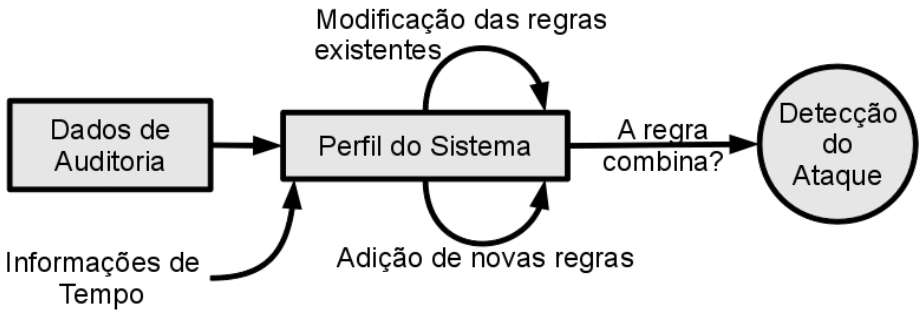


Figura 3.2.1: Métodos de Detecção Baseados em Assinatura. Adaptado de (SUNDARAM, 1996)

Segundo (CANSIAN, 1997), os sistemas baseados em assinatura podem ser:

- sistemas especialistas: capazes de analisar através de regras "se-então", reconhecendo ataques e formulando uma solução ao problema;
- por modelamento: compara informações de registros de auditoria com modelos de comportamentos de ataque para gerar uma suspeita ou para descartá-la.

Os diferentes tipos de assinaturas utilizados nesses sistemas podem ser (WANG, 2009):

- assinaturas de redes: que proveem informações dos pacotes que podem afetar a execução normal do sistema. Consistem em assinaturas de cabeçalhos ou de campo de dados (*payload*). Também são chamadas de assinaturas de conteúdo (*content signatures*). As assinaturas de *payload* verificam ações do usuário, enquanto assinaturas de cabeçalhos verificam pacotes maliciosos que são capazes de identificar, como, por exemplo, ataques de difusão (*broadcast*);
- e assinaturas de *hosts*: que utilizam informações de comportamentos que podem afetar na execução normal do sistema. Um exemplo seria três tentativas seguidas de senha errada de determinado usuário. Essas assinaturas podem ser:
  - de evento-único: um único comando que é um comportamento suspeito;
  - de multi-eventos: formada por grupos de várias assinaturas de evento-único;
  - de multi-*hosts*: formada por sequências de assinaturas de evento-único originados de diferentes *hosts*; ou
  - compostos: une assinaturas de rede com assinaturas de hosts para identificar certos tipos de ataques que não podem ser identificados somente por um tipo dessas assinaturas. Um usuário pode, por exemplo, modificar arquivos utilizando um computador remoto, mas somente unindo essa informação à informação da localização do computador é que pode-se tomar uma decisão melhor do ataque.

Os SDIs deste modelo tem três métodos para modelar assinaturas (WANG, 2009):

- Sistemas Compilados (*Built-in System*): armazena um conjunto de regras de detecção dentro do sistema e provê um editor ao usuário, permitindo-os

selecionar regras baseando-se em suas necessidades;

- Sistemas Programados (*Programming System)*: provê um conjunto de regras e uma linguagem de programação (ou linguagem *script*), permitindo aos usuários selecionar as regras padrões e/ou escrever suas próprias regras. Este modelo provê mais flexibilidade, mas também necessita que o usuário aprenda a utilizar sua linguagem de programação;
- Sistemas Especialistas (*Expert System*): são SDIs direcionados a atender as necessidades específicas de uma organização em especial. Requer profissionais experientes na área para definir as assinaturas de detecção.

3.2.2 Métodos de Detecção baseados em Anomalia

Os sistemas baseados em anomalias (Figura 3.2.2) baseiam-se na supervisão de comportamentos anômalos do sistema, assumindo que as atividades anormais podem ser invasões. Essa técnica constrói primeiro um modelo, geralmente estatístico, capaz de descrever as atividades normais dos usuários, ou dos *hosts*, ou o tráfego normal da rede, então marca qualquer comportamento que significamente desvia de um modelo como um ataque. Esses sistemas tem a vantagem de detectar novos ataques assim que ocorrem, porém eles requerem uma fase de treinamento e uma configuração cuidadosa do seu limiar de detecção, o que requer um desenvolvimento mais complexo. Os resultados mais satisfatórios atualmente são encontrados em sistemas baseados na análise do conteúdo do pacote (*payload)* (CANSIAN, 1997; PIETRO, 2008; SUNDARAM, 1996).

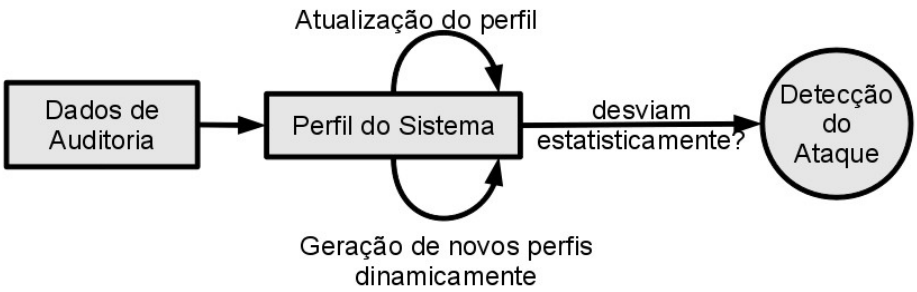


Figura 3.2.2: Métodos de Detecção Baseados em Anomalia. Adaptado de (SUNDARAM, 1996)

Esse tipo de SDI apresenta erros devido à identificação de atividades normais de um usuário como atividades anormais, ou quando deixa de identificar atividades anormais como invasão por parecerem muito com a atividade costumeira de um usuário (Figura 3.2.3). Podem-se encontrar os comportamentos (CANSIAN, 1997; KUMAR, 1995):

- intrusivo, mas não anômalo: também chamados de falso negativos. Neste caso, ocorre uma falha na detecção, pois a invasão não provoca atividade anormal, não sendo então detectada pelo sistema;
- não intrusivo, mas anômalo: também chamados de falso positivos. Neste caso, ocorre uma falha na detecção, que indica erroneamente a anormalidade como uma invasão;
- não intrusivo e não anômalo: também chamados de verdadeiros negativos. Neste caso, ocorre um acerto, pois a atividade não é anômala e não é evidenciada como invasão;
- intrusivo e anômalo: também chamados de verdadeiros positivos. Neste caso, ocorre um acerto, pois a atividade é anômala e é evidenciada como invasão.

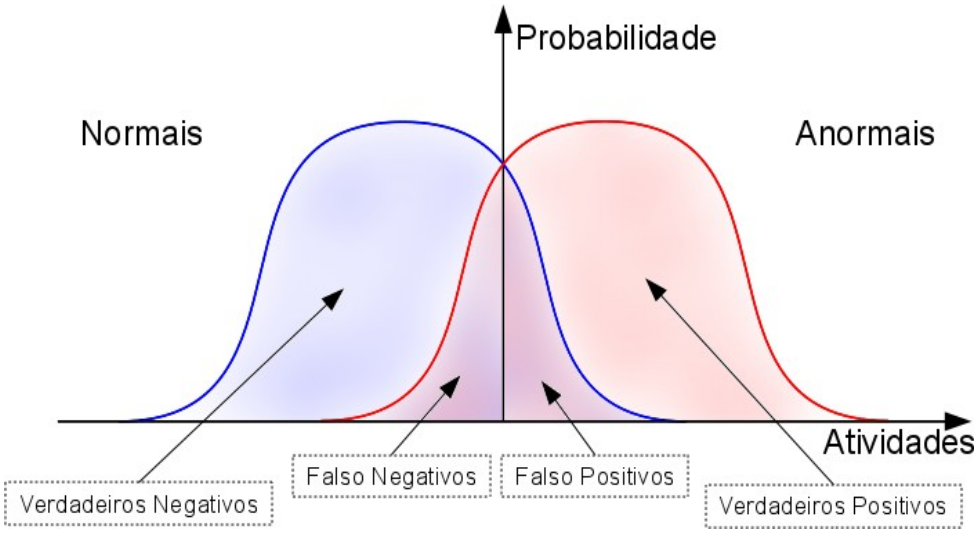


Figura 3.2.3: Probabilidade de Detecção entre Atividades Normais e Anormais

Um sistema baseado em anomalia é classificado quanto ao algoritmo que utiliza, quanto a utilização das características de cada pacote singularmente ou de toda a conexão e quanto aos tipos de dados analisados, dos cabeçalhos do pacote ou do *payload* (PIETRO, 2008).

Das diferentes abordagens de algoritmos utilizados, somente duas tem sucesso nas aplicações da última década: modelos estatísticos e baseados em redes neurais (PIETRO, 2008). Mais de 50% dos modelos existentes são baseados em estatística, criando primeiramente um modelo estatístico (livre de ataques) do comportamento da rede; depois comparando os dados de entrada ao modelo utilizando uma função de distância e indicando uma atividade anômala, um ataque, ao medir uma distância excedente a um limiar definido.

Os sistemas baseados em anomalia que utilizam redes neurais trabalham de forma similar, mas ao invés de construir um modelo estatístico, elas treinam uma rede neural para reconhecer o tráfego de uma anomalia. A fase de treinamento requer que os dados sejam os mais livres quanto possível de ataques, ou seja, que os dados representem um fluxo normal da rede sem incluir atividades maliciosas. Além disso, a fase de treinamento deve ser suficiente para permitir que o sistema crie um modelo fiel (PIETRO, 2008).

Quanto à utilização das características dos pacotes, ocorre a escolha entre a orientação à pacotes, quando utiliza-se cada pacote individualmente, ou orientação à conexões, quando utiliza-se todos os pacotes pertencentes a uma comunicação específica entre duas máquinas, como, por exemplo, uma comunicação *http* de uma página *web*. Um sistema orientado à pacotes utiliza somente um simples pacote como uma fonte mínima de informação, enquanto os sistemas orientados a conexões consideram características de toda a comunicação antes de indicar se o tráfego é anômalo ou não. Os sistemas orientados a conexão poderiam até utilizar o campo de dados (*payload*) dos pacotes, porém o tempo utilizado para essa tarefa seria grande, o que diminuiria o desempenho do sistema. Na prática, tipicamente obtém-se o número de *bytes* enviados e recebidos, a duração da conexão e o protocolo da quarta camada do modelo de referência TCP/IP que foi utilizado (PIETRO, 2008).

Quanto aos tipos de dados utilizados, tem-se os sistemas que utilizam o cabeçalho

e os que utilizam o *payload* do pacote. O primeiro considera somente os cabeçalhos dos pacotes, utilizando os cabeçalhos da terceira camada e, se existir, da quarta camada do modelo de transferência TCP/IP. Já os baseados no *payload* analisam os dados somente da quarta camada do modelo de transferência TCP/IP. Existem também sistemas híbridos, que misturam informações coletadas dos cabeçalhos de pacotes e, se houver, dos dados do *payload* da quarta camada do TCP/IP (PIETRO, 2008).

Quanto ao tipo de detecção realizado, encontram-se os sistemas estatísticos, geradores de prognóstico de padrões e de classificação *bayesiana*.

3.2.2.1 Estatísticos

A análise estatística pode ser utilizada quando a diferença entre os eventos que podem ser aceitos e eventos que não podem ser aceitos pode ser quantificada. Uma das formas de analisar utiliza valores limiares, contando o número de ocorrências de certos tipos de eventos durante um período de tempo. Considera-se um comportamento de invasão quando uma conta excede um limiar predeterminado. É um método simples de implementar, porém a utilização de limiares pode não trazer corretamente a divisão entre as ações anômalas e as ações normais do sistema, pois nem sempre elas podem ser linearmente separáveis (WANG, 2009).

Outra forma de análise é através de perfis de usuários, que utiliza os eventos passados de um usuário ou um grupo de usuários para criar perfis baseados em certas medidas quantificadas. Como exemplos de quantificação temos o tempo que um evento particular ocorre, o número de vezes que determinado evento ocorre em um período de tempo, os valores das variáveis de sistema, etc. Após a criação de um perfil considerado normal, o sistema, rotineiramente, cria um novo perfil com os eventos daquela faixa de tempo e o compara ao perfil normal, se o desvio for maior que determinado limiar, encontram-se sinais de anormalidade, que podem significar uma invasão.

Encontram-se também sistemas que atualizam seu perfil, porém isso pode levar a erros, pois os invasores podem levar o sistema a aceitar aos poucos, através da adaptação do sistema, suas atividades de invasão. Encontrar o limiar correto para o sistema também é uma dificuldade desse sistema, podendo causar vários Falso Positivos e Falso Negativos

(CANSIAN, 1997; COSTA, 2007; SUNDARAM, 1996; WANG, 2009).

3.2.2.2 Gerador de prognóstico de padrões

Essa técnica considera que os eventos não ocorrem de forma aleatória, tentando então prever os próximos eventos baseando-se nos eventos que já ocorreram. Um exemplo seria: se ocorreu o evento E1 e depois o E2, considera-se que o evento E3 tenha 70% de chances de ocorrer, enquanto E4 tenha 90% de chances de ocorrer e E5 tenha 5% de chances de ocorrer (CANSIAN, 1997; COSTA, 2007; SUNDARAM, 1996).

Essa técnica possui uma fase de aprendizado que gera os padrões dinamicamente, baseando-se no comportamento do usuário. Ao predizer resultados corretos várias vezes, a regra ganha confiabilidade na precisão de prognóstico e ao ser aplicada com sucesso em testes monitorados, a regra ganha grau de confiança. Somente as melhores regras, com mais precisão de diagnóstico e de maior confiança, são mantidas para serem utilizadas no sistema. A detecção de eventos anômalos é mais rápida por analisar pequenos conjuntos de regras, porém o sistema só pode detectar os padrões já treinados (CANSIAN, 1997; COSTA, 2007; KUMAR, 1995).

3.2.2.3 Classificação Bayesiana

É uma técnica não supervisionada que consegue trabalhar com informações incertas, baseando-se na teoria da probabilidade. Seu modelo trata as ações finais de determinados comportamentos, procurando então determinar quais as ações mais prováveis de tê-las gerado. Para isso, utiliza a probabilidade de cada ação ter ocorrido no conjunto de dados analisado (CANSIAN, 1997; KUMAR, 1995; MAIA, 2005).

Uma rede *bayesiana* representa todas as variáveis analisadas como nós de um grafo, distribuindo-os por sua probabilidade. Assim, um nó X pode ter os filhos que indicam a ação de origem. Com tais características, uma rede tradicional com estrutura genérica é um problema NP-Completo, necessitando-se de utilizar alternativas que utilizem esse conceito de forma mais eficiente, como o Classificador Naïve Bayes (MAIA, 2005).

3.2.3 Híbridos

São sistemas que possuem técnicas de anomalia e de assinaturas combinadas para reduzir e/ou eliminar as falhas de uma das técnicas utilizadas, melhorando assim o desempenho geral do sistema implementado. O problema encontrado nesses sistemas é a sua complexidade de implementação e sua limitação para uma implantação prática devido quantidade de considerações que o sistema deve analisar (CANSIAN, 1997).

Outra forma de detecção de Intrusão é através da técnica de Perícia de dados Comportamentais (*Behavioral Data Forensics*), que estudam em como utilizar as técnicas de mineração de dados para analisar *logs* de eventos e procurar por informações úteis. Qualquer informação que indique o passado, o presente ou o futuro de atividades de invasão são interessantes a este processo (WANG, 2009).

O objetivo da perícia descrita é encontrar informações úteis em grande quantidade de dados. Após os SDIs coletarem os dados, necessita-se de identificar o comportamento dos intrusos e os dados podem vir em diferentes formatos e em grande quantidade. As técnicas de mineração de dados são:

- refinamento dos dados: melhorar a representação de dados para ajudar a encontrar novas informações;
- interpretação contextual: interpretar os dados de acordo com seu contexto para ajudar a encontrar novas informações;
- combinação da origem: combinar diferentes tipos de fontes de dados para encontrar novas informações à partir de novas perspectivas;
- *Out-of-Band Data*: combinar dados de fora do escopo da intrusão para detectar novas informações;
- *Drill Down*: inicia de um nível mais alto das atividades. Uma vez que encontra um comportamento suspeito, procura por níveis mais baixos de atividades para encontrar mais informações.

Agora que foram expostas as informações necessárias sobre os SDIs, o próximo capítulo apresentará os estudos sobre à aplicação de técnicas de IA em SDIs.

Capítulo 4

Sistemas de Detecção de Intrusão com métodos de Inteligência Artificial

As técnicas atuais de prevenção de intrusão tem falhado em proteger as redes de computadores e seus sistemas agregados contra ataques mais sofisticados e contra *malwares*[1] (WU, 2010). Além de apresentarem outros problemas, como sobrecarga na manutenção das assinaturas da base de dados e sobrecarga na verificação de cada um dos pacotes da rede com todas as assinaturas de ataques existentes (BRITT, 2007). A utilização de técnicas de Inteligência Artificial (IA) em SDIs apresenta-se como medida para suprir as necessidades dos métodos utilizados atualmente, despertando grande interesse de pesquisa devido as suas características de adaptação, de tolerância a falhas, de alta velocidade computacional e de resiliência a erros (WU, 2010). Para melhorar os métodos tradicionais de detecção de intrusão, são propostos vários métodos de Inteligência Artificial, como Inteligência Coletiva (Colônia de Formigas e Enxame de Partículas), Lógica Nebulosa (*Fuzzy*), Métodos Computacionais Evolutivos (Algoritmos Genéticos e Programação Genética), Redes Neurais e Sistemas Imunológicos Artificiais (BRITT, 2007; WU, 2010).

Porém, mesmo com diversas escolhas de métodos em IA, as pesquisas atuais demonstram que a aplicação de um único método apresenta vantagens e desvantagens em sua utilização prática. Para compensar as desvantagens e para aproveitar as vantagens que cada método provê, torna-se ideal utilizar os Sistemas Inteligentes Híbridos, que

1 software que pode se infiltrar em um sistema para causar danos ou para roubar de informações diversas.

consistem em utilizar dois ou mais métodos para realizar determinada tarefa (WU, 2010). Nas subseções seguintes os métodos de IA citados são descritos.

4.1 Inteligência Coletiva

A Inteligência Coletiva baseia-se na propriedade de animais possuírem comportamentos inteligentes ao apresentar habilidade em resolver problemas coletivamente. Seus modelos são baseados em populações, visando a capacidade de seus indivíduos serem capazes de procurar por uma solução satisfatória através de passos de iteração, mudando seu pensamento sobre o espaço de pesquisa por comunicação direta e/ou indireta (WU, 2010).

A Inteligência Coletiva na natureza possui as características (SIERAKOWSKI, 2006):

- Avaliação: princípio básico relacionado a capacidade de analisar o ambiente, determinando o que é positivo ou negativo, atrativo ou repulsivo. O aprendizado depende dessa avaliação;
- Comparação: princípio básico relacionado a comparação dos seres com outros da mesma espécie para avaliar a si mesmo, o que pode gerar em motivação para aprender e evoluir;
- Imitação: princípio básico relacionado a forma efetiva de aprender a fazer ações. Porém, poucos animais tem essa capacidade;
- Estratégia de busca de alimento: relacionada aos métodos de localização, manipulação e ingestão de alimentos. Muitos animais, na busca por alimento, procuram formar colônias que cooperam para atingir objetivos comuns, mas também podem repudiar alguns membros devido a competições internas. Eles procuram maximizar a energia obtida do alimento pelo tempo de sua procura. Os animais com estratégias pobres são eliminados pela natureza e os animais com mais energia se procriam e se propagam, ampliando assim seu ambiente.

Os estudos e aplicações da Inteligência Coletiva procuram resolver problemas complexos dividindo-os em múltiplos agentes simples que não tem controle centralizado ou modelo global. Estima-se que as interações locais entre os agentes e o seu ambiente desenvolvam um padrão global de comportamento. A estratégia emergente e o controle altamente distribuído são as características mais importantes da Inteligência Coletiva, o que pode produzir um sistema autônomo, adaptativo, flexível, paralelo e eficiente (WU, 2010).

Os métodos mais populares de Inteligência Coletiva são: Colônia de Formigas, que simulam o comportamento de formigas e tem resolvido problemas discretos de otimização, e Enxame de Partículas, que simulam um sistema social simplificado de um bando de pássaros ou de um cardume de peixes, sendo mais adequado para resolver problemas de otimização não lineares.

4.1.1 Colônia de Formigas

Uma formiga sozinha não é um animal inteligente, mas as colônias de formigas podem fazer tarefas difíceis que uma única formiga não pode fazer. Demonstram um comportamento emergente ao procurar por comida e ao ter comportamento de classificação dos alimentos capturados. Conseguem encontrar coletivamente um local próximo com boa fonte de comida através de seus feromônios, que marcam suas rotas quando se movem. A concentração de feromônios indica o caminho mais utilizado e encorajam mais formigas a seguir. A formiga que encontrar o alimento primeiro pelo menor caminho, certamente voltará pelo mesmo caminho antes que as outras, aumentando assim sua quantidade de feromônio e fortalecendo-o para ser seguido. O feromônio também tem a característica de evaporar com o tempo, o que desmarca caminhos longos ou pouco percorridos, forçando a melhoria de caminhos mais curtos (SIERAKOWSKI , 2006 ; WU, 2010).

Em SDIs, já foi proposta a utilização de colônias de formigas para descobrir regras de classificação, para acompanhar os passos dos intrusos e para identificar as características dos ataques através dos caminhos de feromônios criados. Outras pesquisas são em agrupar objetos para fazer a detecção de intrusão pelos grupos não identificados,

porém em aplicações reais com a base de dados grandes, inicialmente os algoritmos apresentaram problemas de sensibilidade na separação dos grupos, somente após várias melhorias que conseguiram apresentar resultados muito bons (WU, 2010).

4.1.2 Enxame de Partículas

Esta técnica é uma otimização baseada no comportamento social de animais, baseando-se na manipulação das distâncias entre os indivíduos, em seu comportamento para manter uma distância ótima entre os membros do grupo. Indivíduos da população são considerados partículas que se movem em um espaço de pesquisa multidimensional em direção a melhor solução (otimização global), ajustando sua posição e velocidade de acordo com sua própria experiência (busca local) ou com a experiência de seus vizinhos (busca global) (SIERAKOWSKI , 2006; WU, 2010).

Em sua utilização com SDIs, o método de enxame de partículas foi aplicado para melhorar o aprendizado de regras de classificação de um sistema específico. Outra utilização foi em um algoritmo genético com regras nebulosas, o método foi aplicado para melhorar a qualidade das regras nebulosas através de pesquisa de vizinhança, para assim gerar melhores mutações e cruzamentos nos algoritmos genéticos (WU, 2010).

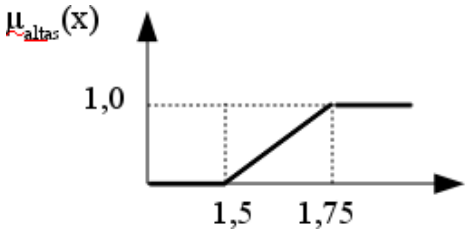
4.2 Lógica Nebulosa

Problemas complexos necessitam de mais valores lógicos além do verdadeiro e falso, devido a isso a Lógica Nebulosa (*Fuzzy Logic*) foi desenvolvida. Sendo iniciada em 1964, foi criada para tratar graus de pertinência intermediários entre a pertinência total e a não pertinência de elementos de determinado conjunto. Trabalha com valores contínuos [0,1], onde sua variação indica a completa falsidade ou a verdade absoluta do evento, assim, por exemplo, o elemento 0,7 pertenceria ao conjunto com 70% de confiança (ARTERO, 2009; REZENDE, 2005).

Tanto pessoas, quanto análises probabilísticas e outros métodos trabalham com valores de indecisão. A estatura, por exemplo, pode ser alta, média ou baixa, com valores indefinidos para cada uma das medidas, pois algumas pessoas podem ou não achar alguém de 1,70m alto. Para trabalhar com o sistema nebuloso necessita-se transformar as

informações quantitativas em informações qualitativas, processo chamado de fuzzificação, para assim criar um conjunto nebuloso que consiga representar corretamente as informações. O conjunto nebuloso formado possui os números reais mapeados no intervalo de [0,1] através de uma função de pertinência.

Geralmente as equações para fuzzificar as informações são definidas por especialistas ou por conjuntos nebulosos já definidos, como o triangular e o trapezoidal. Para trabalhar com esses valores nebulosos, deve-se utilizar os conjuntos da Variável Linguística referente. Para melhor apresentação do conceito, tem-se o seguinte exemplo: suponha a curva e a função de fuzzificação da Figura 4.2.1 para o conjunto de pessoas Altas da Variável Linguística Estatura. Uma altura de 1,70m teria um valor nebuloso de 80% ($\mu_{altas}(1,70)=0.8$) de confiança, enquanto a altura de 1,60m teria 40% ($\mu_{altas}(1,60)=0.4$) de confiança. A variável linguística Estatura também pode possui outros conjuntos nebulosos, como conjunto de pessoas Médias ($\mu_{médias}$) e de pessoas Baixas (μ_{baixas}), para então possuir os graus de pertinência de cada valor real no conjunto especificado.



$$\mu_{altas}(x)=\begin{cases}0 & , se\ x<1.50\\ \dfrac{x-1.50}{1.75-1.50} & , se\ 1.50\le x\le 1.75\\ 1 & , se\ x>1.75\end{cases}$$

Figura 4.2.1: Curva de Fuzzificação e sua equação para estaturas altas (ARTERO, 2009).

Com os conjuntos nebulosos definidos, deve-se aplicar a inferência no sistema através de regras de produção, como E, OU e NOT:

- *SE* estatura *é alta E* estatura *é média ENTÃO* pessoa *é quase alta*; ou seja:

$$Pessoa_{quase\_alta}\ (valor\ \ da\ \ altura) =$$
$$= E\ (\mu_{altas}(valor\ da\ altura)\ ,\ \mu_{médias}(valor\ da\ altura))$$

Para que o resultado das regras produzido possa ser utilizado, deve-se transformar o valor qualitativo novamente em um valor quantitativo através de um processo chamado defuzzificação. Para isso, vários métodos podem ser utilizados, como o método do

Critério Máximo, o método do Centro Geométrico (COG) e o método da Média do Máximo (MOM) (ARTERO, 2009).

Sua utilização em detecção de intrusão é interessante devido aos erros causados pela utilização direta dos valores numéricos coletados dos dados de auditoria e pelo processo de segurança ter uma barreira mal definida entre o normal e o anormal. Para efetuar detecções por abuso, a lógica nebulosa foi inicialmente utilizada em sistemas especialistas, tendo suas regras criadas por especialistas de segurança com seu domínio de conhecimento. O maior problema desses sistemas foi sua limitação devido as regras feitas a mão. Com a evolução dos sistemas, iniciou-se a utilização de métodos com lógica nebulosa em pesquisas com redes neurais artificiais, com computação evolucionária e sistemas imunológicos artificiais. Outro tipo de aplicação das regras nebulosas é chamada de fusão de decisão, que funde as saídas de diferentes modelos para representar uma decisão nebulosa final (WU, 2010).

Para a detecção de anomalias, as regras nebulosas foram utilizadas para extrair padrões normais de dados de auditoria que depois seriam comparados com novos dados para detectar possíveis ataques. Outras pesquisas trabalharam também com Mineração de Dados, no método de Detecção de *Outlier*, onde *outlier* é um fato que desvia muito de outros fatos a ponto de gerar suspeitas que possam ser consideradas como ataques (WU, 2010).

4.3 Métodos Computacionais Evolutivos

A Computação Evolutiva (CE) teve início em meados de 1950 e foi inspirada na Teoria da Evolução Natural e na Genética, sendo mais conhecida por suas subáreas, como Algoritmos Genéticos e Programação Genética, descritos nas seções seguintes. A CE tem vários papeis nos SDI, sendo (WU, 2010):

- otimização: algumas pesquisas tentaram diagnosticar falhas, assim como consultas médicas, criando inicialmente uma matriz com eventos de ataques para servir como base de conhecimento. As falhas indicavam genes binários em um método de Algoritmos Genéticos, que processavam e descartavam as falhas, ou seja, os genes ativos que não eram ataques;

- modelagem automática das estruturas de alguns algoritmos: algumas pesquisas trabalharam com Algoritmos Genéticos para selecionar um conjunto de características ideais para serem utilizadas, para designar novas estruturas em redes do tipo NNTree, que apresentaram um tempo de treinamento menor e para escolher a quantidade de *clusters* em Algoritmos de Cluster;
- classificadores: algumas pesquisas trabalharam com Algoritmos Genéticos e Programação Genética para escolher melhores Regras de Classificação, que são compostas de cláusulas *if-then*, e Funções de Transformação, que diminuem o espaço dimensional dos dados para, por exemplo, 1D ou 2D, onde uma simples linha pode separar dados em diferentes classes.

Nos itens relatados, os métodos de CE melhoraram alguns tipos de detecção, porém demandaram muito tempo de processamento.

4.3.1 Algoritmos Genéticos

Os Algoritmos Genéticos são uma classe de procedimentos que seguem os passos inspirados na teoria do processo biológico de seleção natural de Darwin, seguindo a ideia de sobrevivência do mais forte. Seu processo faz pequenas alterações em gerações de cromossomos para tentar identificar uma solução ótima.

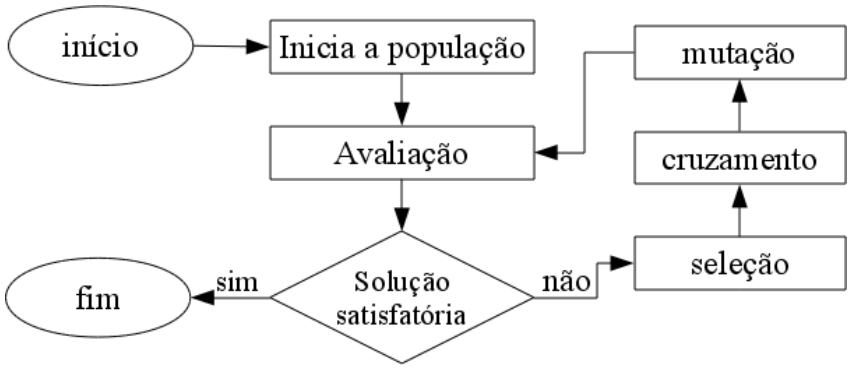


Figura 4.3.1: Algoritmo Genético. Adaptado de (ARTERO, 2009)

No processo (Figura 4.3.1), inicialmente forma-se uma *população* com um conjunto aleatório de *indivíduos* que formam possíveis soluções de um problema. Logo

após, essa população inicia um processo evolutivo, onde cada indivíduo é *avaliado* sobre sua adaptação ao ambiente. Uma porcentagem dos membros mais adaptados é *selecionada* e mantida e o restante é descartado. Dentre os indivíduos mantidos, pode ocorrer a *mutação* e o *cruzamento* (recombinação genética), gerando mais membros. Todo esse processo, chamado *reprodução*, é repetido até encontrar uma solução que seja satisfatória (COPPIN, 2010; REZENDE, 2005).

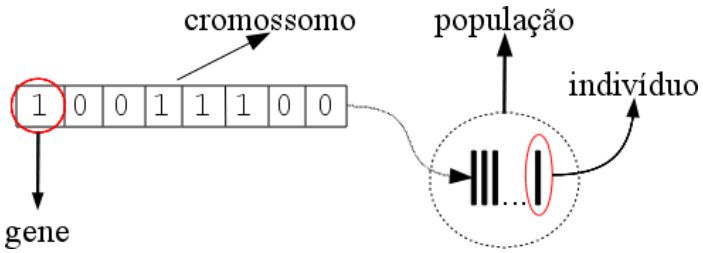


Figura 4.3.2: Geração de uma população

Na implementação de Algoritmos Genéticos, inicialmente escolhe-se como serão formados os *genes*, podendo ser valores reais, inteiros, ou os mais utilizados, binários. Logo após, formam-se os *cromossomos* com as combinações dos genes. Tem-se então o *indivíduo*, que corresponde a um cromossomo utilizado para representar uma solução para o problema em questão. Forma-se a *população* (Figura 4.3.2) com um conjunto de indivíduos que irão competir para serem mantidos e que participarão da reprodução para melhorar sua solução para o problema. A avaliação é realizada através de uma *função de aptidão*, que mede a habilidade do indivíduo para mantê-lo e para reproduzi-lo. A cada iteração de uma reprodução, tem-se uma *geração*. Para fins de melhor entendimento, um cruzamento foi ilustrado na Figura 4.3.3, onde os cromossomos binários tem seus três últimos genes cruzados (ARTERO, 2009; REZENDE, 2005).

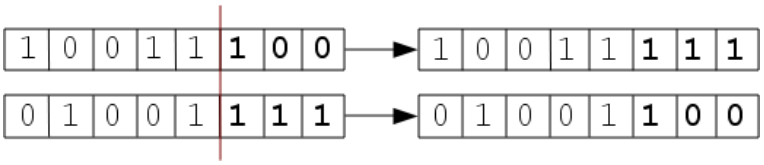


Figura 4.3.3: Cruzamento de Cromossomos

4.3.2 Programação Genética

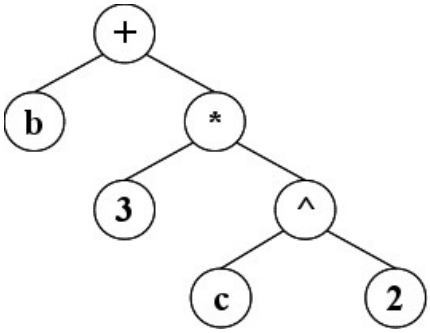


*Figura 4.3.4: Árvore representando um programa para calcular b + 3 * c^2 (REZENDE, 2005).*

A programação genética surgiu em meados de 1980 e procura criar automaticamente programas a partir de uma descrição de alto nível de um problema. Assim como os Algoritmos Genéticos, este método cria uma *população* inicial, formada de *indivíduos*, que desta vez são *programas* de computador formados por *árvores* (Figura 4.3.4) de tamanhos e formas diferentes, ao invés de cromossomos formados por combinações de números. A *avaliação* de cada indivíduo ocorre pela execução do programa. O *aprendizado*, ilustrado na Figura 4.3.5, ocorre pela *mutação* dos nós da árvore, que podem ser funções aritméticas, funções lógicas, operadores condicionais, atribuições e funções específicas, ou pela mutação de uma subárvore (REZENDE, 2005).

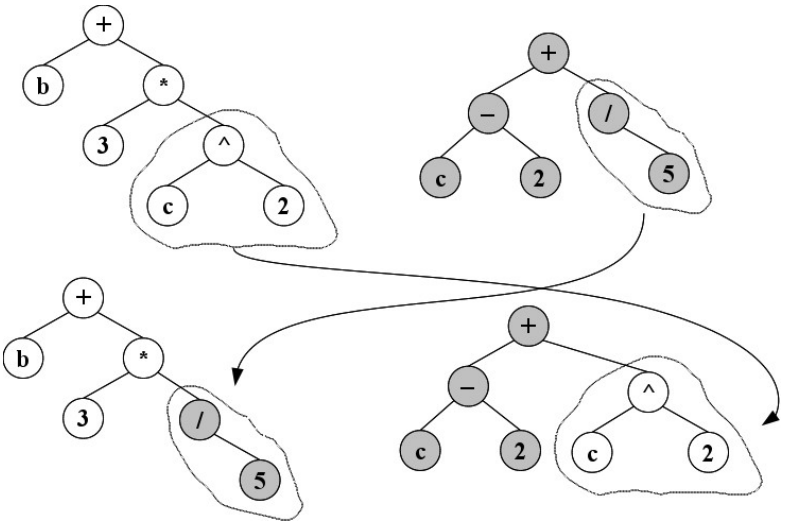


Figura 4.3.5: Exemplo de cruzamento. Adaptado de (REZENDE, 2005).

4.4 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNA) são baseadas nas estruturas e funcionamento do cérebro humano e possuem capacidade de aprendizado e generalização. Iniciaram-se com Warren McCulloch e Walter Pitts, em meados de 1943, que desenvolveram o primeiro modelo matemático de um neurônio (BITTENCOURT, 2006; REZENDE, 2005). As RNAs são capazes de resolver problemas de aproximação, predição, classificação, categorização e otimização (REZENDE, 2005), mas sua maior utilização é no reconhecimento de padrões, em aplicações de reconhecimento de caracteres, de voz, de imagens, dentre outros (ARTERO, 2009; REZENDE, 2005).

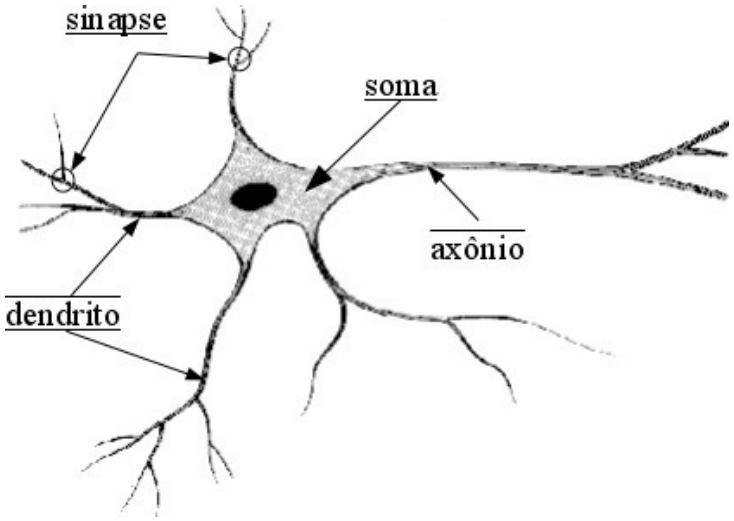


Figura 4.4.1: Representação de um Neurônio Biológico

O cérebro possui mais que dez bilhões de neurônios interconectados. Cada neurônio (Figura 4.4.1) recebe os sinais de outros neurônios através dos dendritos e produz um pulso elétrico de saída se o sinal de entrada exceder um limiar. Um axônio leva esse sinal, conectando-se ao dendrito de outro neurônio através de um espaço chamado sinapse. O cérebro consegue modificar a natureza e o número de conexões dos neurônios, atribuindo pesos para reforçar as conexões que levem a resposta correta e para enfraquecer as conexões que levem a respostas incorretas (COPPIN, 2010).

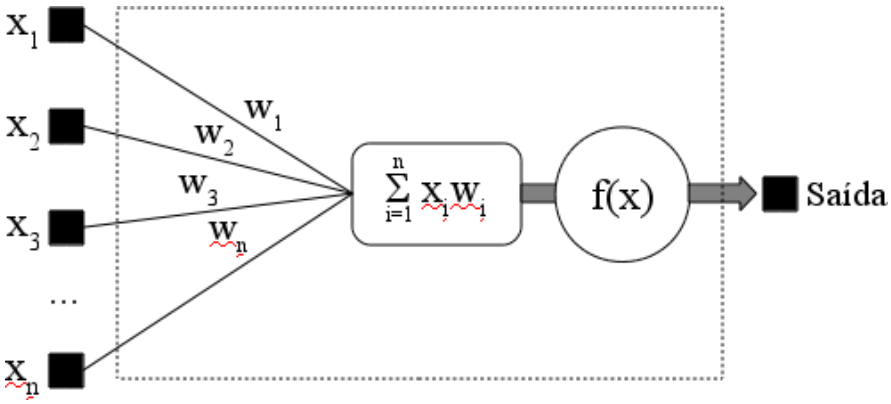


Figura 4.4.2: Neurônio Artificial

Um neurônio artificial, representado na Figura 4.4.2, é composto por diversos valores de entradas (x_1, … ,x_n), sendo cada valor assimilado a um peso (w_1,...,w_n) positivo, que pode simular uma sinapse excitadora, ou a um peso negativo, que pode simular uma sinapse inibidora (BITTENCOURT, 2006). O neurônio então efetua o somatório de cada entrada multiplicada pelo seu peso e passa o valor obtido por uma função de ativação (Figura 4.4.3), que então produzirá sua saída, ou seja, seu nível de ativação. Ainda tem-se as funções de ativação diferenciáveis, aplicadas quando necessário, sendo as mais comuns (ARTERO, 2009; HAYKIN, 2001):

- a função Logística:

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

- a Tangente Hiperbólica:

$$f(x)=\frac{1-e^{-2x}}{1+e^{-2x}}$$

- e a Sigmóide:

$$f(x)=\frac{1}{1+\exp(-ax)}$$, onde a é o parâmetro de inclinação da função.

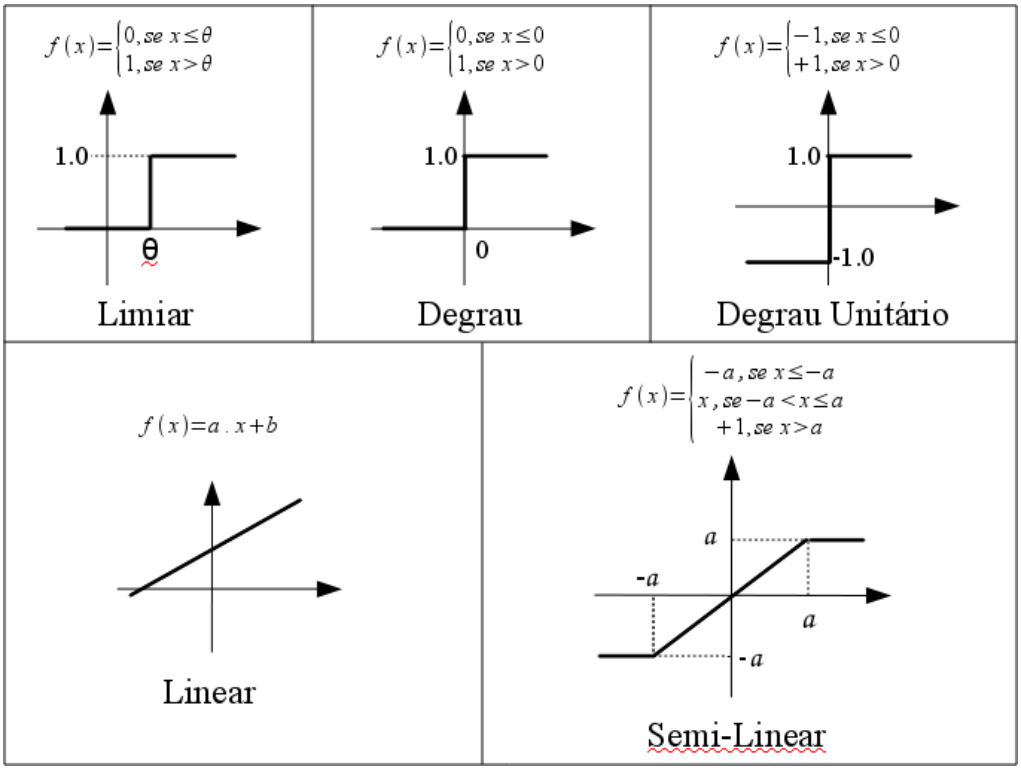


Figura 4.4.3: Funções de ativação mais utilizadas. Adaptado de (ARTERO, 2009)

Para que uma RNA aprenda é necessário efetuar um treinamento anterior ao seu uso utilizando algoritmos para a correção de seus valores internos, esta etapa de aprendizado se repete várias vezes, sendo cada iteração denominada época. As redes neurais artificiais são caracterizadas quanto ao seu aprendizado, que pode ser: supervisionado, possuindo um supervisor externo à rede que indica o acerto ou erro na resposta, indicando assim a necessidade de correção dos valores internos da rede; ou não supervisionado, que não possui um supervisor, efetuando assim o treinamento com intuito de classificar os dados de entrada em classes diferenciadas pelas suas características (REZENDE, 2005). Cada tipo de aprendizado possui algoritmos diferentes para o treinamento, podendo ainda serem utilizados em diversos modelos de rede, caracterizados pela quantidade de camadas que utilizam e distribuição de suas ligações entre os neurônios. Dos modelos mais conhecidos, temos (ARTERO, 2009; BITTENCOURT, 2006):

– *Perceptron*:

  - aplicação: reconhecimento de caracteres;

- vantagem: rede neural mais antiga e conhecida;
- desvantagens: não reconhece padrões complexos; não é sensível a mudanças;

– *Backpropagation*:

- aplicações: possui diversas aplicações, como a classificação supervisionada e reconhecimento de padrões;
- vantagens: rede muito utilizada, é simples e eficiente;
- desvantagens: treinamento supervisionado; o treinamento deve ter muitos conjuntos de entradas;

– *Hopfield*:

- aplicações: recuperação de dados; fragmentos de imagens;
- vantagem: implementação em larga escala;
- desvantagens: sem aprendizado; pesos preestabelecidos;

– *Bidirectional Associative Memories*:

- aplicação: reconhecimento de padrões;
- vantagem: estável;
- desvantagens: pouco eficiente;

– *Kohonen*:

- aplicação: reconhecimento de padrões não especificados;
- vantagem: auto-organização;
- desvantagens: pouco eficiente.

Dentre os diversos modelos de redes neurais artificiais existentes e que ainda podem ser propostos pela mixagem dos métodos atuais, encontram-se a diversidade de propostas que podem ser aplicadas aos SDIs. Segundo (WU, 2010), as pesquisas atuais propõem as redes apresentadas na Figura 4.4.4 para a solução de problemas de

detecção de intrusão.

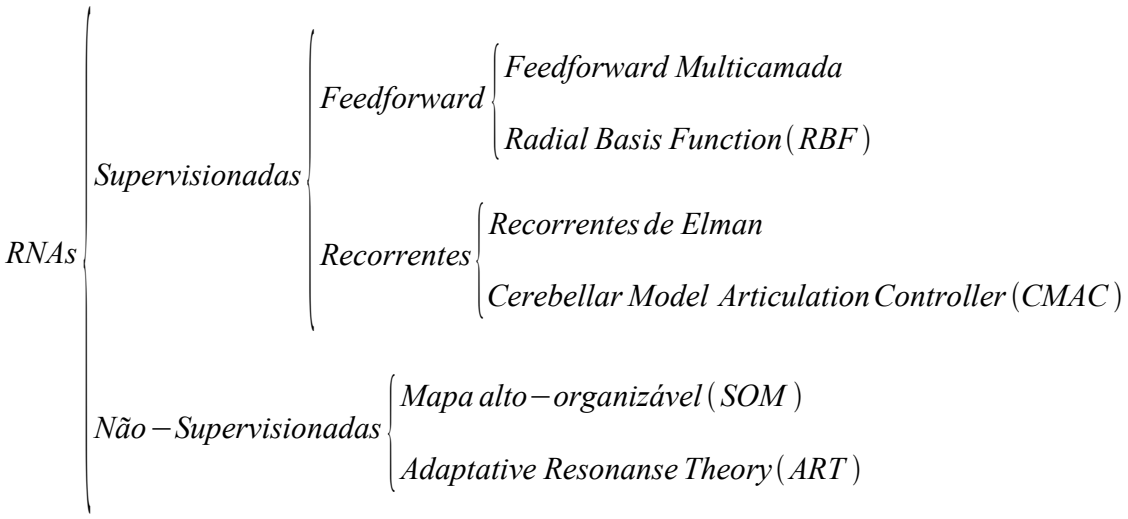


Figura 4.4.4: Tipos de RNA. Adaptado de (WU, 2010).

Os trabalhos que utilizaram redes *Feedforward* Multicamada (MLFF) obtiveram informações de conjuntos de comandos, da taxa de utilização da CPU, do endereço do usuário que logou no sistema, dentre outros, para detectar anomalias no sistema, enquanto outro trabalho utilizou padrões de comandos e sua frequência. Depois os trabalhos começaram a utilizar o comportamento do sistema ao invés do comportamento do usuário, obtendo então as informações das chamadas de sistema. O tráfego da rede também demonstrou ser importante, com trabalhos que utilizaram redes MLFF com *BackPropagation* para detectar invasões por abuso e com trabalhos que utilizaram essas redes como *Multi-class Classifier* (MCC), que podem ter mais neurônios de saída ou podem servir para montar várias redes neurais de classificação. Os algoritmos de aprendizado para as redes são vários e dentre os doze tipos diferentes utilizados no KDD99, o melhor foi o *Resilent Backpropagation*, que apresentou exatidão de 97,04% e tempo de treinamento de 67 épocas.

Outro tipo muito utilizado de rede neural é o *Radial Basis Function* (RBF), elas são mais rápidas que o *backpropagation* e são mais adequadas para problemas com tamanho de amostra grande. Alguns trabalhos referenciados em (WU, 2010) utilizaram essas redes para aprender a identificar grupos de padrões de ataques bem conhecidos e eventos normais, outros já utilizaram a junção dos resultados de múltiplos classificadores, utilizando cinco tipos diferentes de funções de decisão de fusão, como a *Weighted*

Majority Vote e a *Dempster-Shafer*. O trabalho de (JIANG, ZHANG e KAME, 2003), citado em (WU, 2010), apresentou uma proposta que dividia a classificação em duas etapas: a primeira identificando os ataques por abuso e a segunda verificando grupos de anomalias detectadas, ou seja, inicialmente ocorria a divisão dos tipos de ataques através de detectores de abuso, que ao não identificarem o tipo, salvava-os em uma base de dados para uma análise posterior por anomalias.

Em comparações efetuadas sobre as redes MLFF com *Backpropagation* e as redes RBF, os experimentos demonstraram que, para ataques por abuso, as redes com *backpropagation* tiveram uma desempenho um pouco melhor na faixa de detecção e na faixa de falso positivos, mas requerem um longo tempo de treinamento, já as redes RBF requerem um menor tempo de treinamento e apresentam alto grau de detecção e baixa faixa de falso positivos.

Outro fator de interesse é relacionado ao tempo, a representar memórias entre os dados de entrada. As rede neurais recorrentes possuem essa características, sendo inicialmente utilizadas como prognosticador, onde prediziam os próximos eventos de uma sequência de entrada. Uma modificação com as redes de *Elman*, utilizando o tempo *t-1* e *t*, já foi realizada, demonstrando que o prognóstico pode chegar a 80% de eficiência. Um trabalho também já comparou as redes recorrentes com as redes MLFF com *Backpropagation*, demonstrando que as redes recorrentes são melhores prognosticadoras.

As redes recorrentes também já foram treinadas como classificadores, onde destacou-se na pesquisa a importância do *payload* do pacote para o diagnóstico melhor da rede recorrente. Dois trabalhos existentes demonstram que as redes de *Elman* superam as redes MLFF em capacidade de detecção e capacidade de generalização. Também destacou-se uma sobrecarga de processamento relacionada ao treinamento e a operação dessas redes.

Outro tipo de redes neurais recorrentes é o *Cerebellar Model Articulation Controller* (CMAC), que tem capacidade de aprendizado extra, evitando o treinamento da rede a cada novo ataque. Algumas pesquisas apresentaram uma rede CMAC modificada para receber uma realimentação do ambiente, como carga da CPU e memória disponível. Através de experimentos, foi apresentado que essas redes aprenderam novos ataques em

tempo real e conseguiram generalizar bem os padrões de ataques similares.

Com aprendizado não supervisionado, temos as redes com Mapa Alto Organizável (SOM) e as redes *Adaptative Resonance Theory* (ART). As saídas destes tipos de redes neurais são grupos que classificam as entradas por similaridade. Suas implementações podem ser utilizadas em detecção de intrusão quando apresentam capacidade de separar o comportamento normal em grupos específicos, para assim classificar como ataque as entradas pertencentes aos demais grupos.

As redes SOM também são conhecidas como mapas de Kohonen, sendo redes *feedforward* de uma única camada, onde as saídas são agrupadas por similaridade em uma grade dimensional 2D ou 3D. Nas pesquisas, essas redes geralmente são treinadas para detecção de anomalia, sendo utilizadas para detectar vírus em máquinas ou para aprender o comportamento normal de sistemas. Porém já tiveram pesquisas para trabalhar com detecção de abuso, nesse caso, as redes foram utilizadas como preprocessadoras para separar os dados de entrada e fornecê-los a outros algoritmos de classificação, como redes *feedforward*.

Conclusões de estudos posteriores relataram também que cada camada de um protocolo de rede tem sua estrutura e suas funções próprias, fazendo com que ataques que utilizam especificamente um protocolo sejam únicos também. Isso culminou o desenvolvimento de redes SOM separadas para cada camada, como também a divisão da análise das entradas em fases, o que acarretou uma diminuição significante nos falso positivos. Alguns pesquisas também foram realizadas utilizando de duas a cinco camadas nas redes neurais, porém seus resultados apresentaram muitos falso positivos.

Os outros tipos de RNAs citados nesse grupo, são as redes ART, que englobam várias redes com aprendizado supervisionado e não supervisionado, sendo utilizadas para reconhecimento de padrões e para predição. Alguns modelos que trabalham com aprendizado não supervisionado são o ART-1, o ART-2, o ART-3 e o *Fuzzy* ART. Em pesquisas foi relatado que as redes ART-1 tem uma detecção melhor que as redes ART-2, porém as redes ART-2 são de 7 a 8 vezes mais rápidas que as redes ART-1. Em demais pesquisas, relatou-se que as redes ART tiveram um desempenho bem melhor que as redes SOM na detecção offline e online de detecção de intrusão. As redes *Fuzzy* ART também

foram testadas com sequências de entradas arbitrárias, foram mais rápidas e estáveis quanto as redes ART sozinhas. Em outras comparações, os pesquisadores destacaram as redes *Fuzzy* ART e as redes SOM como promissoras para detectar o comportamento anormal da rede de computadores, porém a sensibilidade das redes *Fuzzy* ART foi bem maior que das redes SOM (WU, 2010).

4.5 Sistemas Imunológicos Artificiais

Os Sistemas Imunológicos Artificiais (SIA) são modelados com base no sistema biológico de imunidade que é composto por um conjunto de detectores que buscam encontrar elementos maliciosos, agregá-los e destruí-los quimicamente, cuidando assim da defesa do sistema (BRITT, 2007). Este sistema possui duas linhas de defesa, uma capaz de controlar as infecções e outra capaz de conter ataques e manter a memória de como combatê-los para possíveis casos futuros. Desde sua proposta inicial em meados de 1980, os pesquisadores tentaram utilizar sua metáfora para desenvolver uma grande variedade de problemas. Alguns exemplos de algoritmos são: Seleção Negativa, Seleção Positiva e Seleção de Clonagem. Como alguns exemplos de teoria tem-se a Teoria *Self/Non-Self* e a Teoria *Danger* (SOBH e MOSTAFA, 2011).

Em sua aplicação à detecção de intrusão, tiveram diferentes abordagens, uma delas, por exemplo, implementou um algoritmo para detectar alterações nos arquivos e nas sequências de chamadas de sistemas com a Teoria *Self/Non-self*, outro implementou os componentes de seu sistema com ciclos de vida e demonstrou em trabalhos posteriores que foram necessários várias melhorias para obter os resultados, existem outros modelos que implementam algoritmos de multicamada, com redes de interligação, etc. Esses sistemas demonstram que a teoria dos SIA pode ser aplicada em sistemas de detecção e que várias abordagens trabalharam procurando encontrar o *non-self* do sistema, ou seja, as partes que não pertencem ao sistema, ao invés de trabalhar com assinaturas (WU, 2010).

Outros dois trabalhos sobre os Sistemas Imunológicos Artificiais (SIA) são os descritos em (UCHÔA, 2009), onde é apresentado o "desenvolvimento e análise de ferramentas de segurança computacional baseadas em algoritmos imunoinspirados", e o

trabalho de (KIM et al., 2007), que apresenta uma revisão dos SIA para Detecção de Intrusão.

4.6 Sistemas Inteligentes Híbridos

Os Sistemas Inteligentes Híbridos (SIH), também conhecidos como *Soft Computing*, são sistemas que utilizam duas ou mais técnicas distintas para resolver um problema, com pelo menos uma das técnicas sendo de Inteligência Artificial. Sua abordagem é interessante porque alguns métodos, mesmo com bons resultados em determinados problemas, apresentam deficiências para resolver outros tipos de problemas. Cada técnica, devido a sua limitação e/ou deficiência, pode não ser capaz de resolver sozinha determinado problema. Para isso, torna-se ideal a utilização de duas ou mais técnicas com intuito de combinar as vantagens e superar as desvantagens apresentadas por cada uma com intuito de resolver determinado problema, obtendo uma solução mais robusta e eficiente (REZENDE, 2005).

Os SIH são classificados nos tipos: Substituição de Função, quando "utiliza-se uma técnica para implementar uma das funções da outra técnica"; Híbridos Intercomunicativos, quando utiliza-se diversas subtarefas independentes para solucionar um problema complexo; e Híbridos Polimórficos, quando uma única técnica é adaptada para realizar uma tarefa inerente a uma outra técnica (REZENDE, 2005).

Dentre os Sistemas Inteligentes Híbridos, destacam-se os sistemas que combinam Redes Neurais Artificiais com Sistemas Nebulosos e os sistemas que combinam Computação Evolucionária e Sistemas Nebulosos.

Uma RNA é um modelo não linear que realiza classificações de padrões através de conexões internas entre unidades de processamento simples. Mesmo com bons resultados, não permite a incorporação de conhecimento especialista na aplicação. Já os conceitos nebulosos tratam conceitos vagos, imprecisos ou incertos, sendo utilizado como um apoio a decisão, sendo capaz de representar um conhecimento especialista e de interpolar decisões vindas de entradas com incertezas. Os sistemas nebulosos são capazes de representar algumas características da mente humana, sendo centrados no homem. A concentração dessas características podem ser obtidas através da combinação desses

métodos, denominados Sistemas Neuro *Fuzzy* (REZENDE, 2005).

Os conceitos nebulosos também podem trazer vantagens sendo combinados com os sistemas evolutivos, como algoritmos genéticos ou programação genética, diminuindo as deficiências e emergindo melhores resultados (WU, 2010).

Para a implementação, escolheu-se então utilizar Sistemas Inteligentes Híbridos, compostos por Redes Neurais Artificiais e Sistemas Nebulosos. A aplicação de Algoritmos Genéticos para a melhoria das RNAs também foi utilizada. Os detalhes da implementação serão apresentados no próximo Capítulo.

Capítulo 5

Implementação dos Métodos de Inteligência Artificial para Sistemas de Detecção de Intrusão

Escolheu-se para este trabalho a implementação de um sistema Inteligente Híbrido composto por Redes Neurais Artificiais do tipo MLP e Lógica Nebulosa (Neuro-Fuzzy). Esta escolha se deveu à capacidade de aprendizado, de classificação e de adaptação apresentada pelas RNAs e pela representação do conhecimento apresentado pelos sistemas nebulosos, que permitem incertezas e inserção de regras *se-então* sobre conjuntos de dados de forma mais plausível. Algoritmos Genéticos também foram escolhidos para a melhoria das características da Rede Neural para uma melhor taxa de acertos.

Para a implementação deste trabalho optou-se por utilizar ferramentas livres (OPENSOURCE, 2011), por fornecerem mais flexibilidade e mais facilidade de acesso às pessoas que as utilizam. Nesses sistemas, todo o código pode ser lido e modificado, ajudando assim a entender melhor e a contribuir com o aprendizado de outros, além de contar com a liberdade de distribuição e da liberdade de saber o que está utilizando.

O sistema desenvolvido é formado pela combinação das saídas de uma Rede Neural Artificial e de uma Lógica Nebulosa. Para tal, inicialmente desenvolveu-se um sistema nebuloso e uma RNA simples, que foi modificada e melhorada por um Algoritmo Genético. Após este processo, suas respostas às conexões da rede de computadores eram

processadas para produzir uma resposta final. Para melhor sistematizar a apresentação do desenvolvimento deste trabalho, a seção 5.1 apresenta a definição das bibliotecas e da base de dados utilizadas na implementação e a seção 5.2 apresenta mais detalhes sobre a implementação.

5.1 Definição dos mecanismos de implementação

Inicialmente a linguagem de programação Python (PYTHON, 2011) foi escolhida para implementar este trabalho, por apresentar as seguintes características:

- possuir biblioteca para implementação das Redes Neurais – PyFANN (FANN, 2011);
- possuir biblioteca para a implementação de conjuntos nebulosos – FuzzPy (FUZZPY, 2011);
- ser uma linguagem robusta e de *script,* fornecendo a vantagem de verificar imediatamente o resultado da saída de cada comando, dentre outras diversas qualidades da linguagem (PYTHON, 2011);
- possuir suporte ao *libpcap* (TCPDUMP, 2011), que futuramente poderá ser utilizado para implementação de um SDI com métodos de IA.

Como mencionado, para a implementação das redes neurais artificiais foi utilizada a FANN (*Fast Artificial Neural Network Library*), uma biblioteca livre que implementa redes neurais artificiais em C e fornece um *framework* para criação, treinamento e utilização das redes neurais. Também fornece integração com outras linguagens de programação, como PHP, Python, Object Pascal, etc., e possui uma boa documentação (FANN, 2011). Dentre suas características, nesta seção serão expostas as essenciais e utilizadas no trabalho.

Cada neurônio de uma Rede Neural Artificial pode possuir uma função de ativação, porém geralmente as funções são definidas por camadas. A biblioteca FANN possui suporte às seguintes funções de ativação e que serão utilizadas como genes do Algoritmo Genético:

- funções de ativação linear, linear por partes e linear por partes simétricas;
- funções de ativação limiar (*threshold*) e limiar simétrica:
- funções de ativação sigmóide, sigmóide passo a passo, sigmóide com gradiente e sigmóide simétrica com gradiente;
- funções de ativação gaussiana e gaussiana simétrica;
- funções de ativação *elliot* e *elliot* simétrica;
- funções de ativação seno e seno simétrica; e
- funções de ativação cosseno e cosseno simétrico.

Após a definição da estrutura da Rede Neural Artificial e das funções de ativação, deve-se definir o algoritmo de treinamento, que atualizarão os pesos da RNA a procura de uma resposta final melhor. Os algoritmos suportados pela biblioteca FANN e que serão utilizados pelo Algoritmo Genético são:

- *Backpropagation*: com atualização dos pesos após cada treino e com atualização dos pesos após o cálculo do erro quadrático médio de todo o conjunto de treino. O algoritmo *Backpropagation* ajusta os pesos das camadas intermediárias através da retropropagação dos erros, que são calculados com derivadas parciais, aplicadas sobre o erro da saída em relação a um peso específico da RNA (REZENDE, 2005);
- Retropropagação Resiliente (Rprop): uma melhoria do algoritmo *Backpropagation*, que atualiza os pesos sem a influência das derivadas parciais (CASTANHEIRA, 2008);
- *Quick Propagation* (QuickProp): uma melhoria do algoritmo *BackPropagation*, que efetua a atualização dos pesos de uma forma mais rápida. Neste treinamento, inicialmente é calculada a primeira derivada parcial de cada peso para então calcular e utilizar o gradiente descendente dentro do espaço de erros (FAHLMAN, 1988).

Esses algoritmos podem ser ajustados definindo-se valores diferentes dos padrões

para a taxa de aprendizado, o *momentum*, as décadas de treinamento, o limite de erro, etc., através de variáveis e métodos da biblioteca, permitindo assim influenciar as expressões algébricas utilizadas pelos algoritmos de treinamento para melhorar o aprendizado da rede. Mais informações da biblioteca podem ser encontradas em (FANN, 2011).

Para buscar melhores resultados nas Redes Neurais Artificiais, optou-se por utilizar Algoritmos Genéticos, transformando os itens que compõe uma RNA em genes de cromossomos que participarão da população genética, que evoluirá através de cruzamentos e mutações para retornar os melhores genes de uma Rede Neural para este caso de estudo. A escolha de melhores genes pode gerar em uma melhor combinação de funções de ativação, de uma melhor escolha da quantidade de neurônios ocultos ou do algoritmo de treinamento, dentre outros fatores. Para o tal possibilidade, foi implementada uma API Genética baseando-se nas bases teóricas de (COPPIN, 2010), (REZENDE, 2005) e (ARTERO, 2009). Os passos definidos para a API foram:

1. gerar a primeira população, sendo composta por cromossomos (indivíduos) formados de genes aleatórios;
2. determinar a aptidão de cada indivíduo;
3. verificar se o critério de terminação foi satisfeito,
    - caso positivo, parar;
    - caso contrário, seguir para o passo 4;
4. aplicar o cruzamento aos indivíduos da população e selecionar alguns para aplicar a mutação (esses passos gerarão a nova população);
5. determinar a aptidão de cada indivíduo da nova população
6. avaliar todos os indivíduos e manter aqueles com melhor grau de aptidão;
7. retornar ao passo 3.

Mais detalhes de sua implementação podem ser encontrados na seção 5.2 e todo o código da API pode ser encontrado no “Anexo I – Algoritmo Genético”.

Já como mecanismo para auxiliar na implementação da lógica nebulosa, utilizou-se a biblioteca FuzzPy (FUZZPY, 2011), que possui uma base para a criação de conjuntos nebulosos e para operações matemáticas entre os mesmos. Os conjuntos que a biblioteca suporta são o triangular, o trapezoidal, o poligonal e o gaussiano. Porém, como no trabalho seguiu-se o modelo apresentado em (DASGUPTA, 2001), foram necessários somente os conjuntos triangulares e trapezoidais.

Após a implementação de todo o sistema, escolheu-se a base de dados NSL-KDD para os testes e comprovação dos métodos inteligentes (NSL-KDD, 2011). Estes dados tem origem no conjunto de dados pertencentes ao KDDCUP'99 (KDDCUP99, 2010), sendo este uma versão da base inicial de dados criada pelo *MIT Lincoln Labs*, para o Programa de Avaliação de Detecção de Intrusão DARPA de 1998. Sua base de dados consiste em aproximadamente 4.900.000 vetores simples de conexão, cada um contendo 41 características e um rótulo indicando se é normal ou qual o tipo de ataque que pertence (KDDCUP99, 2010; TAVALLAEE, 2009). Os tipos de ataques utilizados são mostrados na Tabela 5.1.1, sendo classificados em (KENDALL, 1999):

- Ataques de Negação de Serviço (DoS): são aplicados para sobrecarregar algum recurso de memória ou recurso computacional para que usuários legítimos não consigam acessar o sistema. Alguns desses ataques (*mailbomb*, *neptune*, *smurf attack*, etc.) abusam de uma característica específica do serviço atacado, outros (*teardrop* e *ping of death*) já criam pacotes com formatos errados, o que atrapalha a reconstrução do pacote, enquanto outros (*apache2*, *back*, *syslogd*) utilizam erros de determinado serviço de rede;
- Ataques de Usuário para Administrador (U2R): ocorrem quando o invasor consegue acessar o sistema como usuário normal e então explora vulnerabilidades para conseguir acesso de administrador. Dentre os diversos tipos de ataques, o mais comum é o de *overflow*, que ocorre quando um programa copia muitos dados em um *buffer* estático sem verificação, permitindo ao usuário inserir mais dados do que os necessários, podendo então executar códigos arbitrários;

Sistema Vulnerável / Tipo de Ataque	Solaris	SunOS	Linux
DoS	Apache2 Back Mailbomb Neptune Ping of Death Process Table Smurf Syslogd UDP Storm	Apache2 Back Land Mailbomb Neptune Ping of Death Process Table Smourf UDP Storm	Apache2 Back Mailbomb Neptune Ping of Death Process Table Smurf Teardrop UDP Storm
R2U	Dictionary FTP-Write Guest PHF Xlock Xsnoop	Dictionary FTP-Write Guest PHF Xlock Xsnoop	Dictionary FTP-Write Guest Imap Named PHF SendMail Xlock Xsnoop
U2R	Eject ffbconfig fdformat pq	Loadmodule ps	Perl xterm
Probing	ip sweep mscan nmap saint satan	Ip sweep mscan nmap saint satan	ip sweep mscan nmap saint satan

Tabela 5.1.1: Ataques da base de dados KDDCUP. Adaptado de (Kendall, 1999)

- Ataques de Remoto para Local (R2L): ocorrem quando um atacante consegue ganhar acesso em um sistema que ele não tem acesso através de envio de pacotes na rede de computadores. Existem várias formas de efetuar estes ataques, alguns (*Dictionary*, *FTP-Write*, *Guest*) podem explorar por segurança de políticas fracas, outros (*imap*, *named, sendmail*) podem utilizar *buffer overflow*, ou ainda podem utilizar engenharia social para infiltrar *trojans* (Xlock) e obter senhas de acesso;

- Ataques de Reconhecimento (*Probing*): são utilizados para escanear

uma rede de computadores para encontrar informações sobre os sistemas instalados, ou mesmo para encontrar diretamente suas vulnerabilidades. Um atacante que possua um bom mapa de uma rede, pode obter informações sobre pontos mais fracos da rede. Alguns aplicativos (*satan*, *saint*, *mscam*) podem ser utilizados para varrer uma grande área de rede procurando por vulnerabilidades conhecidas.

Mais detalhes de cada tipo de ataque utilizado para a criação da base de dados KDDCUP podem ser encontrados em Kendall (KENDALL, 1999). As características dos dados do KDDCUP'99 podem ser classificadas em três grupos:

1. características básicas: reúne os dados que podem ser extraídos de uma conexão TCP/IP;
2. características de tráfego: reúne dados sobre os recursos fornecidos na rede. São calculadas em um intervalo de janela de tempo, sendo:
   - características de um mesmo *host:* informações apenas das conexões nos últimos dois segundos que têm o mesmo destino de *host* que a conexão atual; e
   - características de um mesmo serviço: informações apenas das conexões nos últimos 2 segundos que tem o mesmo serviço que a conexão atual;

   Essas características são baseadas em tempo, mas não podem servir para identificar todos os tipos de ataque, visto que existem ataques de reconhecimento lentos que possuem um intervalo de tempo muito maior que dois segundos. Para que esses tipos de ataques pudessem ser melhor identificados, os recursos para a mesma máquina e para o mesmo serviço foram recalculados, mas com base em uma janela de conexão de 100 ligações ao invés de uma janela de tempo de dois segundos.
3. recursos de conteúdo: informações mais referentes sobre ataques que utilizam conexões normais, estabelecidas entre dois *hosts* e com tempo normal de outra conexão, ao invés da maioria dos ataques DoS e de

sondagem. Geralmente são referentes a ataques R2L e U2R, embutidos em parcelas de dados dos pacotes e, normalmente, envolvendo apenas uma única conexão.

As especificações desses campos estão expostas nas Tabelas 5.1.2, 5.1.3, 5.1.4. Os vetores de conexão são separados em dois conjuntos, um de treinamento e outro de testes, para avaliar a capacidade dos sistemas de identificar corretamente o tráfego malicioso da rede.

Nome	Descrição	Tipo
duration	Tempo em segundos de conexão	Contínuo
protocol_type	Tipo de protocolo utilizado	Discreto
service	Serviço de rede sendo utilizado	Discreto
flag	Estado da conexão (normal ou erro)	Discreto
src_bytes	Número de bytes enviados da fonte para o destino	Contínuo
dst_bytes	Número de bytes enviados do destino para a fonte	Contínuo
land	1 se a conexão é de/para o mesmo host; 0 caso contrário	Discreto
wrong_fragment	Número de fragmentos com erro	Contínuo
urgent	Número de pacotes com flag urgente	Contínuo

Tabela 5.1.2: Características básicas de uma conexão TCP (EUSAM, 2008; KDDCUP, 2010)

Nome	Descrição	Tipo
hot	Número de indicadores chave (*hot*)	Contínuo
num_failed_logins	Número de tentativas de login sem sucesso	Contínuo
logged_in	1 se login efetuado com sucesso; 0 caso contrário	Discreto
num_compromissed	Número de conexões de "comprometimento"	Contínuo
root_shell	1 se *shell root* foi obtido; 0 caso contrário	Discreto
su_attempted	1 se comando "*su root*" foi tentado; 0 caso contrário	Discreto
num_root	Número de acessos como *root*	Contínuo
num_file_creations	Número de operações de criação de arquivos	Contínuo
num_shells	Números de "*shell prompts*" obtidos	Contínuo
num_access_files	Número de operações em arquivos de controle de acesso	Contínuo
num_outbounds_cmd	Número de comandos externos em uma sessão ftp	Contínuo
is_root_login	1 se login pertence à lista "*hot*"; 0 caso contrário	Discreto
is_guest_login	1 se login utilizou a conta *guest*; 0 caso contrário	Discreto

Tabela 5.1.3: Características da conexão por conhecimento especialista (EUSAM, 2008; KDDCUP, 2010)

Nome	Descrição	Tipo
count	Número de conexões iguais a esta para este mesmo *host* nos últimos 2 segundos	Contínuo
srv_count	Número de conexões para o mesmo serviço que o usado nesta conexão nos últimos 2 segundos	Contínuo
serror_rate	% de conexões que possuem erro SYN	Contínuo
srv_error_rate	% de conexões que possuem erro SYN para este serviço	Contínuo
rerror_rate	% de conexões que possuem erro REJ	Contínuo
srv_rerror_rate	% de conexões que possuem erro REJ para este serviço	Contínuo
same_srv_rate	% de conexões para um mesmo serviço	Contínuo
diff_srv_rate	% de conexões para serviços diferentes	Contínuo
srv_diff_host_rate	% de conexões deste mesmo serviço para *hosts* diferentes	Contínuo
dst_host_count	Número de conexões com mesmo *host* de destino que esta	Contínuo
dst_host_srv_count	Número de conexões com mesmo *host* de destino e mesmo serviço que esta	Contínuo
dst_host_same_srv_count	% de conexões com o mesmo *host* de destino e mesmo serviço que esta	Contínuo
dst_host_diff_srv_rate	% de conexões com o mesmo *host* de destino e serviços diferentes que este	Contínuo
dst_host_same_src_port_rate	% de conexões com o mesmo *host* de destino e a mesma porta de origem que a conexão atual	Contínuo
dst_host_srv_diff_host_rate	% de conexões para o mesmo serviço vindo de diferentes *hosts*	Contínuo
dst_host_serror_rate	% de conexões para o mesmo *host* da conexão atual e que possui um erro S0	Contínuo
dst_host_srv_serror_rate	% de conexões para o mesmo *host* e serviço que o da conexão atual e que possui um erro S0	Contínuo
dst_host_rerror_rate	% de conexões para o mesmo *host* que apresentem *flag* RST	Contínuo
dst_host_srv_rerror_rate	% de conexões para o mesmo *host* e serviço da conexão atual que apresentem *flag* RST	Contínuo

Tabela 5.1.4: Características Temporais (EUSAM, 2008; KDDCUP, 2010)

Preocupados com a qualidade dos dados contidos no KDDCUP99, Tavallaee et al.

(TAVALAEE, 2009) realizaram estudos e argumentaram sobre problemas na coleta dos dados, como a sobrecarga da ferramenta de coleta (*tcpdump*) quando havia alto tráfego e a diferença dos dados fornecidos com os dados costumeiramente coletados em uma rede real, e sobre a falta de explicações sobre definições específicas dos ataques, pois uma sondagem (*probing*), por exemplo, não é necessariamente um ataque até que ultrapasse um limite. Também levantaram problemas argumentados em outros artigos, sobre diversas críticas a base KDD, como por exemplo, a quantidade desproporcional de ataques nas bases de dados.

Para melhorar a qualidade dos dados, uma nova base foi proposta, o NSL-KDD (NSL-KDD, 2011), que retirou os dados redundantes e distribuiu de outra forma os dados do KDDCUP99, separando os dados em três conjuntos:

1. KDDTrain+: que possui a base de treino;
2. KDDTest+: que possui a base de testes;
3. KDDTest^{-21}: que possui os dados que não foram bem classificados pelos algoritmos utilizados na criação da base.

Além desses, eles criaram um conjunto com 20% dos dados de treino e aplicaram em diferentes métodos de aprendizagem fornecidos pelo Weka (WEKA, 2011) na base original do KDDCUP e em suas bases de testes, fornecendo os resultados contidos na Tabela 5.1.5.

Método/Base de Teste	KDDTest (original)	KDDTest+	KDDTest^{-21}
J48	93,82%	81,05%	63,97%
Árvore NB	93,51%	82,02%	66,16%
Random Forest	92,79%	80,67%	63,26%
Random Tree	92,53%	81,59%	58,51%
Perceptron Multicamadas (MLP)	92,26%	77,41%	57,34%
Redes Bayesianas	81,66%	76,56%	55,77%
SVM	65,01%	69,52%	42,49%

Tabela 5.1.5: Taxa de acerto dos métodos de aprendizado nas bases de teste (TAVALLAEE, 2009)

A configuração de cada método utilizou como configuração os padrões já

fornecidos pelo Weka. Todos os métodos implementados apresentaram taxas menores de acerto para os dados do NSL-KDD m relação às taxas obtidas no KDDCUP. Os métodos J48 e Árvore NB forneceram taxas altas de acerto nos dados do KDDCUP e mantiveram valores altos, quanto aos demais algoritmos, nas bases do NSL-KDD. Os métodos *Random Forest* e *Random Tree* obtiveram resultados com leve variação na base KDDCUP e nas taxas do NSL-KDD. As Redes Neurais Artificiais Perceptron Multicamadas já apresentaram alta taxa de acerto nos dados do KDDCUP e taxas inferiores de acerto no NSL-KDD. Os métodos de Redes Bayesianas e de SVM não apresentaram altas taxas de acerto no KDDCUP e mantiveram valores inferiores aos demais algoritmos nas bases do NSL-KDD.

O trabalho de Tavallaee (TAVALLAEE, 2009) apresentou somente a taxa de acerto sobre todos os dados, omitindo os valores sobre as taxas de Falsos Positivos, Falsos Negativos, Verdadeiros Positivos e Verdadeiros Negativos. Dessa forma, não é possível verificar se o método atingiu um valor mais alto de acerto devido a uma taxa específica ou a todas as taxas possíveis nos testes.

5.2 Implementação

Primeiramente uma rede neural artificial com os valores padrões do Weka foi implementada para atingir a mesma taxa de acerto da rede neural implementada no conjunto de testes do NSL-KDD. Logo após foi criada uma API para fornecer base para a utilização da Rede Neural com o Algoritmo Genético. Dos resultados, obteve-se uma nova rede neural com maior taxa de acerto. Em paralelo estavam ocorrendo os estudos e implementação do conjunto nebuloso que também forneceria respostas ao tráfego da rede. Ao terminar das implementações dos métodos, foi realizada sua união para a formação do sistema Neuro-Fuzzy. Os tópicos a seguir descrevem cada etapa da implementação descrita.

5.2.1 Implementação da Primeira Rede Neural

Como ponto de partida, implementou-se a mesma Rede Neural Artificial utilizada no trabalho de Tavallaee (TAVALLAEE, 2009). Como a RNA Multicamada (MLP) utilizada

foi criada seguindo os padrões fornecidos pelo aplicativo Weka, tornou-se necessário levantar tais valores, sendo (WEKA, 2011):

- camadas: entrada (41 neurônios), oculta e de saída (2 neurônios);
- quantidade de neurônios ocultos: (atributos+classes)/2 = (41+2)/2 = 21 neurônios;
- algoritmo de aprendizado: *Backpropagation;*
- taxa de Aprendizado (*Learning Rate*): 0.3;
- *momentum:* 0.2;
- épocas de treinamento: 500;
- funções de ativação: linear na primeira camada e sigmoidais nas camadas seguintes;
- *decay*: 0.0.

A implementação da rede neural com as características apresentadas foi realizada através da biblioteca FANN, utilizando sua interface em Python. Sendo executado mais de uma vez, até seu treinamento fornecer taxas compatíveis com Tavallaee (TAVALLAEE, 2009). O código fonte criado pode ser encontrado no "Anexo II – Criação e execução da 1ª RNA" e os resultados são apresentados e discutidos no próximo capítulo.

5.2.2 Implementação do Algoritmo Genético

As redes neurais multicamadas podem ter diferentes formações, com diferentes funções de ativação, algoritmos de aprendizado, quantidade de neurônios ocultos, taxas de aprendizado, etc. Além disso, apresentam diferentes aprendizados dependendo das épocas de treinamento, influindo também em sua capacidade de generalização.

O trabalho de criar uma rede neural manualmente e treiná-la provoca uma ociosidade que prejudica o projeto, além de não garantir a escolha de um modelo ótimo. Devido a isso, utilizou-se um Algoritmo Genético para a busca de uma estrutura para a

RNA que fornecesse um tempo de treinamento menor e uma taxa de acerto maior. Ele realizou várias permutações e testes nas redes neurais durante 24 horas.

O Algoritmo Genético foi implementado em três classes:

1. classe Cromossomo: classe que abriga os genes e o valor de aptidão do cromossomo;
2. classe Populacao: classe que abriga os cromossomos (indivíduos) de uma população, sendo também a responsável por gerar e avaliar uma população, além de fazer o cruzamento e a mutação; e
3. classe AlgGenetico: classe responsável por manipular a classe população e controlar todo o processo do Algoritmo Genético, requisitando as mutações e o cruzamento dos indivíduos da população, além de requisitar a avaliação dos indivíduos, de controlar as épocas e o critério de satisfação.

Para utilizar a API, basta seguir os seguintes passos:

1. importar a classe AlgGenetico:

```
from algoritmoGenetico import AlgGenetico
```

2. criar uma matriz com os genes possíveis de cada cromossomo. O gene pode ser qualquer objeto: um número, uma função, uma palavra, etc. Ex:

```
tipoGenes = [ [0,1] , [3,4] , ['coringa','azar',''] ]
```

3. criar uma função que realizará a avaliação do indivíduo. Ela deve obrigatoriamente receber um parâmetro, que será um objeto da classe Cromossomo, ex:

```
def avaliaCromossomo( cromo ):
   genes = cromo.getCromossomo()
   if   genes[2] == 'azar':    return 0
   elif genes[2] == 'coringa': return 5
   else:
      return int(genes[0])+int(genes[1])
```

4. criar um objeto da classe AlgGenetico, passando a matriz de genes, a quantidade de indivíduos de uma população, função de avaliação e o critério de satisfação e/ou o máximo de gerações, além de outros valores opcionais. Por exemplo:

```
AG = AlgGenetico( tipoGenes, 20, avaliaCromossomo,
                  considerarMaiores=True,
                  criterioSatisfacao=5 )
```

5. logo após, deve-se efetuar a chamada do método "procriar":

```
resultado = AG.procriar()
```

6. para imprimir os resultados na tela, basta percorrer os cromossomos (indivíduos) da população retornada:

```
for cromossomo in resultado:
   print cromossomo.getCromossomo(),
   print cromossomo.getAvaliacao()
```

O código implementado encontra-se no "Anexo I – Algoritmo Genético".

Para que o algoritmo genético pudesse buscar uma rede neural com maior taxa de acerto, foi criada uma matriz contendo os seguintes genes:

- vários valores para a taxa de aprendizado: 0,1; 0,2; 0,3; 0,4; 0,5; 0,6; 0,7; 0,8; 0,9;
- vários valores para o *momentum*: 0,1; 0,2; 0,3; 0,4; 0,5; 0,6; 0,7; 0,8; 0,9;
- vários valores para o *decay*: 0,1; 0,2; 0,3; 0,4; 0,5; 0,6; 0,7; 0,8; 0,9;
- a quantidade de neurônios ocultos;
- as funções de treinamento disponíveis no FANN;
- as funções de ativação disponíveis no FANN;
- a quantidade de épocas de treinamento.

Além da matriz, foi criada uma função de avaliação que obtinha os valores do

cromossomo, criava a rede neural equivalente, treinava com a base de treino do NSL-KDD pela quantidade de épocas especificada e retornava a taxa de erros (Falso Positivos + Falso Negativos) da saída da Rede Neural Artificial sobre a base de teste do NSL-KDD.

Os valores especificados para o algoritmo genético foram: uma população de 20 indivíduos; um critério de satisfação de 0.10 (referente a 10% de erro); o máximo de 100 gerações; e a opção de considerar os valores menores de avaliação ao invés dos maiores.

Porém, um problema foi encontrado com a avaliação dos indivíduos da população. No treinamento da rede neural, às vezes surgia um erro, devido à incompatibilidade de alguns algoritmos de treinamentos a algumas funções de ativação. Por padrão a biblioteca do FANN apresentava mensagens de erro na tela de forma infinita e sobrecarregava a memória do computador, o que causava a morte do processo pelo sistema operacional. Para resolver esse problema, os seguintes passos foram adotados:

- o código fonte do FANN foi obtido e modificado, adicionando uma interrupção de parada (*exit*) que retornava a mensagem de erro referente. Assim, a criação das redes neurais não poderia sobrecarregar os recursos do computador, forçando o encerramento do programa principal;
- foi criado um *script* em Python independente capaz de receber os genes do cromossomo, criar, treinar e testar a rede neural utilizando a biblioteca FANN e retornar a taxa de erros;
- a função de avaliação foi modificada para fazer a chamada de um processo filho e assim somente tratar uma exceção caso a biblioteca da rede neural terminasse inesperadamente, ao invés de encerrar o processo pai. Essa função foi também responsável por passar os valores dos genes do cromossomo a ser avaliado para o processo avaliador filho e recuperar o valor de retorno, retornando-o ao objeto do algoritmo genético.

Com os resultados (cromossomos da última população) do Algoritmo Genético, foi necessário criar a RNA com os atributos apresentados e treiná-la até obter avaliação compatível com a avaliação apresentada no cromossomo referente. Isto deve-se à forma

da biblioteca FANN definir os primeiros pesos de uma Rede Neural Artificial, pois são utilizados valores aleatórios. Porém, após o treinamento, o valor retornado de uma RNA é sempre o mesmo para um mesmo conjunto de entrada. O código implementado pode ser encontrado no "Anexo III – Execução do Algoritmo Genético para formar a RNA" e os resultados são apresentados no capítulo de seguinte.

5.2.3 Implementação dos Sistema Nebuloso

A lógica nebulosa foi implementada baseando-se no trabalho de Dasgupta (DASGUPTA, 2001) sobre os dados do KDDCUP99. Inicialmente, utilizando a biblioteca FuzzPy, criou-se os conjuntos nebulosos para cada atributo do conjunto de dados NSL-KDD, transformando-os em Variáveis Linguísticas, cada um com seus conjuntos nebulosos. Os atributos discretos foram distribuídos em conjuntos clássicos, como por exemplo, o atributo *procotol_type* que passou a ser representado como Variável Linguística, possuindo três conjuntos clássicos (Figura 5.2.1): conjunto icmp, conjunto tcp, conjunto udp. Assim, ao pesquisar o grau de pertinência de um protocolo tcp no conjunto icmp, teríamos $\mu_{icmp}(tcp)=0.0$, e ao pesquisar o grau de pertinência de icmp no protocolo icmp, teríamos $\mu_{icmp}(icmp)=1.0$. Para que cada Variável Linguística pudesse seguir o modelo de Dasgupta, os atributos contínuos foram distribuídos somente em conjuntos trapezoidais e triangulares, onde seus valores, após normalizados, passaram a pertencer aos conjuntos trapezoidais Muito Baixo (MB) e Muito Alto (MA) e aos conjuntos triangulares Baixo (B), Médio (M) e Alto (A), como é apresentado na a Figura 5.2.2.

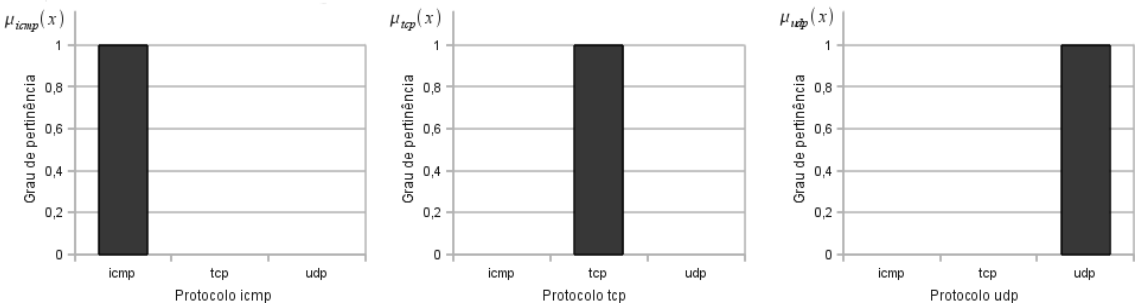


Figura 5.2.1: Conjuntos Nebulosos Discretos da Variável Linguística Protocolo

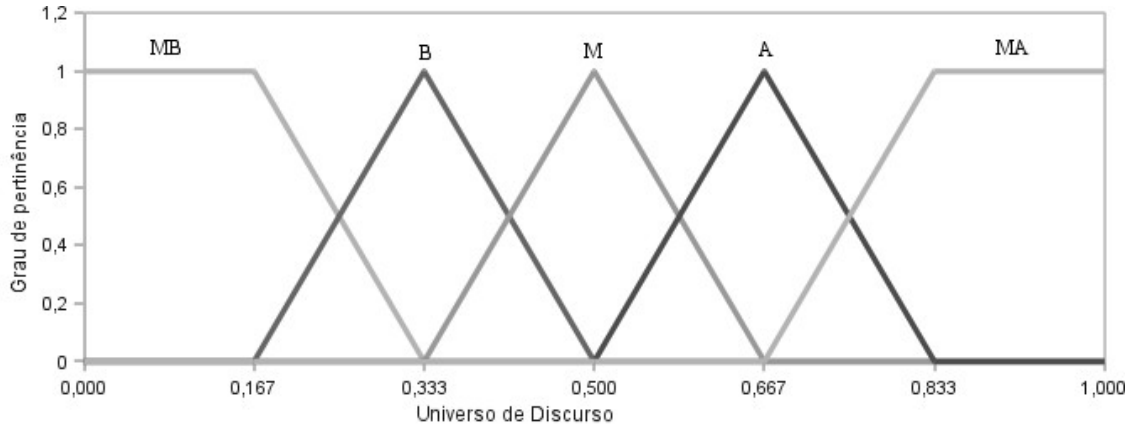


Figura 5.2.2: Conjuntos Nebulosos dos Atributos Contínuos

A normalização dos valores dos atributos contínuos seguiu a seguinte equação:

$$v = \frac{v - ValorMínimo}{ValorMáximo - ValorMínimo}$$

Para auxiliar na criação dos conjuntos nebulosos, foi implementada uma classe capaz de abrigar os dois tipos de Variáveis Linguísticas descritos acima. Como objetos dessa classe, todas as variáveis Linguísticas foram criadas. Exemplos:

```
duration = VariavelLinguistica( continuo=True, valorMinimo=0,
                                valorMaximo=57715,nome='duration',
                                indice=0 )
protocol_type = VariavelLinguistica( discreto=True,
                        valoresDiscretos=[ 'icmp','tcp','udp' ] ,
                        nome='protocol_type', indice=1 )
```

Após todo o conjunto de atributos ser representado como Variáveis Linguísticas, foi implementado um código capaz de percorrer todos os dados do NSL-KDD, aplicar as regras nebulosas e efetuar a defuzzificação através do Método do Critério Máximo, que considera o maior valor da saída como resultado. As regras utilizadas, basearam-se no conhecimento especialista extraído de (DASGUPTA, 2001) e foram adaptadas para o modelo implementado:

- ***SE*** (dst_host_srv_count ***NÃO*** *é baixo* ***OU*** protocol_type ***NÃO*** *é tcp*) ***E*** protocol_type ***NÃO*** *é icmp* ***ENTÃO*** conexão *é* normal;
- ***SE*** dst_host_srv_count *é baixo* ***E*** flag *não é S0* ***E*** protocol_type ***NÃO*** *é icmp* ***E*** dst_host_srv_rerror_rate ***NÃO*** *é alto* ***ENTÃO*** conexão *é* Ataque_U2R;

- ***SE*** (dst_host_srv_count *é baixo* ***OU*** is_guest_login *é verdade*) ***E*** flag ***NÃO*** *é REJ* ***E*** dst_host_same_srv_rate ***NÃO*** *é baixo* ***E*** duration ***NÃO*** *é alto* ***ENTÃO*** conexão *é* Ataque_R2L;
- ***SE*** count ***NÃO*** *é baixo* ***OU*** same_srv_rate *é baixo* ***ENTÃO*** conexão *é* Ataque_DoS;
- ***SE*** dst_host_same_srv_rate *é baixo* ***E*** flag ***NÃO*** *é SF* ***OU*** protocol_type *é icmp* ***ENTÃO*** *conexão é* Ataque_PRB.

Além dessas, a seguinte regra foi adicionada:

- ***SE*** protocol_type *é tcp* ***E*** service *é http* **E** flag *é RSTR* ENTÃO conexão *é* Ataque_DoS.

A seguinte condição especialista também foi adicionada:

- ***SE*** src_bytes *é 54540* ***ENTÃO*** conexão *é* Ataque_DoS.

A implementação referente encontra-se no "Anexo IV – Lógica Nebulosa" e os resultados encontram-se no próximo capítulo.

5.2.4 União dos resultados da RNAs e dos métodos de inferência Fuzzy

Após criar e obter os dados de cada sistema separadamente, foi realizada a união dos sistemas para verificar se haveria melhorias no resultado final. Para tal, as saídas (S_1=ataque, S_2=normal) da RNA com melhor resultado foram concatenadas às saídas das regras nebulosas. O resultado final foi avaliado sobre os conjuntos de resposta expostos na Figura 5.2.3, através do Método do Centro Geométrico (COG):

$$COG = \frac{\sum_{i=1}^{20} (5 \cdot i)\, \mu(5 \cdot i)}{\sum_{i=1}^{20} \mu(5 \cdot i)}$$

A implementação está disponível no "Anexo V – Neuro-Fuzzy" e os resultados estão disponíveis no capítulo 6.

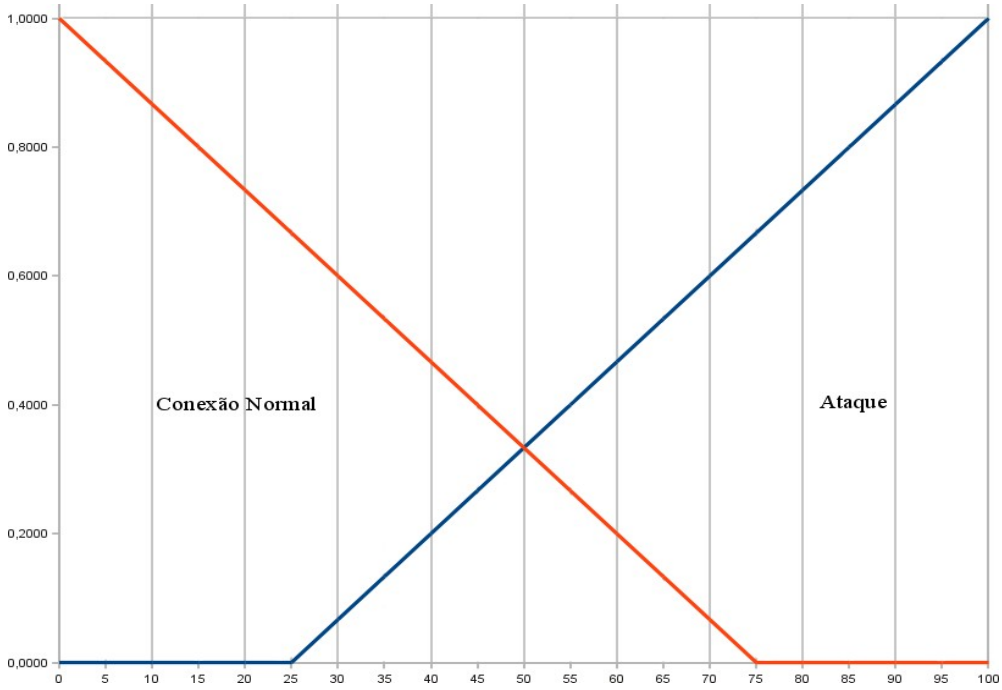


Figura 5.2.3: Conjuntos Nebulosos da Variável Linguística Resultado

Assim, por exemplo, se o maior grau de pertinência obtido dentre as regras de Conexão Normal fosse 0,65 e se o maior grau de pertinência obtido dentre as regras de Ataque fosse 0,55, o primeiro passo seria considerar somente os graus de pertinência até esses limites (área colorida da Figura 5.2.4). Agora, aplicar-se-ia a fórmula do Método do Centro Geométrico:

$$\begin{aligned} Numerador = {} & (5+10+15+20+25)(0{,}65)+30\cdot 0{,}60+35\cdot 0{,}52+40\cdot 0{,}47+45\cdot 0{,}40+ \\ & +50\cdot 0{,}32+55\cdot 0{,}40+60\cdot 0{,}47+65\cdot 0{,}52+ \\ & +(70+75+80+85+90+95+100)(0{,}55) \end{aligned}$$

$$\begin{aligned} Denominador = {} & 0{,}65+0{,}65+0{,}65+0{,}65+0{,}65+0{,}60+0{,}52+0{,}47+0{,}40+0{,}32+0{,}40+ \\ & +0{,}47+0{,}52+0{,}55+0{,}55+0{,}55+0{,}55+0{,}55+0{,}55+0{,}55 \end{aligned}$$

$$COG = \frac{Numerador}{Denominador} = \frac{549{,}0}{10{,}45} = 52{,}54$$

Após isso, obtêm-se o grau de pertinência (sem considerar os limites) do conjunto Conexão Normal e do conjunto Ataque:

$$\mu(52{,}54)_{Conexão\ \ Normal} = 0{,}31 \qquad\qquad \mu(52{,}54)_{Ataque} = 0{,}35$$

Como o valor do grau de pertinência do conjunto Ataque foi maior, considera-se o

tráfego como um Ataque.

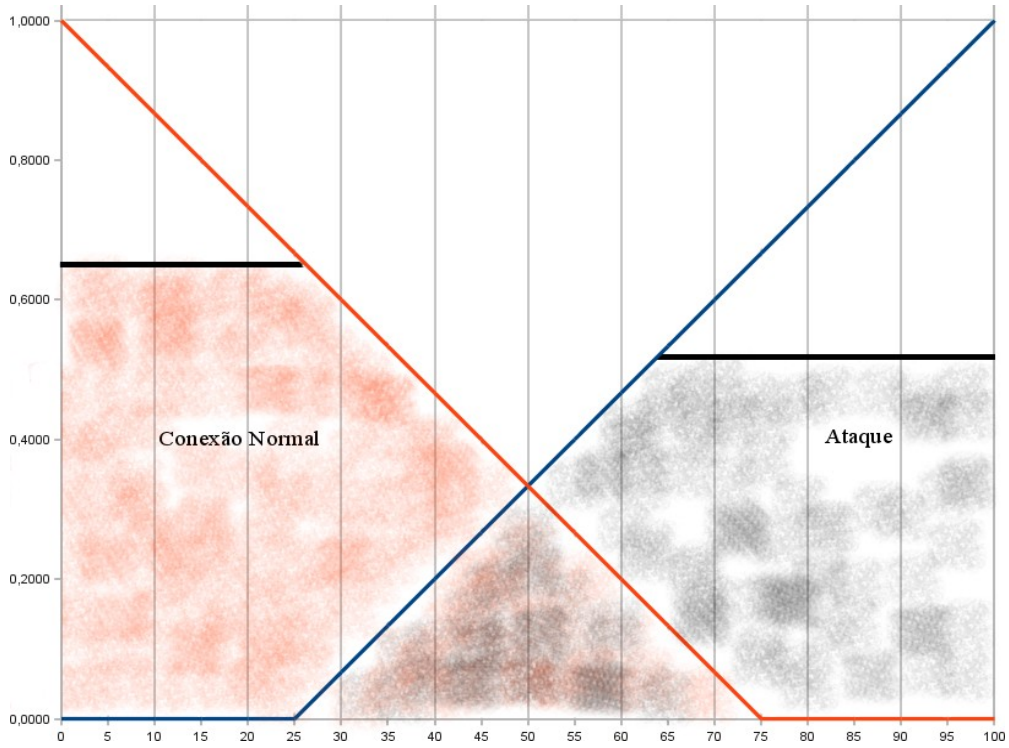


Figura 5.2.4: Exemplo da utilização do Método do Centro Geométrico

5.2.5 Comentários sobre a implementação

A linguagem de programação Python demonstrou-se eficiente pelo suporte fornecido quanto às funções matemáticas e quanto a facilidade de programação. Alguns problemas encontrados foram somente quanto a criação de novos objetos, pois como a linguagem utiliza tipagem dinâmica, na criação de "novos" objetos, porém com mesmo valor, as vezes ocorria de simplesmente efetuar a ligação a outro objeto já existente na memória. Para evitar isso, torna-se necessário sempre inicializar todos os valores internos dos atributos nos construtores da classe, além de passar por referência cópia dos objetos, ao invés de referenciá-los diretamente.

Outro problema encontrado foi quanto a utilização inicial da biblioteca PyBrain para Redes Neurais Artificiais (PYBRAIN, 2011), pois eram apresentados bons resultados de RNAs no Algoritmo Genético, mas ao implementar novamente para obter as saídas para o trabalho, a RNA não apresentava resultados compatíveis e com o mesmo aprendizado que a primeira implementação no Algoritmo Genético.

Capítulo 6

Resultados

Este capítulo apresenta os resultados e a análise dos métodos implementados, exibindo suas taxas de acerto referentes aos dados do conjunto de testes NSL-KDD (NSL-KDD, 2011). Os resultados apresentados neste capítulo foram baseados nas seguintes implementações:

- primeira Rede Neural Artificial, descrita na subseção 5.2.1;
- Algoritmo Genético e das duas Redes Neurais de menor taxa de erros geradas, descritos na subseção 5.2.2;
- Sistema Nebuloso, descrito na subseção 5.2.3;
- e, a fusão do Sistema Nebuloso com as RNAs geradas pelo Algoritmo Genético, descrito na subseção 5.2.4.

Os resultados de cada método também são comparados entre si e com o trabalho de Tavallaee (TAVALLAEE, 2009). A escolha desse trabalho deve-se ao fato do mesmo ser o qual refinou os dados do KDDCUP99 e que gerou os conjuntos de testes NSL-KDD (NSL-KDD, 2011), junto as primeiras análises dos mesmos. Os resultados apresentados seguem o modelo da Tabela 6.1. Os valores do conjunto de dados são apresentados na Tabela 6.2.

	Detectado	Não Detectado
Ataque	Verdadeiros Positivos: $\frac{Ataques\ Detectados}{Total\ de\ Ataques}$	Falso Negativos: $\frac{Ataques\ Não\ Detectados}{Total\ de\ Ataques}$
Normal	Verdadeiros Positivos: $\frac{Normais\ Detectados}{Total\ de\ Normais}$	Falso Positivos: $\frac{Normais\ Não\ Detectados}{Total\ de\ Normais}$
Total	Total de Acertos e proporção: $\frac{Total\ de\ Acertos}{Total\ de\ Conexões}$	Total de Erros e proporção: $\frac{Total\ de\ Erros}{Total\ de\ Conexões}$

Tabela 6.1: Modelo da apresentação dos resultados

	Ataque	Normal	Total de Conexões
KDDTest-21	9698	2152	11850
KDDTest+	12833	9711	22544

Tabela 6.2: Quantidade de conexões Normais e de Ataques do NSL-KDD

6.1 Primeira Rede Neural Artificial Implementada

A Primeira Rede Neural Artificial baseada nos padrões das redes MLP da Weka, porém criada pela biblioteca FANN, conforme descrita na seção 5.2.1, apresentou os seguintes resultados:

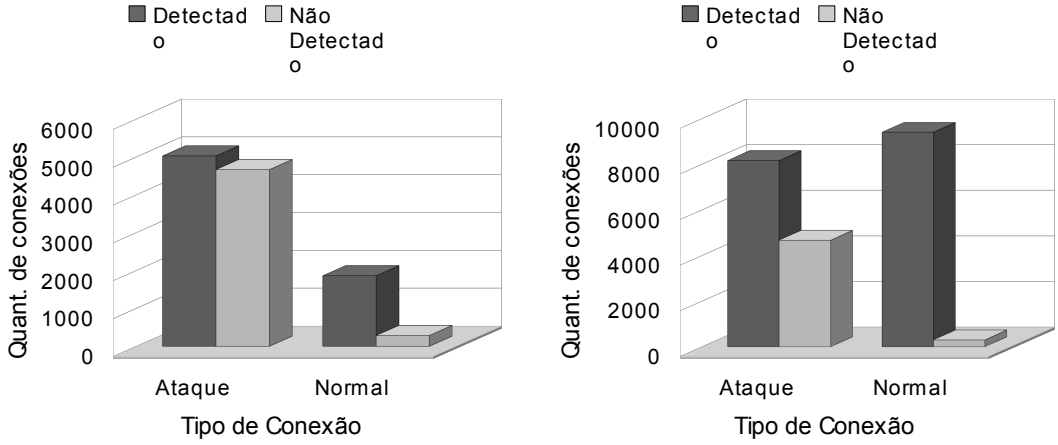
- para os dados do KDDTest-21: Tabela 6.1.1 e Figura 6.1.1a;
- para os dados do KDDTest+: Tabela 6.1.2 e Figura 6.1.1b;
- comparação entre os resultados deste trabalho e os resultados de Tavallaee: Tabela 6.1.3 e Figura 6.1.2;

	Detectado	Não Detectado
Ataque	5029 (52%)	4669 (48%)
Normal	1866 (87%)	286 (13%)
Total	6895 (58%)	4955 (42%)

Tabela 6.1.1: Resultados da primeira RNA com a base KDDTest-21

	Detectado	Não Detectado
Ataque	8164 (64%)	4669 (36%)
Normal	9408 (97%)	303 (3%)
Total	17572 (78%)	4972 (22%)

Tabela 6.1.2: Resultados da primeira RNA com a base KDDTest+



a) Base KDDTest-21 *b) Base KDDTest+*

Figura 6.1.1: Resultados da Primeira Rede Neural Artificial

	Taxa de Acerto KDDTest-21	Taxa de Acerto KDDTest+
Este Trabalho	58%	78%
Tavallaee	57%	77%

Tabela 6.1.3: Comparação dos resultados da primeira RNA e do artigo de Tavallaee

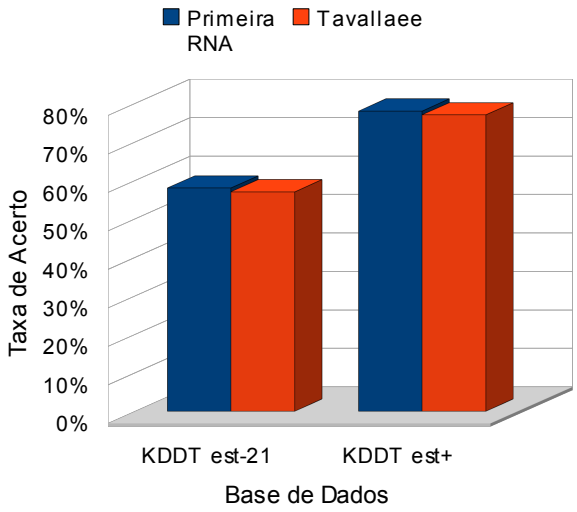


Figura 6.1.2: Resultados da Primeira RNA e do artigo de Tavallaee

Os resultados demonstraram que a biblioteca FANN é eficiente para a criação das RNAs de classificação do problema proposto, comparando-se aos resultados de Tavallaee com diferença de +1% em cada base de dados. Como obteve-se um resultado compatível com o algoritmo MLP inicial utilizado na base NSL-KDD, iniciou-se a busca por melhores resultados através da modificação da formação e do treinamento da Rede Neural Artificial.

6.2 Resultados do Algoritmo Genético

Após a evolução da população inicial do algoritmo genético, descrito na seção 5.2.2,a última população gerada foi avaliada, já que as RNAs não alcançaram o limiar de erro pedido, que era de 10%. A população final de Redes Neurais é exposta na Tabela 6.2.1. As principais características extraídas das respostas foram que:

- a melhor função de ativação para a rede oculta é a sigmóide passo a passo (*stepwise*), por encontrar-se presente em 100% das camadas ocultas da população final. Esta função é implementada pela biblioteca FANN, baseada na função Sigmóide, porém apresenta uma execução mais rápida e um pouco menos precisa (FANN, 2011);

- como função de saída, pode-se optar pelas funções de ativação sigmóide (45%) e sigmóide passo a passo (*stepwise*) (50%), por se apresentarem na maioria da população final. Um dos cromossomos finais também apresentou bom resultado com a função de ativação gaussiana para a camada de saída, sendo treinada 300 épocas utilizando 50 neurônios ocultos, algoritmo de aprendizado RPROP, taxa de aprendizado de 0,3, *momentum de* 0,5 e *Decay* de 0,3;
- as quantidades de neurônios ocultos que apresentaram melhores resultados foram de 25 (45%) e 45 (40%) neurônios. Devido a isso, aconselha-se utilizar a menor quantidade, visto que quanto mais neurônios, mais tempo o algoritmo de treinamento necessita para atualizar todos os pesos, ou seja, mais tempo a rede leva para ser treinada;
- o melhor algoritmo de treinamento encontrado foi o *Backpropagation* com atualização dos pesos após cada treino, estando presente em 90% dos cromossomos; Vale ressaltar que o algoritmo RPROP também apareceu em 10% da população, incluindo a rede cuja camada de saída era gaussiana.

RNA	Função Camada Oculta	Função Camada Saída	Qnt. Neurônios Ocultos	Algoritmo de Treinamento	Avaliação (Taxa de Erro)
1	Sig. Passo a passo	Sigmóide	45	Backpropagation	0,14310
2	Sig. Passo a passo	Sig. Passo a passo	45	Backpropagation	0,14971
3	Sig. Passo a passo	Sig. Passo a passo	25	Backpropagation	0,14984
4	Sig. Passo a passo	Sigmóide	25	Backpropagation	0,15237
5	Sig. Passo a passo	Sigmóide	25	Backpropagation	0,15534
6	Sig. Passo a passo	Sigmóide	45	Backpropagation	0,15618
7	Sig. Passo a passo	Sig. Passo a passo	25	Backpropagation	0,15654
8	Sig. Passo a passo	Sigmóide	25	Backpropagation	0,15716
9	Sig. Passo a passo	Sigmóide	45	Backpropagation	0,15778
10	Sig. Passo a passo	Sig. Passo a passo	25	Backpropagation	0,15995
11	Sig. Passo a passo	Sig. Passo a passo	15	Backpropagation	0,16000
12	Sig. Passo a passo	Sig. Passo a passo	25	Backpropagation	0,16053
13	Sig. Passo a passo	Gaussiana	50	RPROP	0,16142
14	Sig. Passo a passo	Sig. Passo a passo	25	RPROP	0,16164
15	Sig. Passo a passo	Sig. Passo a passo	45	Backpropagation	0,16182
16	Sig. Passo a passo	Sig. Passo a passo	45	Backpropagation	0,16248
17	Sig. Passo a passo	Sigmóide	50	Backpropagation	0,16248
18	Sig. Passo a passo	Sig. Passo a passo	45	Backpropagation	0,16306
19	Sig. Passo a passo	Sigmóide	45	Backpropagation	0,16328
20	Sig. Passo a passo	Sigmóide	25	Backpropagation	0,16372

Tabela 6.2.1: Resultado da última população do Algoritmo Genético – Parte I

RNA	Taxa de Aprendizado	Momentum	Decay	Qnt. Épocas de Treino	Avaliação (Taxa de Erro)
1	0.3	0.3	0.6	200	0,14310
2	0.3	0.5	0.7	300	0,14971
3	0.4	0.3	0.3	300	0,14984
4	0.3	0.3	0.3	300	0,15237
5	0.4	0.3	0.3	300	0,15534
6	0.3	0.5	0.6	200	0,15618
7	0.3	0.5	0.6	200	0,15654
8	0.3	0.5	0.3	300	0,15716
9	0.3	0.5	0.3	300	0,15778
10	0.3	0.5	0.7	300	0,15995
11	0.8	0.3	0.3	300	0,16000
12	0.3	0.3	0.6	200	0,16053
13	0.3	0.5	0.3	300	0,16142
14	0.4	0.2	0.3	300	0,16164
15	0.4	0.2	0.3	300	0,16182
16	0.3	0.5	0.3	300	0,16248
17	0.7	0.7	0.3	300	0,16248
18	0.8	0.3	0.3	300	0,16306
19	0.4	0.3	0.3	300	0,16328
20	0.3	0.5	0.3	300	0,16372

Tabela 6.2.2: Resultado da última população do Algoritmo Genético – Parte II

Vale a ressaltar que, infelizmente, assim como em outra heurística, o algoritmo genético pode encontrar e permanecer em um mínimo local, não sendo capaz de encontrar a melhor resposta. O objetivo da mutação é sair dos mínimos locais para encontrar uma melhor solução, mas devido a ser uma troca aleatória do gene, não se pode garantir a escolha de um melhor resultado.

Com os resultados dos cromossomos da última população, as redes foram implementadas usando a biblioteca FANN. Devido a escolha inicial dos pesos das redes neurais ser aleatória, as RNAs apresentaram uma avaliação diferente quando são

implementadas e treinadas posteriormente. Mesmo assim, não retira do algoritmo genético o crédito pela formação de uma boa população, pois ele manteve as redes que conseguiram alcançar os melhores resultados, servindo de indicação para a implementação posterior.

Das implementações baseadas nos resultados do algoritmo genético, foram extraídas as duas redes com menor taxa de erros (redes 1 e 18). A primeira rede implementada apresentou os seguintes resultados:

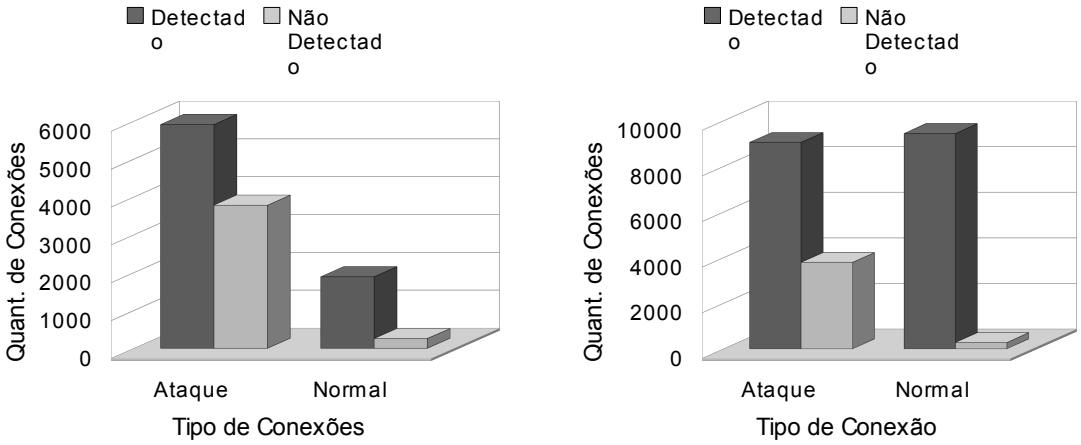
- para os dados do KDDTest-21: Tabela 6.2.3 e Figura 6.2.1a;
- para os dados do KDDTest+: Tabela 6.2.4 e Figura 6.2.1b;
- comparação entre os resultados deste trabalho e os resultados de Tavallaee: Tabela 6.2.5 e Figura 6.2.2;

	Detectado	Não Detectado
Ataque	5914 (61%)	3784 (39%)
Normal	1894 (88%)	258 (12%)
Total	7808 (66%)	4042 (34%)

Tabela 6.2.3: Resultados da RNA (I) gerada pelo Algoritmo Genético com a base KDDTest-21

	Detectado	Não Detectado
Ataque	9048 (71%)	3785 (29%)
Normal	9436 (97%)	275 (3%)
Total	18484 (82%)	4060 (18%)

Tabela 6.2.4: Resultados da RNA (I) gerada pelo Algoritmo Genético com a base KDDTest+



a) Base KDDTest-21 *b) Base KDDTest+*

Figura 6.2.1: Resultados da RNA (I) gerada pelo Algoritmo Genético

	Taxa de Acerto KDDTest-21	Taxa de Acerto KDDTest+
Alg.Gen. RNA I	66%	82%
Tavallaee	57%	77%

Tabela 6.2.5: Comparação dos resultados da RNA (I) gerada pelo Algoritmo Genético e do artigo de Tavallaee

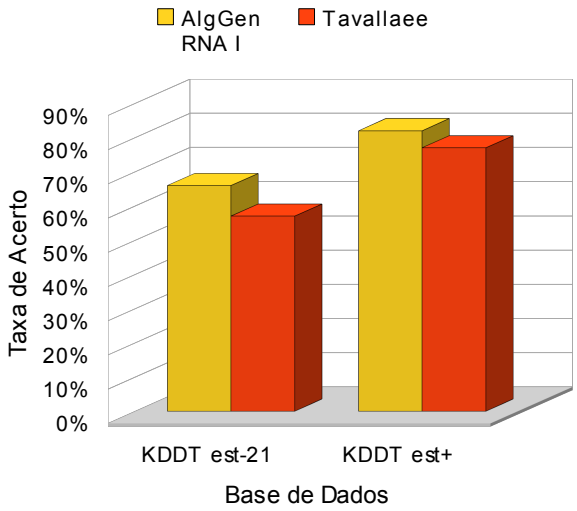


Figura 6.2.2: Resultados da RNA (I) gerada pelo Algoritmo Genético e do artigo de Tavallaee

A comparação da primeira Rede Neural com esta implementação e a Rede MLP

de Tavallaee pode ser vista na Figura 6.2.3. A RNA I encontrada pelo Algoritmo Genético apresentou melhores resultados na taxa de acerto em relação à primeira RNA, com um aumento na taxa de acerto de 13,8% para o KDDTest-21 e de 5,1% para o KDDTest+. Comparando com os resultados do trabalho de Tavallaee, foi observado um ganho na taxa de acerto de 15,8% e 6,5%, para as bases de dados KDDTest-21 e KDDTest+, respectivamente.

Comparando-se com a primeira Rede Neural implementada, teve melhoria em todos os índices, com um acerto maior na classificação das conexões, validando melhor os ataques, com ganho de 17,6% para o KDDTest-21 e de 10,8% para o KDDTest+, e também o tráfego normal, com ganho de 1,5% para o KDDTest-21 e com leve variação de 0.3% para o KDDTest+. Apresentando também uma melhoria para o tráfego detectado de forma errada, deixando de considerar 19,0% de ataques como tráfego normal (Falso Negativos) e 9,8% de conexões normais como ataque (Falso Positivos) para o KDDTest-21 e de 18,9%(Falso Negativos) e 9,2%(Falso Positivos) para o KDDTest+.

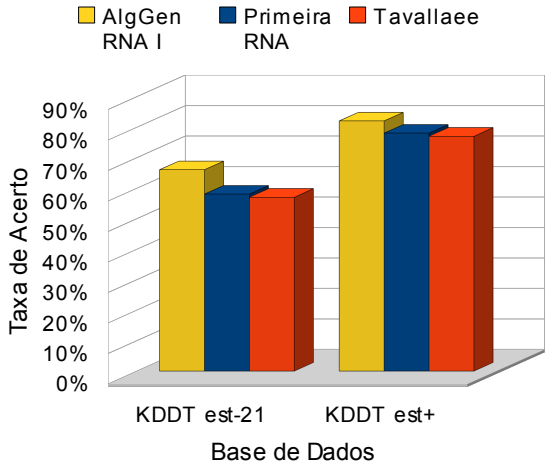


Figura 6.2.3: Taxa de acerto das Redes Neurais implementadas

Quanto a Segunda Rede Neural Artificial, apresentou resultados:

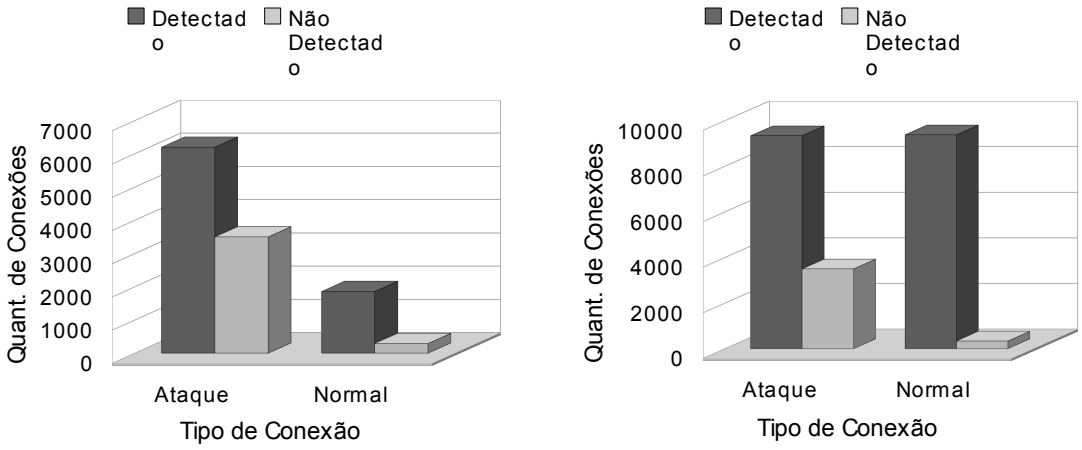
- para os dados do KDDTest-21: Tabela 6.2.6 e Figura 6.2.4a;
- para os dados do KDDTest+: Tabela 6.2.7 e Figura 6.2.4b;
- comparação entre os resultados deste trabalho e os resultados de Tavallaee: Tabela 6.2.8 e Figura 6.2.5;

	Detectado	Não Detectado
Ataque	6199 (64%)	3499 (36%)
Normal	1859 (86%)	293 (14%)
Total	8058 (68%)	3792 (32%)

Tabela 6.2.6: Resultados da RNA (II) gerada pelo Algoritmo Genético com a base KDDTest-21

	Detectado	Não Detectado
Ataque	9334 (73%)	3499 (27%)
Normal	9372 (97%)	339 (3%)
Total	18706 (83%)	3838 (17%)

Tabela 6.2.7: Resultados da RNA (II) gerada pelo Algoritmo Genético com a base KDDTest+



a) Base KDDTest-21 *b) Base KDDTest+*

Figura 6.2.4: Resultados da RNA (II) gerada pelo Algoritmo Genético

	Taxa de Acerto KDDTest-21	Taxa de Acerto KDDTest+
Alg.Gen. RNA II	68%	83%
Tavallaee	57%	77%

Tabela 6.2.8: Comparação dos resultados da RNA (II) gerada pelo Algoritmo Genético e do artigo de Tavallaee

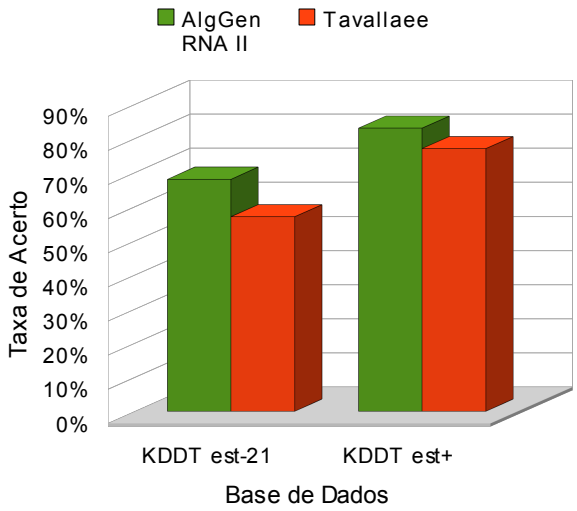


Figura 6.2.5: Resultados da RNA (II) gerada pelo Algoritmo Genético e do artigo de Tavallaee

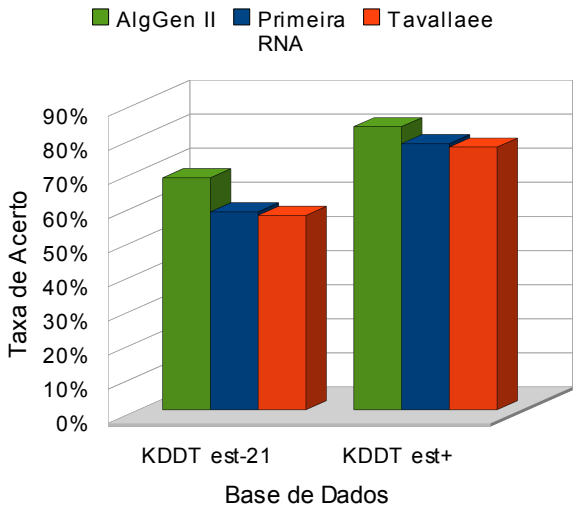


Figura 6.2.6: Taxa de acerto das Redes Neurais implementadas

A comparação da primeira Rede Neural, com esta implementação e a Rede MLP

de Tavallaee pode ser vista na Figura 6.2.6. A RNA II encontrada pelo Algoritmo Genético apresentou melhores resultados na taxa de acerto quando comparada a primeira RNA, com aumento na taxa de acerto de 17,2% para o KDDTest-21 e de 6,4% para o KDDTest+. Na comparação com o trabalho de Tavallaee, o ganho na taxa de acerto foi de 19,3% para o KDDTest-21 e de 7,8% para o KDDTest+. Comparando-se com a primeira Rede Neural implementada, observou-se uma melhoria de 23,3% na identificação de Verdadeiro Positivos e de 25,1% na identificação de Falso Negativos para o KDDTest-21. Para a base de dados KDDTest+, obteve-se um ganho de 14,3% na identificação Verdadeiro Positivos e de 25,1% na identificação de Falso Negativos. Porém, houve um aumento na detecção de Falso Positivos em 2% para o KDDTest-21 e de 11,9% para o KDDTest+.

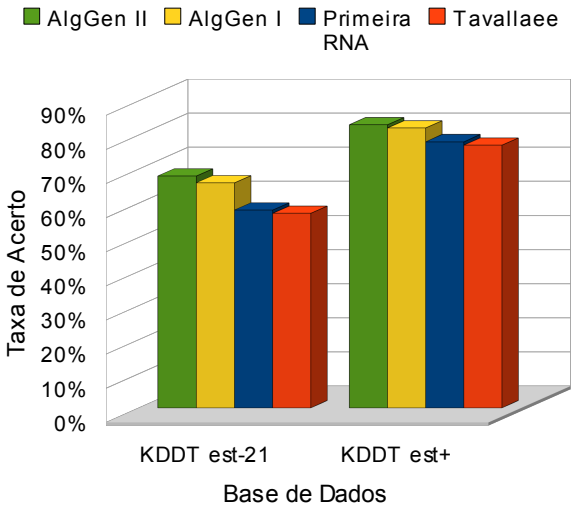


Figura 6.2.7: Taxa de acerto das Redes Neurais implementadas

Os resultados das Redes Neurais Artificiais encontradas pelo algoritmo genético são melhores que os resultados da primeira RNA, confirmando que o algoritmo genético conseguiu aprimorar o sistema existente, fornecendo bons resultados e melhores taxas de Falso Positivos e/ou Falso Negativos. A melhoria deve-se a troca das funções de ativação, que foram substituídas por funções mais adequadas ao conjunto de entrada, diferente do algoritmo de treinamento que foi mantido, mas que apresentou uma taxa diferente de *decay* e que obteve melhores resultados com as novas funções de ativação.

As Redes Neurais Artificiais obtidas pelo Algoritmo Genético apresentaram os

melhores resultados dentre todos sistemas implementados neste trabalho, devido ao fato de apresentarem não somente a taxa de acerto maior, quanto também de apresentarem taxa maior nos Verdadeiro Positivos e nos Verdadeiro Negativos e taxa menor nos Falso Positivos e nos Falso Negativos, enquanto outras implementações melhoram a taxa de acerto, mas pioram os Falso Positivos e/ou Falso Negativos em sua classificação.

6.3 Resultados do Sistema Nebuloso

O sistema nebuloso implementado, descrito na seção 5.2.3, apresentou os seguintes resultados:

- para os dados do KDDTest-21: Tabela 6.3.1 e Figura 6.3.1a;
- para os dados do KDDTest+: Tabela 6.3.2 e Figura 6.3.1b;
- comparação entre os resultados deste trabalho e os resultados de Tavallaee: Tabela 6.3.3 e Figura 6.3.2;

	Detectado	Não Detectado
Ataque	5377 (55%)	4321 (45%)
Normal	1709 (79%)	443 (21%)
Total	7086 (60%)	4764 (40%)

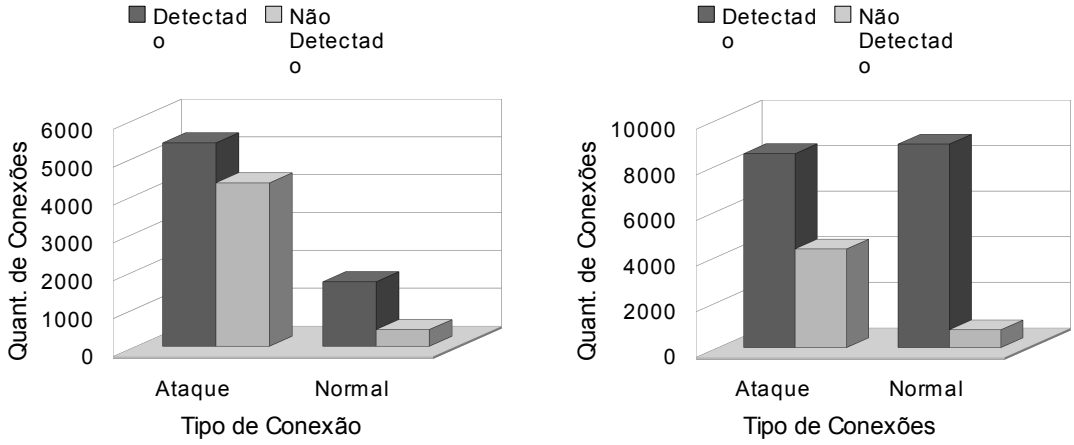
Tabela 6.3.1: Resultados da Lógica Nebulosa com a base KDDTest-21

	Detectado	Não Detectado
Ataque	8506 (66%)	4327 (34%)
Normal	8929 (92%)	782 (8%)
Total	17435 (77%)	5109 (23%)

Tabela 6.3.2: Resultados da Lógica Nebulosa com a base KDDTest+

	Taxa de acerto KDDTest-21	Taxa de Acerto KDDTest+
Lógica Nebulosa	60%	77%
Tavallaee	57%	77%

Tabela 6.3.3: Comparação dos resultados da Lógica Nebulosa e do artigo de Tavallaee



a) Base KDDTest-21 *b) Base KDDTest+*

Figura 6.3.1: Resultados da Lógica Nebulosa

O Sistema Nebuloso apresentou taxa de acerto similar à primeira Rede Neural e à implementação de Tavallaee no conjunto de dados KDDTest+ e uma taxa melhor de acerto de 5,3% na base KDDTest-21 em relação ao trabalho de Tavallaee e 3,4% em relação a primeira Rede Neural implementada. Comparando-se com a primeira RNA implementada, apresentou melhorias de na detecção de ataques, de 6,9% para o KDDTest-21 e 4,1% para o KDDTest+, e no erro de identificação de ataques, de 7,5% no KDDTest-21 e de 7,2% no KDDTest+, porém aumentou a identificação de tráfego normal como ataque em 54,9% para o KDDTest-21 e em 158,1% para o KDDTest+, além de diminuir a quantidade de conexões normais identificadas em 8% e 5%, para o KDDTest-21 e KDDTest+21, respectivamente. A comparação dentre os métodos implementados até o momento podem ser vistas na Figura 6.3.3.

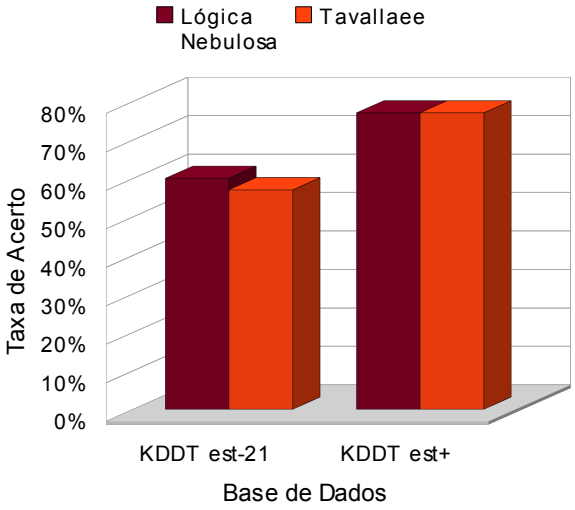


Figura 6.3.2: Resultados da Lógica Nebulosa e do artigo de Tavallaee

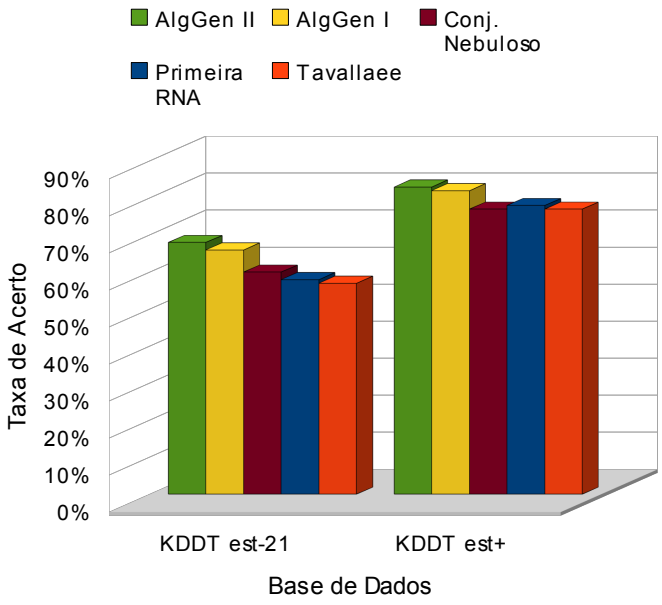


Figura 6.3.3: Taxa de acerto da Lógica Nebulosa

6.4 Resultados do sistema Neuro-Fuzzy

Foram implementados dois sistemas Neuro-Fuzzy, conforme descritos na seção 5.2.4, sendo um para cada RNA de menor taxa de erros encontrada pelo Algoritmo Genético. O sistema Neuro-Fuzzy I implementado apresentou os seguintes resultados:

- para os dados do KDDTest-21: Tabela 6.4.1 e Figura 6.4.1a;
- para os dados do KDDTest+: Tabela 6.4.2 e Figura 6.4.1b;

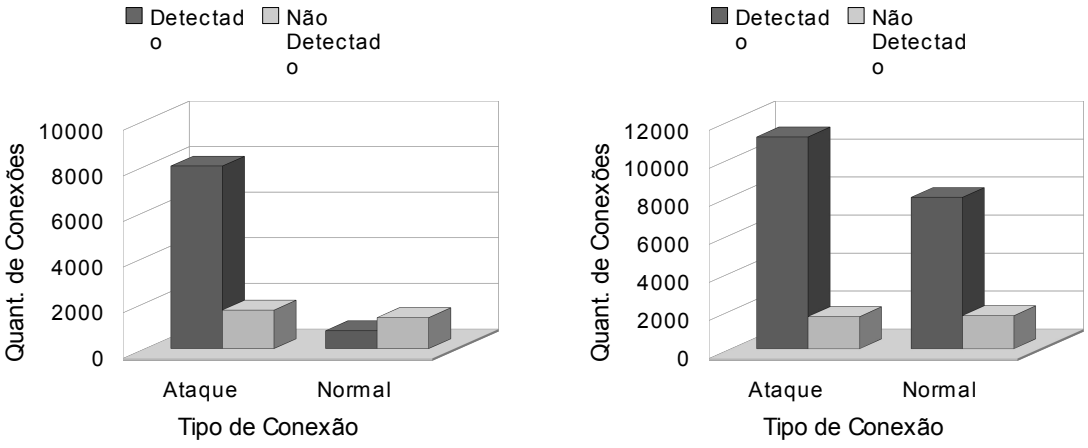
– comparação entre os resultados deste trabalho e os resultados de Tavallaee: Tabela 6.4.3 e Figura 6.4.2;

	Detectado	Não Detectado
Ataque	8007 (83%)	1691 (17%)
Normal	788 (37%)	1364 (63%)
Total	8795 (74%)	3055 (26%)

Tabela 6.4.1: Resultados da Neuro-Fuzzy (I) com a base KDDTest-21

	Detectado	Não Detectado
Ataque	11142 (87%)	1691 (13%)
Normal	7967 (82%)	1744 (18%)
Total	19109 (85%)	3435 (15%)

Tabela 6.4.2: Resultados da Neuro-Fuzzy (I) com a base KDDTest+



a) Base KDDTest[21] *b) Base KDDTest+*

Figura 6.4.1: Resultados do Sistema Neuro-Fuzzy I

	Taxa de Acerto KDDTest-21	Taxa de Acerto KDDTest+
Neuro-Fuzzy I	74%	85%
Tavallaee	57%	77%

Tabela 6.4.3: Comparação dos resultados da Neuro-Fuzzy (I) e do artigo de Tavallaee

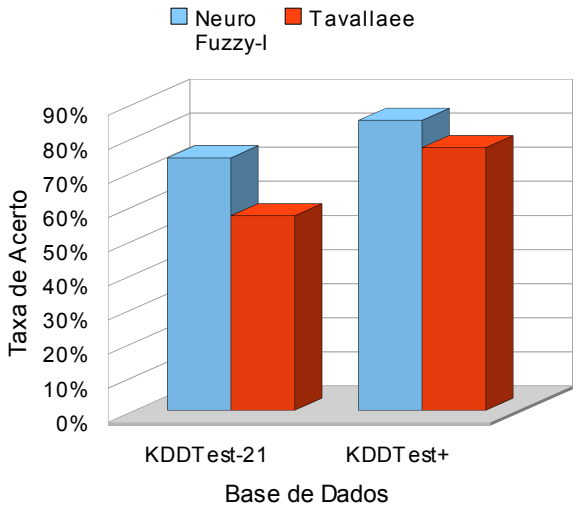


Figura 6.4.2: Resultados da Neuro-Fuzzy (I) e do artigo de Tavallaee

O Neuro-Fuzzy I apresentou uma taxa de acerto melhor em relação à primeira Rede Neural implementada, sendo uma melhoria de 27,6% na base KDDTest-21 e de 9,0% na base KDDTest+. Quanto a implementação de Tavallaee, também apresentou melhorias, sendo 29,9% na base KDDTest-21 e 10,4% na base KDDTest+. Comparando-se com a primeira Rede Neural implementada, identificou de forma melhor 59,2% dos ataques (Verdadeiro Positivos) no KDDTest-21 e 36,5% no KDDTest+. Sua taxa tráfego normal identificado como ataque (Falso Negativos) também melhorou 63,8% para o KDDTest-21 e 63,8%(FN) para o KDDTest+. Porém, apresentou uma baixa ao identificar de forma correta o tráfego normal (Verdadeiros Negativos), de 57,8% para o KDDTest-21 e de 15,3% para o KDDTest+ e um aumento significativo na identificação de tráfegp normal como ataque (Falso Positivos), sendo 377,0% para o KDDTest-21 e 475,6% para o KDDTest+.

A combinação do sistema nebuloso e da Rede Neural trouxe melhorias na taxa de acerto, porém a identificação errada do tráfego normal aumentou devido a falta de regras

nebulosas para conexões normais, pois necessitaria de valores mais altos em uma das regras de ataque ou em uma das regras de tráfego normal para uma defuzzificação melhor, visto que o cálculo inicial de seu resultado vem da combinação desses valores com regras OU. A comparação entre os métodos implementados até o momento pode ser vista na Tabela 6.4.4 e na Figura 6.4.3.

	KDDTest-21	KDDTest+
Neuro-Fuzzy I	74%	85%
AlgGen II	68%	83%
AlgGen I	66%	82%
Conj. Nebuloso	60%	77%
Primeira RNA	58%	78%
Tavallaee	57%	77%

Tabela 6.4.4: Resultados dos Métodos implementados e do artigo de Tavallaee

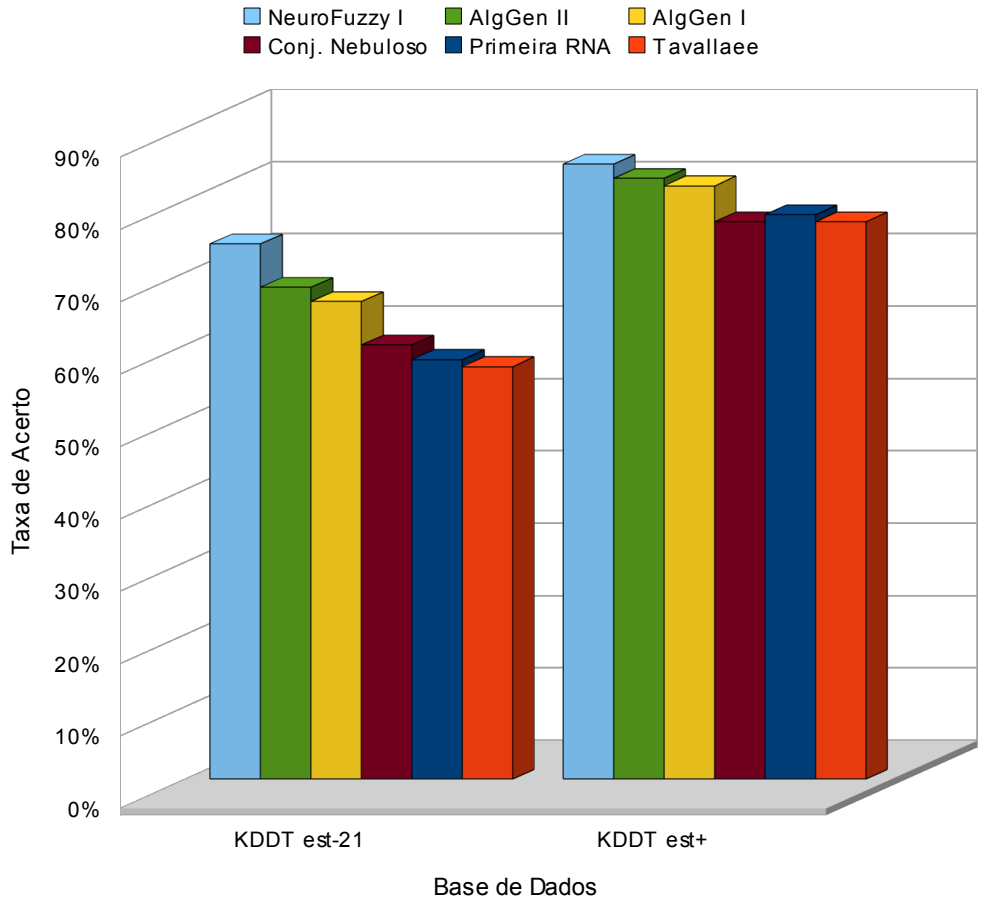


Figura 6.4.3: Resultados dos Métodos implementados e do artigo de Tavallaee

O sistema Neuro-Fuzzy II implementado apresentou os seguintes resultados:

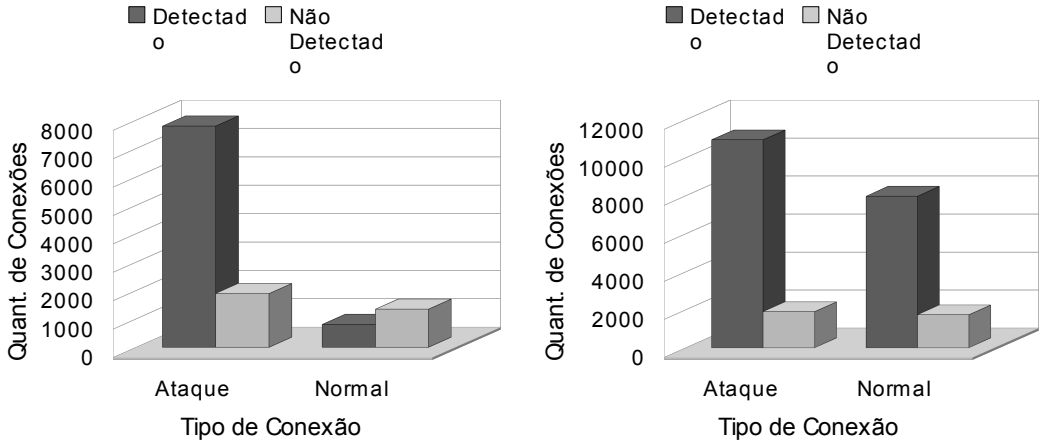
- para os dados do KDDTest-21: Tabela 6.4.5 e Figura 6.4.4a;
- para os dados do KDDTest+: Tabela 6.4.6 e Figura 6.4.4b;
- comparação entre os resultados deste trabalho e os resultados de Tavallaee: Tabela 6.4.7 e Figura 6.4.5;

	Detectado	Não Detectado
Ataque	7798 (80%)	1900 (20%)
Normal	807 (38%)	1345 (62%)
Total	8605 (73%)	3245 (27%)

Tabela 6.4.5: Resultados da Neuro-Fuzzy (II) com a base KDDTest-21

	Detectado	Não Detectado
Ataque	10933 (85%)	1900 (15%)
Normal	7973 (82%)	1738 (18%)
Total	18906 (84%)	3638 (16%)

Tabela 6.4.6: Resultados da Neuro-Fuzzy (II) com a base KDDTest+



a) Base KDDTest[21] *b) Base KDDTest+*

Figura 6.4.4: Resultados da Neuro-Fuzzy (II)

	Taxa de Acerto KDDTest-21	Taxa de Acerto KDDTest+
Este Trabalho	73%	84%
Tavallaee	57%	77%

Tabela 6.4.7: Comparação dos resultados da Neuro-Fuzzy (II) e do artigo de Tavallaee

	KDDTest-21	KDDTest+
Neuro-Fuzzy I	74%	85%
Neuro-Fuzzy II	73%	84%
AlgGen II	68%	83%
AlgGen I	66%	82%
Sist. Nebuloso	60%	77%
Primeira RNA	58%	78%
Tavallaee	57%	77,00%

Tabela 6.4.8: Resultado dos métodos implementados e do artigo de Tavallaee

O Neuro-Fuzzy II também apresentou taxa de acerto melhor em relação a primeira Rede Neural implementada e em relação ao trablaho de Tavallaee, porém, como a Rede Neural utilizada apresentava uma taxa menor de acerto, essa implementação apresentou taxas menores que o primeiro sistema Neuro-Fuzzy. Em realação a primeira Rede Neural implementada, apresentou 25,9% de melhoria no KDDTest-21 e 7,7% de melhoria no KDDTest+ e quanto a implementação de Tavallaee, apresentou 28,1% de melhoria no KDDTest-21 e 9,1% de melhoria no KDDTest+.

Quanto as taxas de detecção, comparando-se com a primeira Rede Neural implementada, teve melhoria em identificar os ataques (Verdadeiro Positivos) de 55,1% no KDDTest-21 e 33,9% no KDDTest+, apresentando também melhorias ao não identificar ataques como tráfego normal (Falso Negativos), sendo 59,3% para o KDDTest-21 e 59,3% para o KDDTest+. Porém, apresentou uma baixa ao identificar o tráfego normal (Verdadeiros Negativos), sendo 56,8% para o KDDTest-21 e 15,3% para o KDDTest+, além de apresentar um aumento nos alertas falsos de ataque (Falso Positivos) de 370,3% para o KDDTest-21 e de 473,6% para o KDDTest+. A comparação entre os todos os métodos implementados até o momento pode ser vista na Tabela 6.4.8 e na Figura 6.4.6.

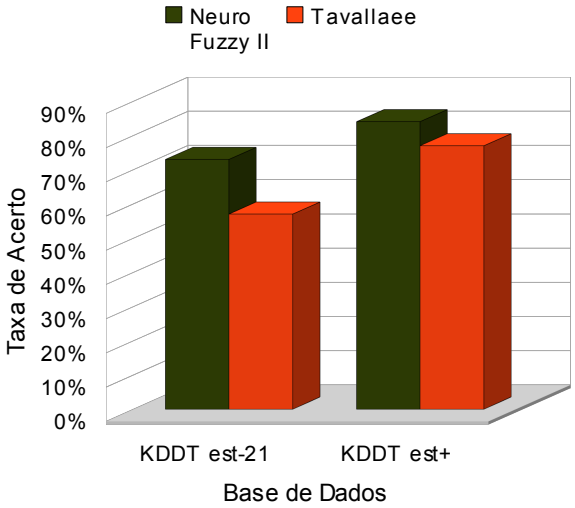


Figura 6.4.5: Resultados da Neuro-Fuzzy (II) e do artigo de Tavallaee

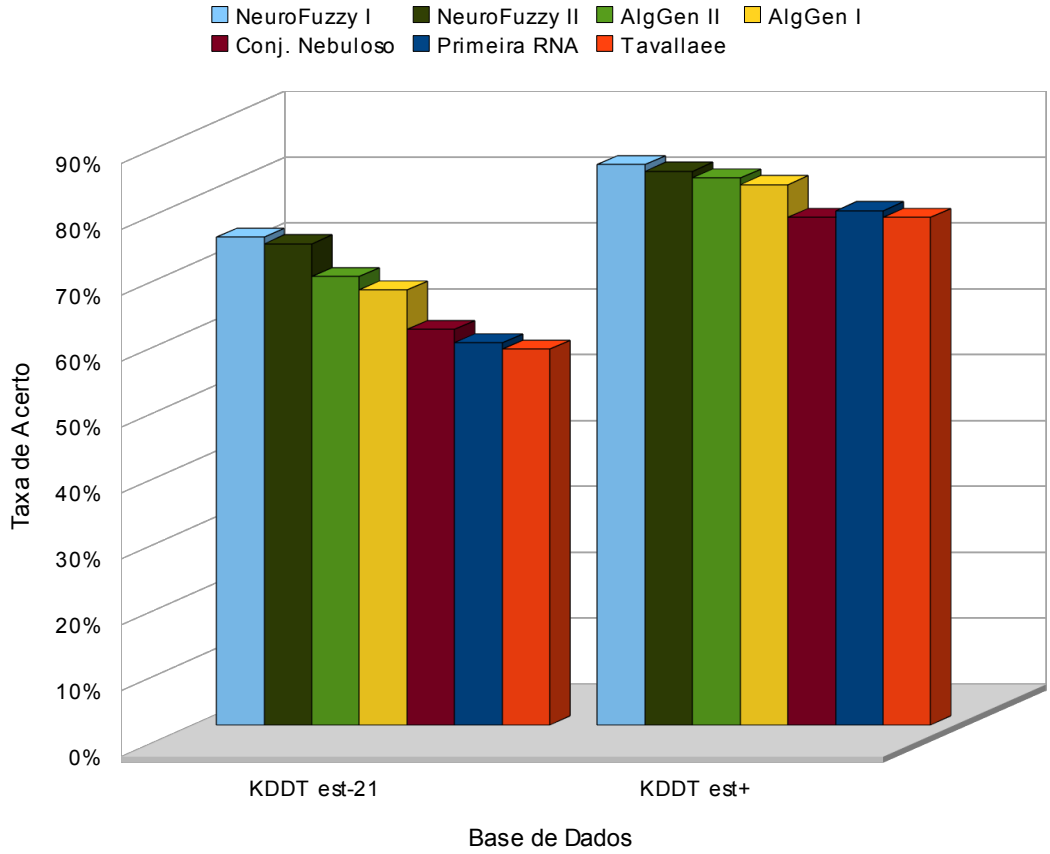


Figura 6.4.6: Taxa de Acerto dos métodos implementados e do artigo de Tavallaee

Capítulo 7

Conclusões e Trabalhos Futuros

7.1 Conclusão

Este trabalho apresentou a Detecção de Intrusão com técnicas de Inteligência Artificial (IA). Para isso, foi utilizada uma nova tendência nos métodos de IA: os Sistemas Inteligentes Híbridos, formados pela combinação de mais de uma técnica de Inteligência Artificial. Para tal, Algoritmos Genéticos foram utilizados na formulação de Redes Neurais Artificiais, que então foram combinadas com um Sistema Lógico Nebuloso. A combinação dos métodos de IA foi utilizada para a classificação de conexões normais e de ataques, baseando-se nas anomalias pertencentes à base de dados NSL-KDD, sendo esta uma base nova, reformulada do KDDCUP99, que apresentava diversos erros descritos em (TAVALLAEE, 2009).

Foi apresentado nos resultados (Capítulo 6) que todos os sistemas implementados conseguiram encontrar uma taxa de acerto melhor do que o artigo de Tavallaee, no qual as duas Redes Neurais Artificiais implementadas com os resultados do Algoritmo Genético conseguiram aumentar a classificação correta em 17,6% e 7,2% das amostras de testes, enquanto o Sistema Nebuloso conseguiu um ganho na classificação correta de 5,3% da segunda amostra. Os sistemas Neuro-Fuzzy também apresentaram taxas de acertos mais altas, com um ganho de 29,9% e 10,4% utilizando a Rede Neural Artificial de melhor classificação e com um ganho de 28,1% e 9,1% utilizando a segunda Rede

Neural Artificial de melhor classificação.

O trabalho de Tavallaee não apresentou os resultados detalhados das taxas de detecção de Falso Positivos e Falso negativos. Na análise comparativa das implementações realizadas, observou-se que alguns sistemas que tiveram resultados melhores (maior acerto) no aumento da detecção de Verdadeiros Positivos e Verdadeiros Negativos, apresentaram um pior desempenho relativo à diminuição de detecção de Falsos Positivos e Falsos Negativos. Os sistemas que utilizaram as Redes Neurais encontrados pelo Algoritmo Genético foram os que apresentaram o melhor desempenho global, pois obtiveram um ganho significativo (da ordem de 13%) na detecção de Verdadeiros Positivos e Verdadeiros Negativos e de Falsos Negativos (da ordem de 25%), sem piorar significativamente (7% em média) a detecção de Falsos Positivos.

Tais resultados demonstram que os Sistemas Inteligentes podem ajudar na melhoria dos métodos atuais para Detecção de Intrusão e que os Sistemas Inteligentes Híbridos realmente podem fornecer um modelo melhor dentre os modelos existentes, por conseguir unir as qualidades de dois ou mais Sistemas Inteligentes diferentes.

Vale ressaltar que os Sistemas Nebulosos apresentaram-se capazes de trabalhar com detecções, porém sua melhor taxa de classificação depende de regras mais especializadas. Na implementação, por exemplo, o número de Falso Positivos foi maior devido a falta de regras especialistas para conexões normais, contando-se que se tratavam de seis regras para ataques e somente uma regra para conexões normais. Devido a isso, a junção com os sistemas nebulosos não apresentaram resultados satisfatórios para os Falso Positivos, interferindo também nos resultados dos Verdadeiro Positivos.

Sistemas de Detecção de Intrusão com Sistemas Inteligentes podem ser implementados devido ao tempo de execução do algoritmo ser rápido e não sobrecarregar o sistema operacional instalado, por efetuar somente contas limitadas de soma e multiplicação, como por exemplo nas redes neurais, onde o a complexidade é igual a soma da multiplicação do valor de seus neurônios pelos seus pesos respectivos: $\sum \theta_i . w_{ji}$, onde θ representa o neurônio e w representa o peso. O maior consumo do processamento do sistema é somente no tempo de treinamento, mas após o sistema estar treinado, sua execução é bem mais rápida.

A existência de novos métodos como os sistemas imunológicos artificiais e os sistemas inteligentes híbridos, ainda pouco abordados abrem uma grande perspectiva de obtenção de bons resultados. Também seria de grande interesse a aplicação das pesquisas teóricas em sistemas reais para a análise mais verídica dos resultados e os possíveis testes pela comunidade, utilizando amostras de tráfego reais. A exposição desses campos e destes resultados visa despertar, entre os pesquisadores da área, interesse para a melhoria de SDIs com IA.

7.2 Trabalhos Futuros

Diante dos resultados apresentados, uma das possibilidades que se coloca é a implementação real de Sistemas de Detecção de Intrusão baseados em Inteligência Artificial, tendo de ser capazes de capturar informações sobre a conexão compatíveis com os dados do NSL-KDD. Qualquer sistema inteligente, com bons resultados sobre os dados do NSL-KDD poderia então ser testado em uma rede real, para assim realmente comprovar sua eficiência. Sistemas de Detecção de Intrusão livres poderiam ser modificados para avaliar o tráfego capturado, ou poder-se-ia fazer uso da biblioteca *libpcap*, utilizada tanto na captura inicial dos pacotes do KDDCUP, quanto em diversos aplicativos, como o Snort (SNORT, 2011), o TcpDump (TCPDUMP, 2011), WireShark (WIRESHARK, 2011), etc. Um exemplo da utilização dessa biblioteca pode ser encontrado no "Anexo VI – Exemplo da utilização do LibPcap e Python".

Além de implementações reais de sistemas de detecção com sistemas inteligentes, como trabalhos futuros, destacam-se:

- a implementação de mais métodos de IA para detecção de intrusões;
- a criação um gerador de regras nebulosas sobre a base de testes através de programação genética;
- a criação e utilização de outras bases de testes;
- a criação de um classificador para cada tipo de ataque, utilizando o método inteligente que melhor apresente respostas;

– a criação de um sistema que utilize assinaturas nebulosas, ao invés do tipo de assinaturas utilizadas atualmente.

Referências Bibliográficas

ANDERSON, James P. **Computer Secutiry Threat Monitoring and Surveillance, Technical Report**. James P. Anderson Co. Fort Washington, PA, abr. 1980.

ANÔNIMO. **Segurança Máxima para Linux**. Rio de Janeiro: Editora Campus, 2000.

ARTERO, Almir Olivette. **Inteligência Artificial:** teoria e prática. São Paulo: Editora Livraria da Física, 2009.

BEALE, Jay et al. **Snort 2.1 Intrusion Detection**, Second Edition. Syngress Publishing, 2004.

BEJTLICH, Richard. **The Tao of Network Security Monitoring Beyond Intrusion Detection**. Addison Wesley, 2004.

BITTENCOURT, Guilherme. **Inteligência Artificial:** ferramentas e teorias. 3 ed. Florianópolis: Editora da UFSC, 2006.

BRACKMANN, Christian et al. **Uma abordagem para Gerenciamento Distribuído de Redes Utilizando Agentes Móveis**. Escola Regional de Redes de Computadores, Passo Fundo, n. 4, setembro 2006.

BRITT, Winard et al. **Computer Defense Using Artificial Intelligence**. In: Proceedings of the 2007 Spring Simulation Multiconference, Volume 3, Norfolk, Virginia, Março, 2007. Spring Simulation Multiconference. Society for Computer Simulation International, San Diego, CA, 2007.

CANSIAN, Adriano Mauro. **Detecção de Intrusos em Redes de Computadores**. Tese (Doutorado em Física Aplicada), Instituto de Física de São Carlos, São Carlos, 1997.

CASTANHEIRA, Luciana Gomes. **Aplicação de Técnicas de Mineração de Dados em Problemas de Classificação de Padrões**. Belo Horizonte: UFMG, 2008. 95 p. Dissertação (Mestrado). Curso de Engenharia Elétrica, Universidade Federal de Minas Gerais, Belo Horizonte, 2008.

CERT. **CERT Statistics (Historical)**. Disponível em: <http://www.cert.org/stats/>. Acesso em: 21 de julho de 2010.

CERT.BR. **Cartilha de segurança para Internet**. Disponível em: <http://cartilha.cert.br/download>. Acesso em: 20 de Julho de 2010.

COPPIN, Ben. **Inteligência Artificial**. Rio de Janeiro: LTC, 2010.

COSTA, Nilson Santos. **Proteção de Sistemas Elétricos Considerando Aspectos de Segurança da Rede de Comunicação** . Tese (Doutorado). Curso de Engenharia Elétrica, Escola de Engenharia de São Carlos, São Carlos, Abril, 2007.

DASGUPTA, Jonathan G. D. **Evolving Fuzzy Classifiers for Intrusion Detection**. Proceddings of the 2002 IEEE. Workshop on Information Assurance. Estados Unidos, Junho 2001.

DEBAR, Hervé et al. **A Neural Network Component for an Intrusion Detection System**. IEEE Computer Society Symposium on Research Security and Privacy, Oakland, CA, pag. 240–250, Maio 1992.

FAHLMAN, S. **An Empirical Study of Learning Speed in Back-Propagation Networks**. Technical report, School of Computer Science, Carnegie Mellon University, (CMU-CS-88-162), Setembro, 1988.

FANN. **Fast Artificial Neural Network Library**. Disponível em: <http://sourceforge.net/ projects/fann/>. Acesso em: Janeiro, 2011.

FRASER, B. **RFC 2196 – Site Security Handbook**. Disponível em: <http://www.faqs.org/rfcs/rfc2196.html>. Setembro, 1997. Acesso em: Fevereiro, 2010.

FUZZPY. **Fuzzy Logic in Python**. Disponível em: <http://code.google.com/p/fuzzpy/>. Acesso em: Janeiro, 2011.

HAYKIN, Simon. **Redes Neurais:** Princípios e Prática. Porto Alegre, Rio Grande do Sul: Bookman, 2001.

HOUARI, Nora; FAR, B. **Application of Intelligent Agent Technology for Knowledge Management Integration**. Proceedings of IEEE ICCI 2004, p. 240-249, 2004

JIANG J., ZHANG C., KAME M.. **RBF-based real-time hierarchical intrusion detection systems**. Proceedings of the International Joint Conference on Neural Networks (IJCNN '03), volume 2, p. 1512–1516, Portland, OR, USA, Julho 2003.

KENDALL, Kristopher. **A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems**. Massachusetts: MIT, 1999. 124 p. Dissertação (Mestrado). Curso de Ciência da Computação, Massachusetts Institute of Technology, Massachusetts, 1999.

KEMPFER, Rafael. **Utilização de Agentes Autônomos Na Gerência de Redes**: ACD, Um Agente Coletor de Dados. Santa Maria: UFSM, 2006. 68 p. Dissertação (Graduação). Curso de Ciência da Computação, Universidade Federal de Santa Maria, Santa Maria, 2006.

KDDCUP99. **KDD Cup 1999 Data**. Disponível em: <http://kdd.ics.uci.edu/databases/ kddcup99/kddcup99.html>. Outubro, 1999. Acesso em: 10 out. 2010.

KIM, Jungwon, et al. **Immune system approaches to intrusion detection – a review**. Natural Computing: Biomedical and Life Sciences, Springer Netherlands, p. 413-466, volume 6, ed. 4, dez. 2007.

KIZZA, Joseph Migga. **Computer Network Security**. Springer Science+Business Media, Inc., 2005.

KUMAR, Sandeep. **Classification and Detection of Computer Intrusions**. Tese (Doutorado), Department of Computer Sciences, Purdue University, Agosto, 1995 .

MAIA, Roberto Bomeny. **Detecção da Intrusão Utilizando Classificação Bayesiana**. Rio de Janeiro: COPPE/UFRJ, 2005. 139 p. Dissertação (Mestrado). Curso de Ciências em Engenharia Elétrica, Universidade Federal do Rio de Janeiro – COPPE, Rio de Janeiro, 2005.

MAIOR, A. O. B.; SANTOS, F. A.; LACQUA S. C. **Gestão de Segurança da Informação**. Dissertação (Graduação). Curso de Sistemas de Informação, Faculdade Gennari & Peartree, Pederneiras, São Paulo, 2006.

NSL-KDD. **The NSL-KDD Data Set**. Disponível em: <http://www.iscx.ca/NSL-KDD/>. Acesso em: Janeiro, 2011.

OPENSOURCE. **The Open Souce Definition**. Disponível em: <http://www.opensource.org/ docs/osd>. Acesso em: Janeiro, 2011.

PIETRO, Roberto Di, MANCINI, Luigi V. **Intrusion Detection Systems**. Springer Publishing Company: Nova Iorque. 1st ed. 2008.

PYBRAIN. **Pybrain**. Disponível em: <http://pybrain.org>. Acesso em: Janeiro, 2011.

PYTHON. **Python Programming Language**. Disponível em: <http://www.python.org>. Acesso em: Janeiro, 2011.

REZENDE, Solange Oliveira. **Sistemas Inteligentes**: fundamentos e aplicações. Barueri, São Paulo: Manole, 2005.

SIERAKOWSKI, Cezar Augusto. **Inteligência Coletiva Aplicada a Problemas de Robótica Móvel**. Dissertação (Mestrado). Programa de Pós-Graduação em Engenharia de Produção e Sistemas, Pontifícia Universidade Católica, Paraná, 2006.

SILVA, Renato Maia; MAIA, Renato Antonio Grivet M. **Redes Neurais Aplicadas à Detecção de Intrusos em Redes TCP/IP**. SSI 2004 Simpósio de Segurança da Informação, Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos, SP, 2004.

SOBH, Tarek S.; MOSTAFA, Wael M. **A cooperative immunological approach for detecting network anomaly**. Applied Soft Computing, Volume 11, Janeiro, 2011, págs. 1275-1283.

SNORT. **Snort.** Disponível em: <http://www.snort.org>. Acesso em: Janeiro, 2011.

SUNDARAM, Aurobindo. **An introduction to intrusion detection**. Crossroads: The ACM Student Magazine, 2 ed. Abril, 1996.

TANENBAUM, Andrew S. **Redes de Computadores**. Rio de Janeiro: Editora Campus, 4 ed. 2003.

TAVALLAEE, M.; BAGHERI, E.; LU, W.; GHORBANI; A. A. **A Detailed Analysis of the KDD CUP 99 Data Set**. Proceeding of the 2009 IEEE Symposium on Computational Intelligence in Security and Defense Applications. (CISDA), 2009.

TCPDUMP. **TCPDUMP/LIBPCAP public repository**. Disponível em: <http://www.tcpdump.org/>. Acesso em: Janeiro, 2011.

UCHÔA, Joaquim Quinteiro. **Algoritmos Imunoinspirados Aplicados em Segurança Computacional:** Utilização de Algoritmos Inspirados no Sistema Imune para Detecção de Intrusos em Redes de Computadores. Tese (Doutorado em Bioinformática), UFMG, Belo Horizonte, Minas Gerais, 2009.

WANG, Jie. **Computer Network Security:** Theory and Practice. Higher Education Press, Beijing and Springer-Verlag GmbH Berlin Heidelberg, 2009.

WEKA. **Weka 3 –** Data Mining with Open Source Machine Learning Software in Java. Disponível em: <http://www.cs.waikato.ac.nz/ml/weka/>. Acesso em: Janeiro 2011.

WIRESHARK. **Wireshark** – Go Deep. Disponível em: <http://www.wireshark.org>. Acesso em: Janeiro, 2011.

WU, Shelly Xiaonan; BANZHAF, Wolfgang. **The use of computational intelligence in intrusion detection systems**:A review.Applied Soft Computing,Volume 10,Janeiro 2010, pág.1-35.

Anexo I – Algoritmo Genético

AlgoritmoGenetico.py

```
# -*- coding: utf-8 -*-
'''
@author: Jacson RC Silva <jacsonrcsilva@gmail.com>
'''

from Populacao import Populacao

class AlgGenetico:
    ''' Algoritmo Genético
        Esta classe visa facilitar a utilização do Algoritmo Genético
        formado pelas classes Cromossomo e População
    '''
    __tipoGenes     = None
    __qntIndividuos = None
    __funcaoAvaliacao    = None
    __criterioSatisfacao = None
    __considMaiorAvaliacao = None
    __maxGeracoes     = None
    __Populacao       = None
    __retCromo        = None
    __verboso         = None

    def __init__(self, tipoGenes, qntIndividuos, funcaoAvaliacao,    \
                 criterioSatisfacao=0.0, considMaiorAvaliacao=True, \
                 maxGeracoes=0, verboso=False, retCromo=True):
        ''' Construtor da classe
            @param tipoGenes: Matriz com os tipos de Genes que podem
                              ser utilizados nos indivíduos
            @param qntIndividuos: Quantidade de indivíduos que uma
                                  população pode ter
            @param funcaoAvaliacao: Função externa que deverá receber o
                                    Cromossomo como parâmetro e
                                    retornar seu valor de satisfação
            @param criterioSatisfacao: valor de satisfação esperado
                                       pelo cromossomo (critério de
                                       satisfação)
            @param considMaiorAvaliacao: Informa que os indivíduos com
                                         os valores menor devem ser
                                         descartados.
                                         Para descartar os indivíduos
                                         com o maior valor, utilize
                                         considMaiorAvaliacao=False
            @param maxGeracoes: Informa qual o máximo de gerações, ou
                                seja, o máximo de cruzamentos
                                realizados.
                                O padrão é maxGeracoes=0, onde o
```

```
                            critério de término será o
                            "criterioSatisfacao"
        @param verboso: Imprime mensagens na tela no decorrer do
                        processo
        @param retCromo: Indica se a função vai trabalhar com o
                         Cromossomo(True) ou com seu vetor(False)
    '''
    try:
        if type(tipoGenes[0]) != type( list() ):
            raise Exception("ERRO: tipoGenes deve ser uma matriz!")
    except:
        raise Exception("ERRO: tipoGenes deve ser uma matriz!")
    self.__tipoGenes     = tipoGenes
    self.__qntIndividuos = qntIndividuos
    self.__funcaoAvaliacao    = funcaoAvaliacao
    self.__criterioSatisfacao = criterioSatisfacao
    self.__considMaiorAvaliacao = considMaiorAvaliacao
    self.__maxGeracoes     = maxGeracoes
    self.__retCromo        = retCromo
    self.__verboso         = verboso
    self.__Populacao       = None

def ordenaPorAvaliacao(self, cromo):
    ''' Método que devolve a avaliação do cromossomo
        @param: Cromossomo
        @return: Avaliação do cromossomo
    '''
    return cromo.getAvaliacao()

def verCriterioTerminacao(self):
    ''' Verifica se algum dos cromossomos atingiu o critério de
        término
        @return:
            True -> a população alcançou a critério de término
            False -> a população não alcançou a critério de término
    '''
    retorno = []
    for cromo in self.__Populacao.getIndividuos():
        if self.__considMaiorAvaliacao:
            if cromo.getAvaliacao() >= self.__criterioSatisfacao:
                retorno.append( cromo )
        else:
            if cromo.getAvaliacao() <= self.__criterioSatisfacao:
                retorno.append( cromo )

    retorno.sort(key=self.ordenaPorAvaliacao,
                 reverse=self.__considMaiorAvaliacao)
    if len( retorno ) == 0:
        return False, retorno
    else:
        return True, retorno


def evoluir(self):
    ''' Método responsável por controlar toda a execução do
```

```
    algoritmo genético, incluindo os cruzamentos e a mutação
    @return: a última população de cromossomos
'''
from random import randint
retorno = []

self.__Populacao = Populacao( self.__tipoGenes )
self.__Populacao.gerarPopulacao( self.__qntIndividuos )
self.__Populacao.avaliarPopulacao( self.__funcaoAvaliacao )
if self.__criterioSatisfacao != 0.0:
    resp, cromos = self.verCriterioTerminacao()

Ger = 1
if self.__maxGeracoes == 0:
    while not resp:
        self.__Populacao.geraNovaPopulacao(
                            self.__funcaoAvaliacao,
                            self.__considMaiorAvaliacao)
        if randint(1,3) == 1: self.__Populacao.fazerMutacao()
        resp, cromos = self.verCriterioTerminacao()

    if self.__verboso:
        print "Indivíduos que alcançaram o critério " \
              "de Satisfação:"

    for i in cromos:
        retorno.append(i)
        if self.__verboso: print i.getCromossomo()
elif self.__criterioSatisfacao != 0.0:
    while not resp and Ger <= self.__maxGeracoes:
        self.__Populacao.geraNovaPopulacao(
                                    self.__funcaoAvaliacao,
                                    self.__considMaiorAvaliacao)
        if randint(1,3) == 1: self.__Populacao.fazerMutacao()
        resp, cromos = self.verCriterioTerminacao()
        if self.__verboso: print "Geração:", Ger
        Ger+=1

    if resp:
        if self.__verboso:
            print "Indivíduos que alcançaram o critério " \
                  "de Satisfação:"
        for i in cromos:
            retorno.append(i)
            if self.__verboso: print i.getCromossomo()
    else:
        if self.__verboso:
            print "Indivíduos não alcançaram o critério " \
                  "de Satisfação!"
        if self.__verboso: print "A última população foi:"
        for i in self.__Populacao.getIndividuos():
            retorno.append(i)
            if self.__verboso: print i.getCromossomo()
else:
    for Ger in range(self.__maxGeracoes):
```

```
                self.__Populacao.geraNovaPopulacao(
                                    self.__funcaoAvaliacao,
                                    self.__considMaiorAvaliacao,
                                    self.__retCromo)
                if randint(1,3) == 1: self.__Populacao.fazerMutacao()
                if self.__verboso: print "Geração:", Ger

            if self.__verboso: print "A última população foi:"
            for i in self.__Populacao.getIndividuos():
                retorno.append(i)
                if self.__verboso: print i.getCromossomo()
        return retorno
```

População.py

```
# -*- coding: utf-8 -*-
'''
@author: Jacson RC Silva <jacsonrcsilva@gmail.com>
'''

from Cromossomo import Cromossomo
import warnings

class Populacao:
    ''' Classe responsável pelo controle da população de
        cromossomos(indivíduos)
    '''
    __individuos          = None
    __velhaPopulacao      = None
    __historicoIndividuos = None
    __matrizPossib   = None
    # individuos já avaliados?
    __indAvaliados   = None
    # velha população já avaliada?
    __velPopAvaliada = None
    __verboso        = None

    def __init__(self, matrizPossib, verboso=False):
        ''' Cronstutor padrão
            @param matrizPossib: matriz com as possibilidades de genes
                                 existentes
            ex: [ [0,1] , ['a','b'] , [func1,func2] ]
        '''
        self.__individuos          = []
        self.__velhaPopulacao      = []
        self.__historicoIndividuos = []
        self.__matrizPossib = matrizPossib
        self.__indAvaliados   = False
        self.__velPopAvaliada = False
        self.__verboso        = verboso

    def addIndividuo(self, cromo):
```

```
        ''' Adiciona um indivíduo a população
            @param cromo: Cromossomo (indivíduo) a adicionar
        '''
        self.__individuos.append(cromo)

    def getIndividuo(self, posicao):
        ''' Método para obter um indivíduo específico da população
            @param posicao: posição do indivíduo
            @return: retorna o indivíduo (cromossomo)
        '''
        return self.__individuos[posicao]

    def setIndividuo(self, posicao, cromo):
        ''' Troca o indivíduo da população pelo indivíduo recebido
            @param posicao: posição do indivíduo
            @param cromo: novo cromossomo (indivíduo)
        '''
        self.__individuos[ int(posicao) ] = cromo

    def getIndividuos(self):
        ''' Método para obter os indivíduos da população
            @return: todos os indivíduos (cromossomos)
        '''
        return self.__individuos

    def gerarPopulacao(self, qntIndividuos):
        ''' Método para gerar uma população aleatória
            @param qntIndividuos: quantidade de indivíduos da população
        '''
        from random import choice
        # o número de indivíduos deve ser par
        # devido ao método de cruzamento
        qntIndividuos += qntIndividuos%2
        while qntIndividuos > 0:
            individuo = Cromossomo( len(self.__matrizPossib))

            for gene in self.__matrizPossib:
                individuo.addGene( choice(gene) )

            self.addIndividuo(individuo)
            del(individuo)

            qntIndividuos-=1

        self.__indAvaliados = False

    def avaliarPopulacao(self, funcao, avaliarInd=True, retCromo=True):
        ''' Percorre os indivíduos, executando a função de ativação
            fornecida para cada indivíduo.
            @param funcao: funcao de avaliacao de cada indivíduo
            @param avaliarInd:
                        True  -> Avaliar os Indivíduos dessa população
                        False -> Avaliar a velha população
            @param retCromo:
                        True  -> utiliza o cromossomo na função
```

```
                    False -> utiliza o vetor de genes na função
        '''
        if avaliarInd:
            pop = self.__individuos
            self.__indAvaliados = True
        else:
            pop = self.__velhaPopulacao
            self.__velPopAvaliada = True

        for i in pop:
            if retCromo:
                # @todo: let cuter
                print i
                i.setAvaliacao( funcao(i) )
            else:
                i.setAvaliacao( funcao(i.getCromossomo()) )

    def geraNovaPopulacao(self, funcao, baixos=True, retCromo=True):
        ''' Remove os cromossomos piores
            @param baixos: informa se os piores valores são os mais
                           baixos, caso False, ele considerará piores
                           os mais altos
            @param funcao: funcao de avaliacao de cada indivíduo
            @param retCromo:
                        True  -> utiliza o cromossomo na função
                        False -> utiliza o vetor de genes na função
        '''
        def ordenacao(cromo):
            return cromo.getAvaliacao()

        if self.__velhaPopulacao == []:
            if self.__verboso: print 'Fazendo Cruzamento...'
            self.fazerCruzamento()

        if not self.__indAvaliados:
            if self.__verboso:
                print 'Avaliando indivíduos dessa população...'
            self.avaliarPopulacao(funcao, avaliarInd=True,
                                  retCromo=retCromo)
        if not self.__velPopAvaliada:
            if self.__verboso: print 'Avaliando velha população...'
            self.avaliarPopulacao(funcao, avaliarInd=None,
                                  retCromo=retCromo)

        # nova populacao recebe todos os indivíduos
        novaPopulacao = self.__individuos+self.__velhaPopulacao
        # nova população é ordenada pela funcao de ordenacao
        novaPopulacao.sort(key=ordenacao, reverse=baixos)
        # os individuos são definidos como
        self.__individuos = novaPopulacao[:len(self.__individuos)]
        self.__velhaPopulacao = []
        self.__velPopAvaliada = False

    def getSize(self):
        ''' Idem getQntIndividuos
```

```
        '''
        return self.getQntIndividuos()

    def getQntIndividuos(self):
        ''' @return: Retorna a quantidade de Indivíduos
        '''
        return len(self.__individuos)

    def __cruzar(self, ind1, ind2):
        ''' Método que faz o cruzamento entre dois indivíduos
            (Cromossomos)
            @param ind1: indivíduo a sofrer mutação
            @param ind2: indivíduo a sofrer mutação
        '''
        from random import randint
        cruzamentos = randint(1, len(self.__matrizPossib)/2)
        cut = randint(1,len(self.__matrizPossib)-2)
        retorno = [ ind1.getCromossomo()[:cut]+
                    ind2.getCromossomo()[cut:]
                    ,
                    ind2.getCromossomo()[:cut]+
                    ind1.getCromossomo()[cut:]
                  ]
        cruzamentos-=1
        while cruzamentos > 0:
            cut = randint(1,len(self.__matrizPossib)-2)
            retorno = [ retorno[0][:cut]+retorno[1][cut:] ,
                        retorno[1][:cut]+retorno[0][cut:]  ]
            cruzamentos-=1
        return [ Cromossomo( len(retorno[0] ), retorno[0] ),
                 Cromossomo( len(retorno[1] ), retorno[1] )  ]

    def verificarDuplo(self, cromo):
        ''' Verifica se o individuo (cromossomo) já existe na população
            @param cromo: cromossomo a avaliar
            @param usarHistorico: guarda o histórico de todos os
                                  indivíduos
            @return: True  -> indivíduo já existe na população
                     False -> indivíduo não existe a população
        '''
        for i in self.__velhaPopulacao:
            if i.getCromossomo() == cromo.getCromossomo():
                return True
        return False

    def fazerCruzamento(self):
        ''' Realiza o cruzamento entre os indivíduos da População
        '''
        if len(self.__individuos) < 1:
            warnings.warn("Primeiro gere uma população com " \
                          "mais de 1 elemento", stacklevel=2)
        else:
            self.__velhaPopulacao = list(self.getIndividuos())
            self.__velPopAvaliada = self.__indAvaliados
            self.__historicoIndividuos += self.__velhaPopulacao
```

```
            from random import shuffle
            # cria vetor com o mesmo número de indivíduos
            aux = range( 0, self.getQntIndividuos() )
            # embaralha o vetor
            shuffle(aux)
            # separa o vetor ao meio
            id1 = aux[:self.getQntIndividuos()/2]
            id2 = aux[self.getQntIndividuos()/2:]

            # faz o cruzamento entre os indivíduos
            for i in range(self.getQntIndividuos()/2):
                tentativas    = 30
                ret1_OK       = False
                ret2_OK       = False
                while (not ret1_OK or not ret2_OK) and tentativas > 0:
                    ret1, ret2 = self.__cruzar(
                                            self.getIndividuo( id1[i] ),
                                            self.getIndividuo( id2[i] ) )
                    # para cada um dos retornos do Cruzamento
                    for retC in [ ret1, ret2 ]:
                        # se não for um indivíduo já existente
                        if not self.verificarDuplo( retC ):
                            # se ainda nao tiver adicionado na pos. id1
                            if not ret1_OK:
                                self.setIndividuo(id1[i], retC)
                                ret1_OK = True
                            # se ainda nao tiver adicionado na pos. id2
                            elif not ret2_OK:
                                self.setIndividuo(id2[i], retC)
                                ret2_OK = True
                    #DEBUG: print "Tentativa:", tentativas
                    tentativas -= 1

                if tentativas == 0:
                    if not ret1_OK:
                        self.setIndividuo(id1[i], ret1)
                    if not ret2_OK:
                        self.setIndividuo(id2[i], ret2)

                self.__indAvaliados = False

    def fazerMutacao(self):
        ''' Realiza a mutação em um dos genes de um dos indivíduos
            A escolha é aleatória
        '''
        from random import randint
        # qual gene mutar:
        pos_gene = randint(0,len(self.__matrizPossib)-1)
        # qual cromossomo:
        pos_cromo = randint(1,self.getQntIndividuos()-1)

        # obtendo o gene atual do cromossomo
        gene_atual = self.getIndividuo(pos_cromo).getGene(pos_gene)
        # selecionando a mutação do gene
```

```
        gene_novo  = self.__matrizPossib[pos_gene][ \
                        randint(0,len(self.__matrizPossib[pos_gene])-1)
]
        if len(self.__matrizPossib[pos_gene]) > 1:
            while gene_atual == gene_novo:
                gene_novo  = self.__matrizPossib[pos_gene][ \
                             randint(0, len(
                                 self.__matrizPossib[pos_gene])-1) ]

        #@todo: turn it better, because this is ugly :P
        # modificando o cromossomo
        self.__individuos[pos_cromo].setGene(pos_gene, gene_novo)
```

Cromossomo.py

```
# -*- coding: utf-8 -*-
'''
@author: Jacson RC Silva <jacsonrcsilva@gmail.com>
'''

import warnings

class Cromossomo:
    '''
    Classe que abriga um Cromossomo de tamanho e
    campos determinados pelo usuário
    '''
    __cromossomo = []
    __qntGenes   = 0
    __avaliacao  = 0.0

    def __init__(self, qntGenes, cromo=None, avaliacao=0.0):
        ''' Construtor, recebe a quantidade de genes
                        que o cromossomo terá
            @param qntGenes: Quantidade de Genes do Cromossomo
            @param cromo: Vetor para iniciar o objeto desse cromossomo
            @param avaliacao: Valor de aavliação desse cromossomo
        '''
        self.__avaliacao  = avaliacao
        self.setQntGenes(qntGenes)
        if cromo != None:
            self.__cromossomo = list(cromo)
        else:
            self.__cromossomo = []

    def __len__(self):
        return self.__qntGenes

    def addGene(self, Gene):
        ''' Adiciona um Gene, se possível, ao final do cromossomo
            @param Gene: Gene para adicionar ao cromossomo
        '''
```

```
        if len(self.__cromossomo) > self.__qntGenes:
            warnings.warn("Este gene não pôde ser adicionado,"+
                          "ele já possui o máximo de genes!" %
                          self.__qntGenes, stacklevel=2)
            return False
        else:
            self.__cromossomo.append(Gene)
            return True

    def setGene(self, posicao, Gene):
        ''' Define o valor de determinado Gene do Cromossomo
            pelo valor Gene recebido
            @param posicao: indica a posição a modificar
            @param Gene: Gene para adicionar na posição especificada
        '''
        posicao = int(posicao)

        if posicao >= self.__qntGenes:
            #raise Exception("Posição Inválida")
            return False
        else:
            self.__cromossomo[posicao] = Gene
            return True

    def getGene(self, posicao):
        ''' Retorna o Gene de Determinada posição do cromossomo
            @param posicao: Posição desejada do Gene a obter
        '''
        posicao = int(posicao)

        try:
            return self.__cromossomo[posicao]
        except:
            return 0

    def getSize(self):
        ''' @return: Retorna o tamanho do cromossomo
        '''
        return len(self.__cromossomo)

    def setQntGenes(self, qntGenes):
        ''' Define a quantidade máxima de Genes
            que o cromossomo pode ter
            @param qntGenes: número de cromossomos
        '''
        self.__qntGenes = qntGenes
        if len(self.__cromossomo) > self.__qntGenes:
            warnings.warn("Diminiundo o cromossomo para %d genes" %
                                        self.__qntGenes, stacklevel=2 )
            self.__cromossomo = self.__cromossomo[0:self.__qntGenes]

    def getQntGenes(self):
        ''' @return: retorna o número máximo de Genes
                     que o cromossomo pode ter
        '''
```

```
        return self.__qntGenes

    def setCromossomo(self, cromo):
        ''' Define todos os genes do cromossomo
            @param cromo: cromossomo
        '''
        if len(cromo) != self.__qntGenes:
            raise Exception("Quantidade de Genes Errada")
        else:
            self.__cromossomo = cromo
            return True

    def getCromossomo(self, posicoes=None):
        ''' @return: Retorna o cromossomo inteiro ou uma parte do mesmo
            @param posicoes: Lista de dois elementos, indicando a
                             posição de início e de fim do cromossomo
        '''
        if posicoes == None:
            if len(self.__cromossomo) < self.__qntGenes:
                warnings.warn("O Cromossomo não está completo",
                              stacklevel=2)
                print self.__cromossomo
            return self.__cromossomo
        elif len(posicoes) != 2:
            raise Exception("O arg. 2 deve ser uma lista de dois
itens")
        else:
            return self.__cromossomo[ posicoes[0]:posicoes[1] ]

    def getPosOf(self, valor):
        ''' @return: retorna a posição do Gene
                     correspondente ao @param valor
            @param valor: valor do Gene a procurar no cromossomo
        '''
        try:
            return self.__cromossomo.index(valor)
        except:
            return -1

    def getQntPosicoesVazias(self):
        ''' @return: Retorna a quantidade de posições não preenchidas
                     do cromossomo
        '''
        return self.__qntGenes - len(self.__cromossomo)

    def setAvaliacao(self, valor):
        ''' Define seu valor de avaliação (aptidão)
            @param valor: valor de avaliação
        '''
        self.__avaliacao = valor

    def getAvaliacao(self):
        ''' @return: valor de avaliação
        '''
        return self.__avaliacao
```

```
def clean(self):
    ''' Limpa o cromossomo e sua avaliação
    '''
    self.__avaliacao  = 0.0
    self.__cromossomo = []

def __repr__(self):
    r = ""
    for i in self.__cromossomo:
        r += str(i) + " "
    return r
```

Anexo II – Criação e execução da 1ª RNA

CriarRedeNeural.py

```
# -*- coding: utf-8 -*-
'''
Created on 19/01/2011

@author: Jacson RC Silva@gmail.com
'''

from pyfann import libfann
import os

def print_callback(epochs, error):
    print "Épocas     %8d. Erro MSE Atual: %.10f\n" % (epochs, error)
    return 0

# Inicializando parâmetros
connectionRate = 1
learningRate   = 0.3
momentum       = 0.2
desiredError   = 0.000001
maxIterations  = 500
iterationsBetweenReports = 25

InputNeurons  = 41
HiddenNeurons = 21
OutputNeutons = 2

print "Criando a Rede Neural Artificial..."
RNA = libfann.neural_net()
RNA.create_sparse_array(connectionRate,
                        ( InputNeurons, HiddenNeurons, OutputNeutons ))
RNA.set_learning_rate( learningRate )
RNA.set_learning_momentum( momentum )

print "Definindo os algoritmos de treinamento..."
RNA.set_activation_function_hidden( libfann.SIGMOID )
RNA.set_activation_function_output( libfann.SIGMOID_STEPWISE )
RNA.set_training_algorithm( libfann.TRAIN_INCREMENTAL )

print "Abrindo o arquivo de treino..."
DadosTreinamento = libfann.training_data()
DadosTreinamento.read_train_from_file( "KDDTrain+_20Percent.train" )
```

```
print "Treinando a Rede..."
RNA.train_on_data( DadosTreinamento, maxIterations,
                   iterationsBetweenReports, desiredError )

print "Testando a Rede..."
DadosTeste = libfann.training_data()
DadosTeste.read_train_from_file("KDDTest+.train")

RNA.reset_MSE()
RNA.test_data( DadosTeste )
print "MSE error on test data: %f" % RNA.get_MSE()

print "Salvando a rede"
RNA.save( os.path.join("RedeNeural.net") )

RNA.destroy()
```

ExecutarRedeNeural.py

```
# -*- coding: utf-8 -*-
'''
Created on 19/01/2011

@author: Jacson RC Silva
'''

from sys import argv, exit

if len(argv) != 2:
    print 'Utilize: %s rede.net' % argv[0]
    exit(1)

arquivosTeste = [ 'KDDTest-21.train', 'KDDTest+.train' ]

from pyfann import libfann
RNA = libfann.neural_net()
RNA.create_from_file( argv[1] )

print "Testando a Rede com as entradas..."
for filename in arquivosTeste:
    DadosTeste = libfann.training_data()
    DadosTeste.read_train_from_file( filename )

    entrada = DadosTeste.get_input()
    saida   = DadosTeste.get_output()
    Normal = 0
    Normal_Ataque = 0
    Normal_Normal = 0
    Ataque = 0
    Ataque_Normal = 0
    Ataque_Ataque = 0
    for i in range( len(entrada) ):
```

```
            resposta = RNA.run(entrada[i])

            if saida[i][0] == 1.0:
                Ataque+=1
                if resposta[0] > resposta[1]:
                    Ataque_Ataque+=1
                else:
                    Ataque_Normal+=1
            else:
                Normal+=1
                if resposta[0] > resposta[1]:
                    Normal_Ataque+=1
                else:
                    Normal_Normal+=1

        DadosTeste.destroy_train()

        print "_"*60
        print "[ Estatísticas %s ]" % (filename.split('/')[-1])
        print "Total de entradas: ", Normal + Ataque

        print "Tráfego Malicioso Detectado (VP):     %d de %d %.0f%%" %
            (Ataque_Ataque,Ataque,float(Ataque_Ataque)/float(Ataque)*100.0)

        print "Tráfego Malicioso Não Detectado (FN): %d de %d %.0f%%" %
            (Ataque_Normal,Ataque,float(Ataque_Normal)/float(Ataque)*100.0)

        print "Tráfego Normal Detectado (VN):        %d de %d %.0f%%" %
            (Normal_Normal,Normal,float(Normal_Normal)/float(Normal)*100.0)

        print "Tráfego Normal Não Detectado (FP):    %d de %d %.0f%%" %
            (Normal_Ataque,Normal,float(Normal_Ataque)/float(Normal)*100.0)

        print "Taxa de Sucesso: %d de %d %.0f%%" %
            (Ataque_Ataque+Normal_Normal, Normal+Ataque,
                 (Ataque_Ataque+Normal_Normal)/float(Normal+Ataque)*100.0)
        print       "_"*60

RNA.destroy()
```

Anexo III – Execução do Algoritmo Genético para formar a RNA

RNAGenetico.py

```
# -*- coding: utf-8 -*-
'''
@author: Jacson RC Silva <jacsonrcsilva@gmail.com>
'''

from algoritmoGenetico import AlgGenetico
from pyfann import libfann
from math import sqrt

qntNeuroniosEntrada  = 41
qntNeuroniosSaida    = 2

# Genes de um cromossomo
# x*0.1 for x in (range(1,9)) = 0.1,..., 0.9
learningRate = [ x*0.1 for x in (range(1,9)) ]
momentum     = [ x*0.1 for x in (range(1,9)) ]
lrdecay      = [ x*0.1 for x in (range(1,9)) ]
trainers     = [  libfann.TRAIN_INCREMENTAL, libfann.TRAIN_BATCH,
                  libfann.TRAIN_INCREMENTAL, libfann.TRAIN_QUICKPROP,
                  libfann.TRAIN_RPROP ]
camadas      = [  libfann.SIGMOID, libfann.SIGMOID_SYMMETRIC,
                  libfann.SIGMOID_STEPWISE,
libfann.SIGMOID_SYMMETRIC_STEPWISE,
                  libfann.LINEAR, libfann.LINEAR_PIECE,
                  libfann.LINEAR_PIECE_SYMMETRIC, libfann.THRESHOLD,
                  libfann.THRESHOLD_SYMMETRIC, libfann.GAUSSIAN,
                  libfann.GAUSSIAN_STEPWISE, libfann.GAUSSIAN_SYMMETRIC,
                  libfann.ELLIOT, libfann.ELLIOT_SYMMETRIC,
                  libfann.COS_SYMMETRIC, libfann.SIN_SYMMETRIC ]

neuroniosOcultos  = [ 15, 20, 25, 30, 35, 40, 45, 50, 60,
                    int(sqrt(qntNeuroniosEntrada+qntNeuroniosSaida)*2),
                    int((qntNeuroniosEntrada+qntNeuroniosSaida)/2),
                    (2*qntNeuroniosEntrada+1) ]
epocasTreinamento = [ 100, 200, 300 ]

# Arquivo que conterá toda a saída da avaliação algoritmo genético
log = open("Execucao_Alg_Gen.log", "w")

def avaliacaoRNA(cromo):
    ''' Função responsável pela avaliação da RNA
    '''
```

```
    try:
        c = cromo.getCromossomo()
        log.write("Treinando %s"%c)
        from os import popen
        resultado = popen(
               "python Avaliador_Fann.py %s %s %s %s %s %s %s %s" % \
               (c[0], c[1], c[2], c[3], c[4], c[5], c[6], c[7] )).
               readlines()

        for i in resultado:
            log.write(i)

        log.write("_"*50+'\n')
        return float(resultado[-1].split()[-1])
    except:
        log.write("Resultado: 999.0 " \
                  "(ERRO: Combinação de rede errada)\n")
        log.write("_"*50+'\n')
        return 999.0


# Matriz com os tipos de genes
tipoGenes = [ camadas, camadas, neuroniosOcultos, trainers,
              learningRate, momentum, lrdecay, epocasTreinamento ]
''' Conteúdo do vetor tipoGenes:
    [0] Função da camada oculta
    [1] Função da camada de saída
    [2] Quantidade de Neurônios Ocultos
    [3] Algoritmo de Aprendizado
    [4] Learning Rate
    [5] Momentum
    [6] Lr Decay
    [7] Quantidade de épocas a treinar
'''

# Instância do Algoritmo Genético:
AG = AlgGenetico(tipoGenes, 20, avaliacaoRNA, criterioSatisfacao=0.01,
                 considMaiorAvaliacao=False, maxGeracoes=100,
                 verboso=True)
# Obtendo os resultados:
resultado = AG.evoluir()

# Fechando o Log da Avaliação
log.close()

# Salvando os resultados:
arqResultado = open("Resultado_Alg-Genetico.txt","w")
for i in resultado:
    for c in i.getCromossomo():
        arqResultado.write(str(c)+' ')
    arqResultado.write('\n'+str( i.getAvaliacao() )+'\n')
arqResultado.write('\n')
arqResultado.close()
```

Avaliador_Fann.py

```
# -*- coding: utf-8 -*-
'''
Script responsável por receber os parâmetros do algoritmo Genético e
criar a Rede Neural referente, retornando ao fim do código a expressão
"Resultado:", seguida do valor do resultado

@author: Jacson RC Silva <jacsonrcsilva@gmail.com>
'''

from sys import argv, exit

if len(argv) < 8:
   print 'Utilize: %s função-Cam.Oculta função-Cam.Saída Qnt-
Neur.Ocultos  \\' % argv[0]
   print 'Alg.Aprendizado LearningRate Momentum Decay' \
         'epocasTreinamento'
   exit (1)

from pyfann import libfann

cromo = argv
qntNeuroniosEntrada = 41
qntNeuroniosSaida   = 2

# genes do cromossomo
nomeArqTreino = "KDDTrain+_20Percent.train"
nomeArqTeste  = "KDDTest+.train"
fcOculta      = int  (cromo[1])
fcSaida       = int  (cromo[2])
neurOcultos   = int  (cromo[3])
algAprend     = int  (cromo[4])
learnR        = float(cromo[5])
moment        = float(cromo[6])
decay         = float(cromo[7])
epocasTreino  = int  (cromo[8])

# abrindo arq. treino e testes
dadosTreino = libfann.training_data()
dadosTeste = libfann.training_data()
dadosTreino.read_train_from_file( nomeArqTreino )
dadosTeste.read_train_from_file ( nomeArqTeste  )

# criando rede
RNA = libfann.neural_net()
RNA.create_sparse_array(1,
                  (qntNeuroniosEntrada, neurOcultos, qntNeuroniosSaida))
RNA.set_learning_rate( learnR )
RNA.set_learning_momentum( moment )

# treinamento
```

```
RNA.set_activation_function_hidden( fcOculta )
RNA.set_activation_function_output( fcSaida  )
RNA.set_training_algorithm( algAprend )
if RNA.get_training_algorithm() == libfann.TRAIN_QUICKPROP:
    RNA.set_quickprop_decay(decay)

maxIterations =epocasTreino
iterationsBetweenReports = 50
desiredError = 0.00001
RNA.train_on_data( dadosTreino, maxIterations,
                   iterationsBetweenReports, desiredError )

# Avaliação pela taxa de acerto
entrada = dadosTeste.get_input()
saida   = dadosTeste.get_output()
Normal = 0
Normal_Ataque = 0
Normal_Normal = 0
Ataque = 0
Ataque_Normal = 0
Ataque_Ataque = 0
for i in range( len(entrada) ):
    resposta = RNA.run(entrada[i])
    if saida[i][0] == 1.0:
        Ataque+=1
        if resposta[0] > resposta[1]:
            Ataque_Ataque+=1
        else:
            Ataque_Normal+=1
    else:
        Normal+=1
        if resposta[1] >= resposta[0]:
            Normal_Normal+=1
        else:
            Normal_Ataque+=1

dadosTeste.destroy_train()
#_____________________________________

try:
    print "FP: %d FN: %d (Total: %d)" %
      (Normal_Ataque, Ataque_Normal, Normal+Ataque)
    # devolve a taxa de erro
    print "Resultado:", \
      (float(Normal_Ataque)+float(Ataque_Normal))/
      (float(Normal)+float(Ataque))
except:
    print "Resultado: 999.0"

RNA.destroy()
```

Anexo IV – Lógica Nebulosa

VariavelLinguistica.py

```
# -*- coding: utf-8 -*-
'''
@author: Jacson RC Silva <jacsonrcsilva@gmail.com>
'''

from fuzz import
FuzzySet,FuzzyElement,TriangularFuzzyNumber,TrapezoidalFuzzyNumber

class VariavelLinguistica:
    ''' Classe responsável por abrigar conjuntos nebulosos e
        clássicos de uma Variável Linguística
    '''
    __conjuntosDiscretos = None
    __conjuntosContinuos = None
    __conjContMaxMin     = None
    __discreta           = None
    __nome               = None
    __indice             = None

    def __init__(self, discreto=False, valoresDiscretos=None,
continuo=False, valorMinimo=None, valorMaximo=None, nome="", indice=0):
        ''' Construtor
            @param discreto:
                   informa se a variável conterá conjuntos discretos
            @param valoresDiscretos:
                   lista com os valores discretos do conjunto
            @param continuo:
                   informa se a variável conterá conjuntos contínuos
            @param valorMinimo:
                   caso o conjunto seja contínuo, informa o valor
                   mínimo dos dados
            @param valorMaximo:
                   caso o conjunto seja contínuo, informa o valor
                   máximo dos dados
            @param nome: nome da variável linguística
            @param indice:
                   adaptação para o KDD,
                   informando o índice do atributo
        '''
        if discreto and continuo:
            raise ValueError("Não há como um conjunto ser discreto"+
                             " e contínuo ao mesmo tempo")
        if not discreto and not continuo:
            raise ValueError("Informe se o conjunto é "+
                             "discreto ou contínuo")
```

```
        self.__conjuntosDiscretos = {}
        self.__conjuntosContinuos = {}
        self.__conjContMaxMin     = {}
        self.__nome               = nome
        self.__indice             = int(indice)
        if discreto:
            self.__orgDiscreto(valoresDiscretos)
        elif continuo:
            self.__orgContinuo(valorMinimo, valorMaximo)

    def __orgDiscreto(self, valoresDiscretos):
        ''' Organiza a classe para seus conjuntos discretos
        '''
        self.__discreta = True

        if type(valoresDiscretos) != type(list()):
            valoresDiscretos = [valoresDiscretos]
        valoresDiscretos =[  self.conjuntoFuzzy( (i, 1.0) ) \
                                    for i in valoresDiscretos ]

        for i in valoresDiscretos:
            self.__conjuntosDiscretos.update( {i.keys()[0]:i} )

    def __orgContinuo(self, valorMinimo, valorMaximo):
        ''' Organiza a classe para seus conjuntos contínuos
        '''
        valorMinimo = float(valorMinimo)
        valorMaximo = float(valorMaximo)
        self.__discreta = False
        self.__conjContMaxMin.update( {'Min':valorMinimo,
'Max':valorMaximo} )

        aux = valorMaximo - valorMinimo
        p = [0.166*aux, 0.333*aux, 0.5*aux, 0.666*aux, 0.833*aux]
        self.__conjuntosContinuos.update( {
        'MB': TrapezoidalFuzzyNumber( kernel=(valorMinimo, p[0]),
                                      support=(valorMinimo, p[1])  ),
        'B' :  TriangularFuzzyNumber(p[1], (p[0], p[2] ) ),
        'M' :  TriangularFuzzyNumber(p[2], (p[1], p[3] ) ),
        'A' :  TriangularFuzzyNumber(p[3], (p[2], p[4] ) ),
        'MA': TrapezoidalFuzzyNumber(kernel=(p[4], valorMaximo),
                                      support=(p[3], valorMaximo)  ),
                                })

    def getNome(self):
        ''' Obter Nome da classe
            @return: nome da classe
        '''
        return self.__nome

    def getIndice(self):
        '''  Obter índice KDD do atributo
            @return: índice KDD do atributo
        '''
```

```
        return self.__indice

    def __repr__(self):
        return '<%s %s>' % (self.__class__.__name__, self.__nome)

    def getConjNomes(self):
        ''' Obtém os Nomes dos Conjuntos da Classe
            @return: os Nomes dos Conjuntos da Classe
        '''
        if self.getEhDiscreto():
            return self.__conjuntosDiscretos.keys()
        else:
            return self.__conjuntosContinuos.keys()


    def getEhDiscreto(self):
        '''
        @return: True -> é discreto | False -> não é discreto
        '''
        return self.__discreta

    def getEhContinuo(self):
        '''
        @return: True -> é contínuo | False -> não é contínuo
        '''
        return not self.__discreta

    def conjuntoFuzzy(self, elementos ):
        ''' Cria um conjunto Fuzzy Discreto
            @param elementos: elementos de um conjunto Fuzzy
            @return: o conjunto Fuzzy Discreto criado
        '''
        if type(elementos) != type(list()):
            elementos = [elementos]
        elementos = [ FuzzyElement(i[0],i[1]) for i in elementos ]
        return FuzzySet( elementos )

    def mu(self, conjunto, valor):
        ''' Obter o Grau de pertinência do valor em determinado
            conjunto
            @param conjunto: conjunto a ser pesquisado
            @param valor: valor a ser pesquisado
            @return: o grau de pertinência do elemento do grupo
        '''
        # discreta
        if self.__discreta:
            return self.__conjuntosDiscretos[conjunto].mu(valor)
        # contínua
        else:
            valor = float(valor)-self.__conjContMaxMin['Min']
            return self.__conjuntosContinuos[conjunto].mu(valor)


def AND(valA, valB):
    ''' operação AND
```

```
    '''
    return min(valA, valB)

def OR(valA, valB):
    ''' operação OR
    '''
    return max(valA, valB)

def NOT(val):
    ''' operação NOT
    '''
    return 1.0 - val
```

Dados_KDDCUP.py

```
# -*- coding: utf-8 -*-
'''
@author: Jacson RC Silva <jacsonrcsilva@gmail.com>
'''

from VariavelLinguistica import VariavelLinguistica

# 1: duration
duration = VariavelLinguistica(continuo=True, valorMinimo=0,
                    valorMaximo=57715, nome='duration', indice=0)
# 2: protocol_type
protocol_type = VariavelLinguistica( discreto=True,
                    valoresDiscretos=[ 'icmp','tcp','udp' ] ,
                    nome='protocol_type', indice=1            )
# 3: service
service = VariavelLinguistica(discreto=True, valoresDiscretos=[
                   'aol','auth',
                   'bgp','courier','csnet_ns','ctf','daytime',
                   'discard','domain','domain_u','echo','eco_i',
                   'ecr_i','efs','exec','finger','ftp','ftp_data',
                   'gopher','harvest','hostnames','http','http_2784',
                   'http_443','http_8001','imap4','IRC','iso_tsap',
                   'klogin','kshell','ldap','link','login','mtp',
                   'name','netbios_dgm','netbios_ns','netbios_ssn',
                   'netstat','nnsp','nntp','ntp_u','other','pm_dump',
                   'pop_2','pop_3','printer','private','red_i',
                   'remote_job','rje','shell','smtp','sql_net',
                   'ssh','sunrpc','supdup','systat','telnet',
                   'tftp_u','time','tim_i','urh_i','urp_i',
                   'uucp','uucp_path','vmnet','whois',
                   'X11','Z39_50' ] , nome='service', indice=2)
# 4: flag
flag = VariavelLinguistica(discreto=True, valoresDiscretos=[
                   'OTH','REJ','RSTO',
                   'RSTOS0','RSTR',
                   'S0','S1','S2',
                   'S3','SF','SH' ] ,
```

```
                    nome='flag', indice=3 )
# 5: src_bytes
src_bytes = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=1379963888, nome='src_bytes', indice=4)
# 6: dst_bytes
dst_bytes = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=1309937401, nome='dst_bytes', indice=5)
# 7: land
land = VariavelLinguistica(discreto=True,
            valoresDiscretos=[ '0','1' ] ,
            nome='land', indice=6 )
# 8: wrong_fragment
wrong_fragment = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=3, nome='wrong_fragment', indice=7)
# 9: urgent
urgent = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=3, nome='urgent', indice=8)
# 10: hot
hot = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=101, nome='hot', indice=9)
# 11: num_failed_logins
num_failed_logins = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=5, nome='num_failed_logins', indice=10)
# 12: logged_in
logged_in = VariavelLinguistica(discreto=True,
            valoresDiscretos=[ '0','1' ] ,
            nome='logged_in', indice=11)
# 13: num_compromised
num_compromised = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=7479, nome='num_compromised', indice=12)
# 14: root_shell
root_shell = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=1, nome='root_shell', indice=13)
# 15: su_attempted
su_attempted = VariavelLinguistica(continuo=True,
            valorMinimo=0, valorMaximo=2,
            nome='su_attempted', indice=14)
# 16: num_root
num_root = VariavelLinguistica(continuo=True,
            valorMinimo=0, valorMaximo=7468,
            nome='num_root', indice=15)
# 17: num_file_creations
num_file_creations = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=100, nome='num_file_creations', indice=16)
# 18: num_shells
num_shells = VariavelLinguistica(continuo=True,
            valorMinimo=0, valorMaximo=5,
            nome='num_shells', indice=17)
# 19: num_access_files
num_access_files = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=9, nome='num_access_files', indice=18)
# 20: num_outbound_cmds
num_outbound_cmds = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=0, nome='num_outbound_cmds', indice=19)
# 21: is_host_login
```

```
is_host_login = VariavelLinguistica(discreto=True,
            valoresDiscretos=['0','1'],
            nome='is_host_login', indice=20)
# 22: is_guest_login
is_guest_login = VariavelLinguistica(discreto=True,
            valoresDiscretos=['0','1'] ,
            nome='is_guest_login', indice=21)
# 23: count
count = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=511,
            nome='count', indice=22)
# 24: srv_count
srv_count = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=511,
            nome='srv_count', indice=23)
# 25: serror_rate
serror_rate = VariavelLinguistica(continuo=True, valorMinimo=0.00,
            valorMaximo=1.00, nome='serror_rate', indice=24)
# 26: srv_serror_rate
srv_serror_rate = VariavelLinguistica(continuo=True, valorMinimo=0.00,
            valorMaximo=1.00, nome='srv_serror_rate', indice=25)
# 27: rerror_rate
rerror_rate = VariavelLinguistica(continuo=True, valorMinimo=0.00,
            valorMaximo=1.00, nome='rerror_rate', indice=26)
# 28: srv_rerror_rate
srv_rerror_rate = VariavelLinguistica(continuo=True, valorMinimo=0.00,
            valorMaximo=1.00, nome='srv_rerror_rate', indice=27)
# 29: same_srv_rate
same_srv_rate = VariavelLinguistica(continuo=True, valorMinimo=0.00,
            valorMaximo=1.00, nome='same_srv_rate', indice=28)
# 30: diff_srv_rate
diff_srv_rate = VariavelLinguistica(continuo=True, valorMinimo=0.00,
            valorMaximo=1.00, nome='diff_srv_rate', indice=29)
# 31: srv_diff_host_rate
srv_diff_host_rate = VariavelLinguistica(continuo=True,
            valorMinimo=0.00,
            valorMaximo=1.00, nome='srv_diff_host_rate', indice=30)
# 32: dst_host_count
dst_host_count = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=255, nome='dst_host_count', indice=31)
# 33: dst_host_srv_count
dst_host_srv_count = VariavelLinguistica(continuo=True, valorMinimo=0,
            valorMaximo=255, nome='dst_host_srv_count', indice=32)
# 34: dst_host_same_srv_rate
dst_host_same_srv_rate = VariavelLinguistica(continuo=True,
            valorMinimo=0.00,
            valorMaximo=1.00, nome='dst_host_same_srv_rate',indice=33)
# 35: dst_host_diff_srv_rate
dst_host_diff_srv_rate = VariavelLinguistica(continuo=True,
            valorMinimo=0.00,
            valorMaximo=1.00, nome='dst_host_diff_srv_rate',indice=34)
# 36: dst_host_same_src_port_rate
dst_host_same_src_port_rate =VariavelLinguistica(continuo=True,
            valorMinimo=0.00,
            valorMaximo=1.00, nome='dst_host_same_src_port_rate',
```

```
            indice=35)
# 37: dst_host_srv_diff_host_rate
dst_host_srv_diff_host_rate = VariavelLinguistica(continuo=True,
            valorMinimo=0.00,
            valorMaximo=1.00, nome='dst_host_srv_diff_host_rate',
            indice=36)
# 38: dst_host_serror_rate
dst_host_serror_rate = VariavelLinguistica(continuo=True,
            valorMinimo=0.00,
            valorMaximo=1.00, nome='dst_host_serror_rate', indice=37)
# 39: dst_host_srv_serror_rate
dst_host_srv_serror_rate = VariavelLinguistica(continuo=True,
            valorMinimo=0.00,
            valorMaximo=1.00, nome='dst_host_srv_serror_rate',
            indice=38)
# 40: dst_host_rerror_rate
dst_host_rerror_rate = VariavelLinguistica(continuo=True,
            valorMinimo=0.00,
            valorMaximo=1.00, nome='dst_host_rerror_rate', indice=39)
# 41: dst_host_srv_rerror_rate
dst_host_srv_rerror_rate = VariavelLinguistica(continuo=True,
            valorMinimo=0.00,
            valorMaximo=1.00, nome='dst_host_srv_rerror_rate',
            indice=40)

# Lista com todos os atributos do KDD
KDDCUP      = [ duration, protocol_type, service, flag, src_bytes,
                dst_bytes, land, wrong_fragment, urgent, hot,
                num_failed_logins, logged_in,
                num_compromised, root_shell, su_attempted,num_root,
                num_file_creations, num_shells, num_access_files,
                num_outbound_cmds, is_host_login, is_guest_login, count,
                srv_count, serror_rate, srv_serror_rate, rerror_rate,
                srv_rerror_rate, same_srv_rate, diff_srv_rate,
                srv_diff_host_rate, dst_host_count, dst_host_srv_count,
                dst_host_same_srv_rate, dst_host_diff_srv_rate,
                dst_host_same_src_port_rate,
                dst_host_srv_diff_host_rate,
                dst_host_serror_rate,dst_host_srv_serror_rate,
                dst_host_rerror_rate, dst_host_srv_rerror_rate ]

# Dicionário com todos os campos do KDD
KDDCUP_Dict = { 'duration':duration, 'protocol_type':protocol_type,
            'service':service, 'flag':flag, 'src_bytes':src_bytes,
            'dst_bytes':dst_bytes, 'land':land,
            'wrong_fragment':wrong_fragment,
            'urgent':urgent, 'hot':hot,
            'num_failed_logins':num_failed_logins,
            'logged_in':logged_in,
            'num_compromised':num_compromised, 'root_shell':root_shell,
            'su_attempted':su_attempted, 'num_root':num_root,
            'num_file_creations':num_file_creations,
            'num_shells':num_shells,
            'num_access_files':num_access_files,
            'num_outbound_cmds':num_outbound_cmds,
```

```
            'is_host_login':is_host_login,
            'is_guest_login':is_guest_login,
            'count':count, 'srv_count':srv_count,
            'serror_rate':serror_rate,
            'srv_serror_rate':srv_serror_rate,
            'rerror_rate':rerror_rate,
            'srv_rerror_rate':srv_rerror_rate,
            'same_srv_rate':same_srv_rate,
            'diff_srv_rate':diff_srv_rate,
            'srv_diff_host_rate':srv_diff_host_rate,
            'dst_host_count':dst_host_count,
            'dst_host_srv_count':dst_host_srv_count,
            'dst_host_same_srv_rate':dst_host_same_srv_rate,
            'dst_host_diff_srv_rate':dst_host_diff_srv_rate,
            'dst_host_same_src_port_rate':dst_host_same_src_port_rate,
            'dst_host_srv_diff_host_rate':dst_host_srv_diff_host_rate,
            'dst_host_serror_rate':dst_host_serror_rate,
            'dst_host_srv_serror_rate':dst_host_srv_serror_rate,
            'dst_host_rerror_rate':dst_host_rerror_rate,
            'dst_host_srv_rerror_rate':dst_host_srv_rerror_rate
                }

# Lista com todos os Nomes do KDD
KDDCUP_Names= [ 'duration', 'protocol_type', 'service', 'flag',
                'src_bytes', 'dst_bytes', 'land',
                'wrong_fragment','urgent', 'hot',
                'num_failed_logins', 'logged_in', 'num_compromised',
                'root_shell', 'su_attempted', 'num_root',
                'num_file_creations',
                'num_shells', 'num_access_files', 'num_outbound_cmds',
                'is_host_login', 'is_guest_login', 'count',
                'srv_count', 'serror_rate', 'srv_serror_rate',
                'rerror_rate',
                'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
                'srv_diff_host_rate', 'dst_host_count',
                'dst_host_srv_count',
                'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
                'dst_host_same_src_port_rate',
                'dst_host_srv_diff_host_rate',
                'dst_host_serror_rate', 'dst_host_srv_serror_rate',
                'dst_host_rerror_rate', 'dst_host_srv_rerror_rate' ]
```

AvaliadorRegrasFuzzy.py

```
# -*- coding: utf-8 -*-
'''
@author: Jacson RC Silva <jacsonrcsilva@gmail.com>
'''

from Dados_KDDCUP import *
from VariavelLinguistica import AND, OR, NOT

from sys import argv, exit
```

```
if len(argv) < 2:
    print "Utilize: %s arquivo [resultados]" % argv[0]
    print "   utilize \"resultados\" para imprimir as",
    print "respostas ao invés da análise"
    exit(1)

Acerto=0
Erro=0

respostas=False
if len(argv) == 3:
    if argv[2] == 'respostas':
        respostas = True

Normal = 0
Normal_Ataque = 0
Normal_Normal = 0
Ataque = 0
Ataque_Normal = 0
Ataque_Ataque = 0

# percorrendo as linhas do arquivo
for linha in open(argv[1]).readlines():
    #substituindo o \n
    linha  = linha.replace('\n','').replace('.','')
    # transformando em uma lista
    campos = linha.split(',')
    # convertendo os números referentes para float
    for c_idx in [0,4,5,7,8,9,10,12,15,16,17,18,19] + range(22,41):
        campos[ c_idx ] = float(campos[ c_idx ])

    #SE dst_host_srv_count é baixo E flag não é S0 E protocol_type NÃO
é icmp E
    # dst_host_srv_rerror_rate NÃO é alto ENTÃO conexão é Ataque_U2R
    p1  = AND( dst_host_srv_count.mu('MB', campos[32]),
          NOT( flag.mu('S0', campos[3]) ) )
    p2  = AND( p1, NOT( protocol_type.mu('icmp', campos[1]) ) )
    U2R = AND( p2, NOT( dst_host_srv_rerror_rate.mu('A', campos[38]) ))

    # SE ( dst_host_srv_count é baixo OU is_guest_login é verdade ) E
    # flag NÃO é REJ E dst_host_same_srv_rate NÃO é baixo E duration
NÃO é alto
    # ENTÃO record_type é Ataque_R2L
    p1  =  OR( dst_host_srv_count.mu('MB', campos[32]),
               is_guest_login.mu('1', campos[21]) )
    p2  = AND( p1, NOT( flag.mu('REJ', campos[3]) ) )
    p3  = AND( p2, NOT( dst_host_same_srv_rate.mu('MB', campos[33]) ) )
    # duration is in level-4
    p4  = duration.mu('A', campos[0])
    if p4 > 0.0 and p4 < 1.0: p4 = 1.0
    else: p4 = 0.0
    R2L = AND( p3, NOT( p4 ) )

    # SE count NÃO é baxo OU same_srv_rate é baixo ENTÃO
    # record_type é Ataque_DOS
```

```
    p1 = OR( NOT(count.mu('MB', campos[22])),
             same_srv_rate.mu('MB', campos[28]) )
    # Adicionado pelo Autor
    p2 = AND( protocol_type.mu('tcp', campos[1]),
              service.mu('http', campos[2]) )
    p3 = AND( p2, flag.mu('RSTR', campos[3]) )
    DOS = OR( p1,p3 )

    # SE dst_host_same_srv_rate é baixo E flag NÃO é SF OU
    # protocol_type é icmp ENTÃO record_type é Ataque_PRB.
    p1 = AND( dst_host_same_srv_rate.mu('MB', campos[33]),
              NOT( flag.mu('SF', campos[3]) ) )
    PROBE = OR(p1, protocol_type.mu('icmp', campos[1]) )

    # SE (dst_host_srv_count NÃO é baixo OU protocol_type NÃO é tcp) E
    # protocol_type NÃO é icmp ENTÃO conexão é normal
    p1 =  OR( NOT( dst_host_srv_count.mu('MB', campos[32]) ) ,
              NOT( protocol_type.mu('tcp',campos[1]) ) )
    NORMAL = AND( p1, NOT( protocol_type.mu('icmp', campos[1]) ) )

    BACK    = 0.8 if float(campos[4]) == 54540 else 0.0

    # obtendo o maior do ataques
    resp = OR(U2R, OR(R2L, OR(DOS, OR(PROBE,BACK))))

    if respostas:
        print "%.4f %.4f %.4f %.4f %.4f %.4f %.4f %s" %
              ( U2R, R2L, DOS, PROBE, BACK, NORMAL, resp, campos[41] )

    if campos[41] == 'normal':
        Normal+=1
        if resp > NORMAL:
            Normal_Ataque+=1
        else:
            Normal_Normal+=1
    else:
        Ataque+=1
        if resp > NORMAL:
            Ataque_Ataque+=1
        else:
            Ataque_Normal+=1


if respostas: exit(0)

print "_"*60
print "[ Estatísticas %s ]" % (argv[1].split('/')[-1])
print "Total de entradas: ", Normal + Ataque
print "Tráfego Malicioso Detectado (VP):     %d de %d %.0f%%" %
      (Ataque_Ataque,Ataque, float(Ataque_Ataque)/float(Ataque)*100.0)
print "Tráfego Malicioso Não Detectado (FN): %d de %d %.0f%%" %
      (Ataque_Normal,Ataque, float(Ataque_Normal)/float(Ataque)*100.0)
print "Tráfego Normal Detectado (VN):        %d de %d %.0f%%" %
      (Normal_Normal,Normal, float(Normal_Normal)/float(Normal)*100.0)
print "Tráfego Normal Não Detectado (FP):    %d de %d %.0f%%" %
```

```
      (Normal_Ataque,Normal, float(Normal_Ataque)/float(Normal)*100.0)
print "Taxa de Sucesso: %d de %d %.0f%%" %
      ( Ataque_Ataque+Normal_Normal, Normal+Ataque,
      (float(Ataque_Ataque)+float(Normal_Normal))/
       float(Normal+Ataque)*100.0 )
print "_"*60
```

Anexo V – Neuro-Fuzzy

Neuro-Fuzzy.py

```
# -*- coding: utf-8 -*-

import fuzz

tipoTrafego = {}
tipoTrafego.update( {
      'A': fuzz.TriangularFuzzyNumber(100.0, (25.0, 100.0) ),
      'N': fuzz.TriangularFuzzyNumber(  0.0, ( 0.0,  75.0) )  } )

from sys import argv, exit
if len(argv) < 3:
    print "Utilize: %s arqFuzzy arqRNA" % (argv[0])
    print "\targFuzzy: Arquivo com as saídas Fuzzy"
    print "\targRNA  : Arquivo com as saídas da RNA"
    exit(1)

arq  = argv[1]
arq2 = argv[2]

LIMITE = 26.7
acerto=0
erro=0
dados  = open(arq).readlines()
dados1 = open(arq2).readlines()
Normal = 0
Normal_Ataque = 0
Normal_Normal = 0
Ataque = 0
Ataque_Normal = 0
Ataque_Ataque = 0

for pos in range( len(dados) ):
    dados [pos] = dados [pos].replace('\n','')
    dados1[pos] = dados1[pos].replace('\n','')

    D = [ float(dados1[pos].split()[0]) ] + \
        [ float(x) for x in dados[pos].split()[:5] ]
    ataque = dados[pos].split()[-1]

    # ____ COG ______
    MAX_N= max( float(dados1[pos].split()[-1]),
                1-float(dados1[pos].split()[-2]) )

    MAX_A=0
    for i in range( len(D) ):
```

```
        MAX_A = max( MAX_A, D[i] )

    num=0
    den=0
    for x in range(5,101,5):
        mu   = max( min ( tipoTrafego['A'].mu(x), MAX_A ),
                    min ( tipoTrafego['N'].mu(x), MAX_N ) )
        num += x*mu
        den += mu

    if den == 0: resultado = 0.0
    else:        resultado = num/den

    TIPO = ''
    if tipoTrafego['N'].mu(resultado) > tipoTrafego['A'].mu(resultado):
        TIPO = 'normal'
    else:
        TIPO = 'ataque'

    # ____ COG ______

    if ataque == 'normal':
        Normal += 1
        if TIPO == 'normal':
            Normal_Normal+=1
        else:
            Normal_Ataque+=1
    else:
        Ataque += 1
        if TIPO == 'normal':
            Ataque_Normal+=1
        else:
            Ataque_Ataque+=1

print "_"*60
print "[ Estatísticas ]"
print "Total de entradas: ", Normal + Ataque
print "Tráfego Malicioso Detectado (VP):     %d de %d %.0f%%" %
       (Ataque_Ataque,Ataque, float(Ataque_Ataque)/float(Ataque)*100.0)
print "Tráfego Malicioso Não Detectado (FN): %d de %d %.0f%%" %
       (Ataque_Normal,Ataque, float(Ataque_Normal)/float(Ataque)*100.0)
print "Tráfego Normal Detectado (VN):        %d de %d %.0f%%" %
       (Normal_Normal,Normal, float(Normal_Normal)/float(Normal)*100.0)
print "Tráfego Normal Não Detectado (FP):    %d de %d %.0f%%" %
       (Normal_Ataque,Normal, float(Normal_Ataque)/float(Normal)*100.0)
print "Taxa de Sucesso: %d de %d %.0f%%" %
       (Ataque_Ataque+Normal_Normal, Normal+Ataque,
       (float(Ataque_Ataque)+float(Normal_Normal))/
        float(Normal+Ataque)*100.0)
print "_"*60
```

ColherResultadosFuzzy.py

```
# -*- coding: utf-8 -*-
'''
Script para obter a saída dos conjuntos nebulosos

@author: Jacson RC Silva <jacsonrcsilva@gmail.com>
'''

from Fuzzy.Dados_KDDCUP      import *
from Fuzzy.VariavelLinguistica import AND, OR, NOT

from sys import argv, exit

if len(argv) != 2:
    print "Utilize: %s arqKDD.txt" % argv[0]
    exit(1)

# Percorrendo as linhas do arquivo KDD
for linha in open( argv[1] ).readlines():
    # retirando o \n
    linha  = linha.replace('\n','').replace('.','')
    # obtendo os campos da linha
    campos = linha.split(',')
    # convertendo os valores referentes para float
    for c_idx in [0,4,5,7,8,9,10,12,15,16,17,18,19] + range(22,41):
        campos[ c_idx ] = float(campos[ c_idx ])

    #SE dst_host_srv_count é baixo E flag não é S0 E protocol_type NÃO
é icmp E
    # dst_host_srv_rerror_rate NÃO é alto ENTÃO conexão é Ataque_U2R
    p1  = AND( dst_host_srv_count.mu('MB', campos[32]),
               NOT( flag.mu('S0', campos[3]) ) )
    p2  = AND( p1, NOT( protocol_type.mu('icmp', campos[1]) ) )
    U2R = AND( p2, NOT( dst_host_srv_rerror_rate.mu('A', campos[38]) )
)

    # SE ( dst_host_srv_count é baixo OU is_guest_login é verdade ) E
    # flag NÃO é REJ E dst_host_same_srv_rate NÃO é baixo E duration
NÃO é alto
    # ENTÃO record_type é Ataque_R2L
    p1  =  OR( dst_host_srv_count.mu('MB', campos[32]),
               is_guest_login.mu('1', campos[21]) )
    p2  = AND( p1, NOT( flag.mu('REJ', campos[3]) ) )
    p3  = AND( p2, NOT( dst_host_same_srv_rate.mu('MB', campos[33]) ) )
    # duration is in level-4
    p4  = duration.mu('A', campos[0])
    if p4 > 0.0 and p4 < 1.0: p4 = 1.0
    else: p4 = 0.0
    R2L = AND( p3, NOT( p4 ) )

    # SE count NÃO é baxo OU same_srv_rate é baixo
    # ENTÃO record_type é Ataque_DOS
    p1 = OR( NOT(count.mu('MB', campos[22])),
```

```
                same_srv_rate.mu('MB', campos[28]) )
    # Adicionado pelo Autor
    p2 = AND( protocol_type.mu('tcp', campos[1]),
              service.mu('http', campos[2]) )
    p3 = AND( p2, flag.mu('RSTR', campos[3]) )
    DOS = OR( p1,p3 )

    # SE dst_host_same_srv_rate é baixo E flag NÃO é SF OU
protocol_type é icmp
    # ENTÃO record_type é Ataque_PRB.
    p1 = AND( dst_host_same_srv_rate.mu('MB', campos[33]),
              NOT( flag.mu('SF', campos[3]) ) )
    PROBE = OR(p1, protocol_type.mu('icmp', campos[1]) )

    # SE ( dst_host_srv_count NÃO é baixo OU protocol_type NÃO é tcp )
E
    # protocol_type NÃO é icmp ENTÃO conexão é normal
    p1 =  OR( NOT( dst_host_srv_count.mu('MB', campos[32]) ) ,
              NOT( protocol_type.mu('tcp',campos[1]) ) )
    NORMAL = AND( p1, NOT( protocol_type.mu('icmp', campos[1]) ) )

    # adicionado pelo autor
    BACK = 0.8 if float(campos[4]) == 54540 else 0.0

    resp = OR(U2R, OR(R2L, OR(DOS, OR(PROBE,BACK))))

    print "%.4f %.4f %.4f %.4f %.4f %.4f %.4f %s" %
          ( U2R, R2L, DOS, PROBE, BACK, NORMAL, resp, campos[41] )
```

ColherResultadosRNA.py

```
# -*- coding: utf-8 -*-
'''
Script para obter a saída de uma Rede Neural

@author: Jacson RC Silva <jacsonrcsilva@gmail.com>
'''

from sys import argv, exit

if len(argv) != 3:
    print 'Utilize: %s RedeNeural.net arquivoKDD.train' % argv[0]
    exit(1)

rnafilename = argv[1]
filename    = argv[2]

from pyfann import libfann
RNA = libfann.neural_net()
RNA.create_from_file( rnafilename )

DadosTeste = libfann.training_data()
```

```
DadosTeste.read_train_from_file( filename )

entrada = DadosTeste.get_input()
saida   = DadosTeste.get_output()
Normal = 0
Normal_Ataque = 0
Normal_Normal = 0
Ataque = 0
Ataque_Normal = 0
Ataque_Ataque = 0
for i in range( len(entrada) ):
    resposta = RNA.run(entrada[i])
    print "%.4f %.4f" % (resposta[0], resposta[1])

DadosTeste.destroy_train()
RNA.destroy()
```

Anexo VI – Exemplo da utilização do LibPcap e Python

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# adaptado de /usr/share/doc/python-pcap/testsniff.py

import getopt, sys
try:    import dpkt
except: raise Exception("Instale o pacote python-dpkt")
try:    import pcap
except: raise Exception("Instale o pacote python-pypcap")

def ajuda():
    print 'Utilize: %s [-i device] [padrão]' % sys.argv[0]
    sys.exit(1)

def main():
    opts, args = getopt.getopt(sys.argv[1:], 'i:h')
    name = None
    for o, a in opts:
        if o == '-i': name = a
        else: ajuda()

    pc = pcap.pcap(name)
    pc.setfilter(' '.join(args))
    decode = { pcap.DLT_LOOP:dpkt.loopback.Loopback,
               pcap.DLT_NULL:dpkt.loopback.Loopback,
               pcap.DLT_EN10MB:dpkt.ethernet.Ethernet }[pc.datalink()]
    try:
        print 'ouvindo de %s: %s' % (pc.name, pc.filter)
        for ts, pkt in pc:
            print ts, `decode(pkt)`
    except KeyboardInterrupt:
        nrecv, ndrop, nifdrop = pc.stats()
        print '\n%d pacotes recebidos pelo filtro' % nrecv
        print '%d pacotes bloqueados pelo kernel' % ndrop

if __name__ == '__main__':
    main()
```