UNIVERSIDADE SALVADOR - UNIFACS
MESTRADO EM SISTEMAS E COMPUTAÇÃO

EUSAM PEREIRA DE SOUZA

ESTUDO SOBRE SISTEMA DE DETECÇÃO DE INTRUSÃO POR ANOMALIAS:
UMA ABORDAGEM UTILIZANDO REDES NEURAIS

Salvador
2008

EUSAM PEREIRA DE SOUZA

ESTUDO SOBRE SISTEMA DE DETECÇÃO DE INTRUSÃO POR ANOMALIAS:
UMA ABORDAGEM UTILIZANDO REDES NEURAIS

Dissertação apresentada ao Curso de Mestrado em Sistemas e Computação, da Universidade Salvador – UNIFACS, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

Orientador: Prof. Dr. José Augusto Suruagy Monteiro.

Salvador
2008

FICHA CATALOGRÁFICA
(Elaborada pelo Sistema de Bibliotecas da Universidade Salvador – UNIFACS

Souza, Eusam Pereira de
Estudo sobre sistema de detecção de intrusão anolias: uma abordagem utilizando redes neurais/Eusam Pereira de Souza - .Salvador, 2008.
149 f.

Dissertação (Mestrado) - Universidade Salvador – UNIFACS. Mestrado em Sistemas de Computação, 2008.
Orientador: Prof. Dr. Jose Augusto Suruagy Monteiro

1. Sistemas de informação. 2. Sistema de Detecção de Intrusão - SDI 3. Redes Neurais Artificiais.4. Anomalias de tráfego. I. Monteiro Neto, Manoel Gomes de, orient. II. Título.

CDD: 005.74068

DEPARTAMENTO DE ENGENHARIA E ARQUITETURA

MESTRADO EM SISTEMAS E COMPUTAÇÃO

EUSAM PEREIRA DE SOUZA

ESTUDO SOBRE SISTEMA DE DETECÇÃO DE INTRUSÃO POR ANOMALIAS:
UMA ABORDAGEM UTILIZANDO REDES NEURAIS

Dissertação aprovada como requisito para obtenção do título de Mestre em Sistemas e Computação, Universidade Salvador - UNIFACS, pela seguite banca examinadora:

Prof. Dr. José Augusto Suruagy Monteiro (Orientador) _____
Universidade Salvador - UNIFACS

Prof. Dr. Joberto Sérgio Barbosa Martins _____
Universidade Salvador - UNIFACS

Profª. Drª. Fabiana Cristina Bertoni _____
Universidade Estadual de Feira de Santana - UEFS

Salvador, 18 de dezembro de 2008.

A todos que ajudaram na efetivação deste trabalho.

AGRADECIMENTOS

Ao Programa de Pós-Graduação em Sistemas e Computação, pela oportunidade de realização de trabalhos em minha área de pesquisa.

Ao Professor José Augusto Suruagy Monteiro, pelo seu auxílio e orientação nas tarefas desenvolvidas durante a realização deste trabalho.

RESUMO

A probabilidade de ocorrerem ataques em redes de computadores, tendo como conseqüência o surgimento de anomalias no tráfego, aumentou com o crescimento da complexidade das redes de computadores. A identificação, análise e tratamento dessas anomalias no menor tempo possível tornaram-se parte essencial e mais importante no gerenciamento de redes. O principal objetivo de um sistema de detecção de intrusão é ser capaz de alcançar uma alta taxa de acertos e uma baixa taxa de alarmes falsos. IDS por anomalias utiliza técnicas que procuram identificar diferenças baseadas na comparação de padrões de tráfego considerados normais, com padrões anômalos. Utilizando a capacidade de generalização das redes neurais, foi possível realizar a classificação de ataques existentes na base de registro de conexões disponibilizado pela Competição Internacional de Mineração de Dados KDD Cup 1999, inclusive para ataques ainda não conhecidos pela Rede Neural, durante a etapa de treinamento. Foram feitas comparações entre os resultados obtidos neste trabalho, com os obtidos pelo vencedor da competição, para avaliar a eficácia do método. Em seguida, foram apresentadas à Rede Neural apenas as características consideradas relevantes dos registros de conexões, com base nos resultados do trabalho de Zincir-Heywood et al (2005), ou seja, aquelas características extremamente úteis para uma determinação rigorosa das classes de ataque a que aquele registro de conexão pertence, com o objetivo de reduzir o tempo de classificação da Rede. Com isso, a Rede Neural teve seu tamanho reduzido, diminuindo o tempo de classificação dos ataques, sem contudo comprometer sua taxa de acertos. Ainda com o objetivo de reduzir o tempo de aprendizagem e classificação da Rede Neural, de modo a viabilizar o uso dessa técnica na detecção em tempo real de intrusões em redes de computadores, foram utilizadas algumas heurísticas disponíveis no livro de Haykin (2001), para melhorar o desempenho da rede.

Palavras-chave: Sistema de Detecção de Intrusão - SDI. Redes Neurais Artificiais. Anomalias de tráfego.

ABSTRACT

The probability of occurring attacks in computers networks, and as a consequence the emergence of traffic anomalies, increased with the complexity growth of computer networks. The identification, analysis, and treatment of these anomalies in the shortest time possible have become more important and an essential part in network management. The main goal of an intrusion detection system (IDS) is to be able to achieve a high hit and a low false alarm rates. Anomaly based IDS using techniques that seek to identify differences based on comparison of traffic patterns deemed normal, with anomalous patterns. Using the generalization ability of neural networks, it was possible to make the classification of attacks present on the connection record base, made available by the International Data Mining Competition KDD Cup 1999, including attacks not yet known by the neural network, during its training stage. The results obtained in this work were compared with those from the competition winner, to assess the effectiveness of the method. Then, it was submitted to the neural network only the connection record characteristics considered relevant, based on the results obtained by Zincir-Heywood et al (2005), i.e., those features extremely useful for an accurate determination of the attack class that a connection record belongs to. In this way, the neural network had its size reduced, reducing the attacks classification time without compromising its hit rate. Furthermore, with the objective of reducing the neural network learning and classification times, in order to facilitate the use of this technique in the real-time intrusion detection, it has been used some heuristics available in the literature to improve the neural network performance.

Keywords: Intrusion Detection System – IDS. Artificial Neural Nets. Traffic Anomalies.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

As redes de computadores têm apresentado aumento significativo em tamanho e importância no decorrer dos anos. A conseqüência desse avanço é a maior exigência por confiabilidade e a necessidade de garantia de Qualidade de serviço - *Quality of Service* (QoS).

Assim, a probabilidade de surgimento de anomalias de tráfego causadas por falhas e ataques aumentou com o crescimento da complexidade, pondo em risco a garantia de confiabilidade. A identificação, análise e tratamento dessas anomalias no menor tempo possível tornaram-se parte essencial do cotidiano das redes e fez da detecção de anomalias uma das áreas mais importantes do gerenciamento de redes.

1.1 DEFINIÇÕES

Anomalias em redes de computadores se referem, tipicamente, a circunstâncias onde as operações da rede se distanciam de um padrão considerado normal. Essas anomalias são caracterizadas por mudanças bruscas correlacionadas nos diversos conjuntos de dados monitorados. Essas mudanças bruscas ocorrem antes ou até mesmo durante o evento anômalo (ZARPELÃO, 2004).

Detecção de anomalias é definida como a tarefa de distinguir ou descobrir dados que estejam se diferenciando de um comportamento normal e esperado para o conjunto ao qual ele pertence. Seguindo a mesma linha, a detecção de anomalias é definida como a tarefa de determinar a discrepância entre o comportamento realmente encontrado e o comportamento esperado no tráfego de rede (ZARPELÃO, 2004).

Intrusão é definida como o conjunto de ações desencadeadas pelo intruso que compromete a estrutura básica da segurança da informação de um sistema informático: integridade, confidencialidade e disponibilidade. Enquanto que intruso é definido como qualquer usuário ou entidade que invade um sistema sem que para tal tenha obtido autorização. A detecção de um intruso é uma tarefa complexa, uma vez que este pode ter em seu poder a identificação de um usuário legítimo ou um comportamento muito semelhante ao de um usuário legítimo. Intrusos podem ser classificados como internos e externos (SILVA et al., 2001).

Intrusos externos são todos os usuários que não pertencem ao sistema e que implementam ações de intrusão ao mesmo, enquanto que intrusos internos são todos os usuários que têm algum tipo de autorização de acesso legítimo ao sistema e que, normalmente ultrapassando os seus direitos de acesso, tentam realizar ações de intrusão a esse mesmo sistema (SILVA et al., 2001).

Detecção de intrusão significa detectar o uso não autorizado ou ataques em um sistema ou em uma rede. Um sistema de detecção de intrusão *Intrusion Detection System* (IDS) inspeciona o conteúdo do tráfego da rede para procurar e desviar possíveis ataques, exatamente como um pacote de software antivírus inspeciona o conteúdo de arquivos, anexos de e-mail, etc., para procurar assinaturas de vírus, padrões que correspondem a software danoso conhecido, ou possíveis ações maldosas, padrões de comportamento que são no mínimo suspeitos, se não completamente inaceitáveis (CASWELL et al., 2003; MONZON, 2007).

Sistemas detectores de intrusão têm por finalidade oferecer um mecanismo que venha a reduzir a possibilidade de intrusão. Porém, este sistema sozinho não oferece nenhuma garantia de eliminar esta possibilidade (CASWELL et al., 2003; MONZON, 2007).

Um IDS tornou-se para muitos profissionais de segurança o segundo produto a ser adquirido logo depois do *firewall*. *Firewall* é o nome dado ao dispositivo de uma rede de computadores que tem por objetivo aplicar uma política de segurança a um

determinado ponto de controle da rede. Sua função consiste em regular o tráfego de dados entre redes distintas e impedir a transmissão e/ou recepção de acessos nocivos ou não autorizados de uma rede para outra. Um *firewall* é utilizado para prevenção, um IDS para detecção. O *firewall* previne alguns ataques, mesmo sem qualquer gestão ou supervisão. O IDS pode detectar alguns ataques, mas muitas vezes é necessária a intervenção de um humano para interpretar e responder a alarmes (CASWELL et al., 2003; MONZON, 2007).

1.2 MOTIVAÇÃO

O desenvolvimento e aperfeiçoamento de sistemas de detecção de intrusão são motivados por diversas razões, tais como:

a) a maioria dos sistemas computacionais utilizados possui alguma falha de segurança que pode ser explorada por usuários mal intencionados. Pesquisar e corrigir todas as potenciais falhas de segurança não é viável técnica e comercialmente.

b) inviabilidade de substituição de sistemas com falhas de segurança por versões mais seguras.

c) desenvolver sistemas completamente livres de falhas de segurança, mesmo que isso seja uma meta crítica do projeto, é considerado uma tarefa virtualmente impossível.

d) sistemas altamente seguros podem ser comprometidos por usuários internos que abusem de seus privilégios.

Há muitos IDSs desenvolvidos nos últimos anos. No entanto, a maior parte das ferramentas IDS *freeware* comerciais são baseadas em assinaturas. Esses instrumentos só podem detectar ataques conhecidos anteriormente descritos pelas

suas correspondentes assinaturas. A base de dados de assinaturas deverá ser mantida e atualizada periodicamente com o surgimento de novos ataques.

Alarmes falsos são causados por algumas situações que mostram um desvio de comportamento, mas não causam qualquer tipo de problema explícito no funcionamento da rede. Em geral, essas situações são rajadas que compõem naturalmente o tráfego de rede. Falsos positivos devem ser evitados, pois acabam desviando a atenção dos administradores de rede e atrapalhando o objetivo dos sistemas de detecção de anomalias. Os falsos positivos acontecem quando pacotes normais são identificados como tentativas de ataques, distorcendo a análise. Os falsos negativos ocorrem quando pacotes provenientes de ataques não são identificados, tornando um grande problema. É importante ressaltar que, a reunião de fontes de dados diferentes na correlação de alarmes, pode ajudar na diminuição da quantidade de alarmes falsos (CASWELL et al., 2003; MONZON, 2007).

O presente trabalho pretende utilizar-se das Redes Neurais Artificiais para Classificação dos tipos de intrusões existentes em uma base de registros de conexões, utilizada como *benchmark*. Além disso, pretende testar o uso de heurísticas que visam melhorar o desempenho das redes neurais, e utilizar os resultados do trabalho de Silva e outros (2004), para selecionar apenas as características consideradas relevantes dos registros de conexões, visando diminuir o tamanho da Rede Neural e torná-la mais rápida na detecção de ataques.

O desenvolvimento dos modelos de Redes Neurais Artificiais surgiu como uma tentativa inicial de reproduzir o alto desempenho do cérebro humano em tarefas cognitivas extremamente complexas. Uma Rede Neural Artificial é uma estrutura composta de unidades processadoras simples, distribuídas e paralelas, que tem o propósito de armazenar conhecimento por meio de mecanismos empíricos e torná-lo disponível para o uso.

A vantagem de se utilizar das Redes Neurais na classificação de ataques em padrões de tráfego de rede está na sua capacidade de generalização, onde se espera que, no presente trabalho, as redes neurais consigam detectar novos ataques, mantendo uma alta taxa de acertos.

Para os cenários abordados, será possível avaliar a capacidade de generalização das Redes Neurais na identificação de padrões de ataques desconhecidos, utilizar e analisar o uso de heurísticas que visam melhorar o desempenho das redes neurais. Além disso, será possível utilizar os resultados do trabalho de Silva e outros (2004), para selecionar apenas as características consideradas relevantes dos registros de conexões, visando diminuir o tamanho da Rede Neural e viabilizar o uso dessa técnica, em trabalhos futuros, na detecção de intrusões em tempo real.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado da seguinte forma: no capítulo 2 é apresentada uma introdução teórica sobre Sistemas de Detecção de Intrusão, tipos de ataques e de intrusões, assim como os métodos e técnicas utilizadas atualmente para detecção de intrusos; o capítulo 3 mostra conceitos teóricos relacionados às Redes Neurais Artificiais; o 4 apresenta as vantagens na utilização de Redes Neurais na classificação de padrões; o 5 aborda com detalhes a proposta de solução para um Sistema de Detecção de Intrusão baseados em Redes Neurais, para redes TCP/IP, assim como as heurísticas apontadas por Haykin (2001) e utilizadas no presente trabalho, para melhorar o desempenho da rede e possibilitar a aplicação futura deste método na detecção em tempo real de ataques em redes de computadores; o 6 mostra os resultados obtidos e análises sobre tais resultados, assim como a utilização apenas das características consideradas relevantes dos registros de conexões, com base nos resultados do trabalho de Silva e outros (2004), ou seja, aquelas características extremamente úteis para uma determinação rigorosa das classes de ataque a que aquele registro de conexão pertence. Essa abordagem tem

como objetivo reduzir o tempo de aprendizagem e classificação da Rede Neural, de modo a viabilizar o uso dessa técnica na detecção em tempo real de intrusões em redes de computadores; no capítulo 7 são apresentadas as conclusões e apontados os aspectos que poderão ser tratados em trabalhos futuros; no 8 são apresentadas as referências bibliográficas consultadas; e no 9, um apêndice do trabalho, é mostrado o código dos scripts utilizados no SQL para tratamento dos registros de conexões da base de dados utilizada como *benchmark*.

2 ATAQUES E INTRUSÕES EM REDES DE COMPUTADORES

Este capítulo trata dos aspectos teóricos relativos a ataques e intrusões em redes de computadores, aos sistemas detectores de intrusão e ao tráfego de redes e anomalias de tráfego.

2.1 ASPECTOS TÉCNICOS DO IDS

Apesar das redes de computadores utilizarem algoritmos cooperativos, para que um invasor consiga comprometer o funcionamento da rede, ele precisa ter acesso físico à mesma. Esses algoritmos se baseiam na boa fé de várias partes da rede para funcionar, como os protocolos MAC.

Para construir uma rede realmente segura, é essencial que haja um esquema de detecção de intrusões quando elas ocorrerem, para que medidas de resposta a essas intrusões possam ser tomadas.

Uma invasão pode ser definida como qualquer conjunto de ações que tem como intuito o comprometimento da integridade, confidencialidade ou disponibilidade de um recurso (FRANCESQUINI, 2004).

Os sistemas detectores de intrusão têm por finalidade oferecer um mecanismo que venha a reduzir a possibilidade de intrusão, através da antecipação e acompanhamento dos ataques, bem como auxiliar a compreensão das estratégias utilizadas pelos atacantes e códigos de computadores maliciosos (HEINEN; OSÓRIO, 2006).

As intrusões são um dos principais ataques atualmente utilizados, conforme estatísticas disponíveis em organismos internacionais voltados ao estudo de incidentes de segurança. O *Malaysian Computer Emergency Response Team* (MyCERT) é um organismo internacional que fornece referências para a comunidade da Internet, sobre incidentes de segurança em redes de computadores e dos métodos de prevenção. Dados estatísticos de ataques, obtidos no MyCERT, são mostrados na figura 1.

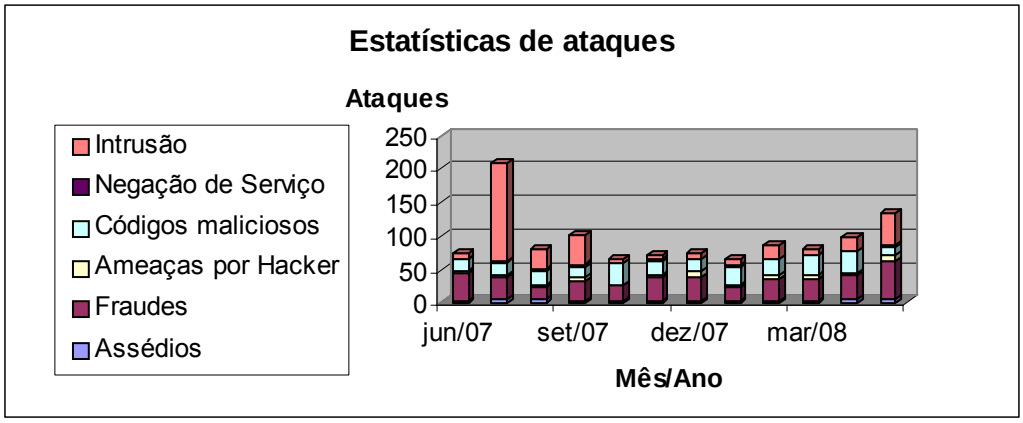

Figura 1 - Estatísticas de ataques.
Fonte: http://www.mycert.org.my

A figura 2 mostra a estatística de incidentes reportados ao Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (CERT/BR), onde se pode perceber o aumento significativo no total de incidentes registrados nos últimos anos, evidenciando assim a necessidade de se desenvolver mecanismos cada vez mais eficazes na detecção de ataques e intrusões em redes de computadores.

As seguintes atividades são realizadas pelos IDS em graus variáveis de precisão:

a) monitoração e análise de atividades de sistema e dos utilizadores;
b) realização de auditorias à infra-estrutura de rede, às falhas e às vulnerabilidades do sistema;
c) identificação do modelo de atividades do sistema, de forma a reconhecer sinais de ataques e de alertas;

d) realização de uma análise estatística a um modelo de comportamento anômalo;
e) avaliação da integridade dos dados e dos sistemas; e
f) realização de auditorias de gestão aos sistemas de informação, e de reconhecimento do comportamento dos usuários que desobedecem à política de segurança da organização.

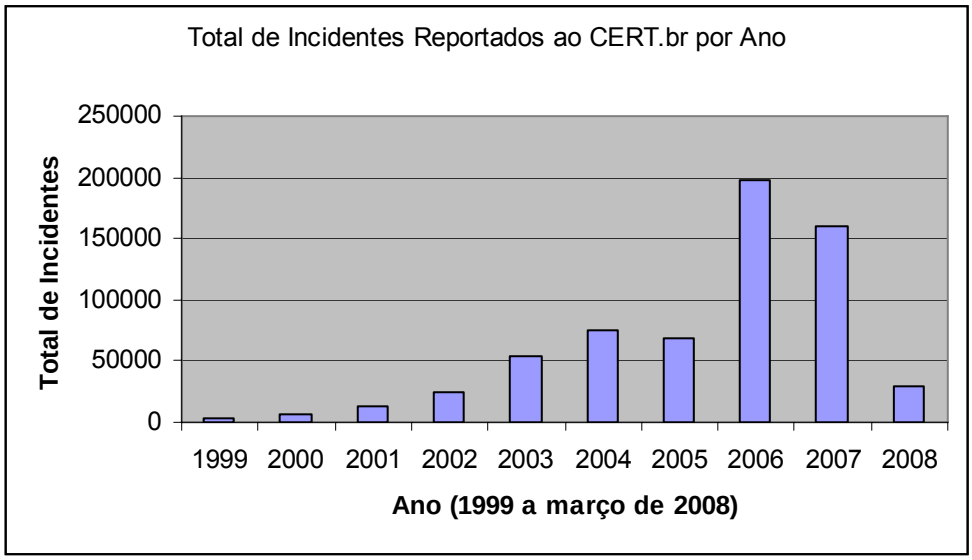

Figura 2 - Incidentes reportados ao CERT.br – 1999 a 2008.
Fonte: http://www.cert.br.

Existem IDS capazes de reagir quando da detecção de ações não autorizadas, de forma a conter ou parar o causador, por exemplo, desligando a conexão à rede. Um exemplo elucidativo é ao ser detectado uma varredura de portas (*portscan*), que visa encontrar portas abertas que possam ser aproveitadas por cavalos de tróia (*trojan horses*), o IDS poderá bloquear conexões vindas desse endereço IP.

O *portscan* ou varredor de portas é o processo de acessar portas TCP ou UDP de um sistema alvo, com o intuito de identificar a existência ou não de um serviço atrelado àquela porta. Logo, varrer uma porta é a arte de identificar, externamente, os serviços disponíveis para cada interface. Se tais serviços estiverem mal

configurados ou com falhas de segurança, estes podem comprometer toda a rede do sistema, permitindo a um usuário não autorizado ter acesso a informações restritas.

As técnicas de detecção de intrusões pressupõem que o comportamento e as atividades de um usuário ou programa legítimo são diferentes do comportamento e das atividades de um invasor. Elas funcionam com base na análise e classificação dos comportamentos e atividades de um usuário como legítimos ou ilegítimos. Ou seja, essas técnicas também assumem que o comportamento e as atividades de um usuário são observáveis e possíveis de serem analisadas.

As técnicas utilizadas pelos sistemas de detecção de intrusões podem ser divididas, basicamente, em duas categorias: as de detecção de abusos e as de detecção de anomalias.

As técnicas de detecção de abusos são as mais comuns e baseiam-se em padrões conhecidos de ataques, além dos pontos fracos conhecidos do sistema, para tentar reconhecer um ataque. Um exemplo de padrão é: o usuário *X* tentou por *n* vezes se conectar à rede com uma senha inválida nos últimos *t* minutos. A vantagem deste tipo de técnica é que ela é muito eficaz e precisa no reconhecimento de ataques cujos padrões já são conhecidos. Em contrapartida, não é capaz de perceber se o sistema está sob ataque caso o mesmo seja “inovador”. Portanto, ela pode não perceber que o sistema está sendo atacado caso os seus padrões não estejam na lista de padrões conhecidos.

Por outro lado, as técnicas de detecção de anomalias trabalham capturando dados sobre o comportamento e as atividades do usuário, e através da análise e comparação deles com os perfis pré-cadastrados, determinam se o sistema está ou não sob ataque. Por exemplo, se uma secretária que geralmente só usa programas de edição de texto repentinamente começar a usar o *GNU Compiler Collection (*GCC*),* conjunto de compiladores de linguagens de programação –, ela claramente está fugindo do padrão pré-estabelecido.

A vantagem desse tipo de técnica está no fato de que o sistema não precisa ser alimentado a priori com os padrões de ataques dos quais ele deve ser capaz de se defender, e por isso é capaz, inclusive, de detectar ataques “inovadores”, já que analisa o perfil de uso do sistema. A desvantagem dessa técnica é que ela pode ter uma porcentagem relativamente alta de alarmes falsos, além de não ser capaz de dizer, ao certo, qual o tipo de ataque está ocorrendo. Também exige que o sistema de detecção de intrusos seja treinado para aprender os perfis de uso normal da rede.

Mais detalhes sobre detecção por abuso (*misuse detection*) e por anomalias (*anomaly detection*) serão vistos na seção 2.3.

2.2 CLASSIFICAÇÃO DE IDS QUANTO À FONTE DE INFORMAÇÃO

Os Sistemas de Detecção de Intrusão, segundo Caswell e outros (2003) e Francesquini (2004), podem ser classificados quanto à fonte de informação. Vejamos:

a) **Sistema de Detecção de Intrusão para *Host* ou nós** (*Host Intrusion Detection System* - HIDS) são sistemas que monitoram um determinado computador (*host*). Normalmente, estes sistemas monitoram variáveis como a taxa de utilização de processador, chamadas de funções de sistemas e modificações em arquivos críticos, procurando por atividades suspeitas. Os HIDS analisam uma máquina específica, recebendo pacotes com destino a esta máquina, na qual está instalado o IDS. Ficam em cada uma das unidades constituintes da rede trabalhando em conjunto com o sistema operacional para monitorar o comportamento e as atividades dos usuários e programas.

b) **Sistema de Detecção de Intrusão para Redes** (*Network Intrusion Detection System* - NIDS) são sistemas que monitoram segmentos de redes de

computadores. Normalmente, estes sistemas capturam todos os pacotes de dados que trafegam em determinado segmento da rede procurando por atividades suspeitas. Assim, um NIDS monitora todo o tráfego do segmento de rede, recebendo todos os pacotes, tendo a capacidade de monitorar a atividade em diversas máquinas. Geralmente ficam nos *gateways* da rede, capturando os pacotes e analisando-os em busca de comportamentos que possam evidenciar uma invasão.

c) **Sistema de Detecção de Intrusão Distribuído** (*Distributed Intrusion Detection System* - DIDS) são sistemas que funcionam com sensores remotos, sendo esses um NIDS ou um HIDS, com o gerenciamento e armazenamento de logs ocorrendo em uma única máquina centralizada. Um exemplo de DIDS é o sistema desenvolvido por Shyu e outros (2007), onde vários agentes são responsáveis por coletar atividades de tráfego na rede, na camada de *host* da solução, e na camada de classificação, os agentes, mais especializados que os da camada de *hos* são responsáveis pela classificação dos tipos de ataques.

2.3 CLASSIFICAÇÃO DE IDS QUANTO ÀS TÉCNICAS DE DETECÇÃO

A detecção por abuso, por uso indevido ou por conhecimento (*misuse detection*), tem o objetivo de comparar eventos do sistema em tempo real a cenários de ataques generalizados, ou seja, a modelos de detecção de abusos. Diferente da detecção por abuso, a detecção por anomalia ou por comportamento (*anomaly detection*) compara os eventos correntes do sistema ou do tráfego de rede a perfis de atividades normais, ou seja, aos modelos de detecção por anomalia. Nesta abordagem, as anomalias são consideradas como possíveis intrusões.

Sistemas de detecção de intrusos por abuso (*misuse detection*) são sistemas que procuram por padrões de ataques e intrusões previamente conhecidos. São também

conhecidos como sistemas de detecção baseado em padrões de assinaturas de ataques ou em sistemas de detecção de intrusão baseado em conhecimento. A premissa básica dos sistemas que se enquadram nesta classe é que existem ataques com características precisas, bem definidas e que podem ser facilmente codificadas em um sistema especialista. Sistemas de detecção dessa classe normalmente possuem uma base de assinaturas de ataques conhecidos que deve ser constantemente atualizada à medida que novos ataques sejam descobertos.

Um exemplo de assinatura para um Sistema de Detecção de Intrusão baseado em abuso para detecção de um ataque de negação de serviço denominado "*Ping of Death*"[1] consiste em considerar como ataque todo pacote "*ICMP Echo Request*" maior do que 64.000 bytes.

A principal desvantagem dos sistemas de detecção de intrusos por abuso é a sua incapacidade de detectar novos ataques, ou mesmo, ataques que ainda não façam parte da sua base de assinaturas e padrões. Este é o principal fator motivador para pesquisa e desenvolvimento de sistemas de detecção de intrusões baseado em anomalias, pois esses sistemas não usam bases de assinaturas, sendo, portanto possível a detecção de novos ataques.

Sistemas de detecção de intrusão por anomalia, também chamados sistemas de detecção de intrusão baseados em comportamento, procuram determinar ou criar modelos que representem o comportamento normal ou esperado do sistema computacional ou rede em análise e alertam sempre que forem encontrados desvios no comportamento esperado. A premissa básica destes sistemas é que atividades de intrusão ou ataque fazem parte do subconjunto composto por atividades anômalas. Idealmente, o conjunto de atividades maliciosas será igual ao conjunto de atividades anômalas. Nesta situação o sistema não gerará nenhum falso-positivo e nenhum falso-negativo.

[1] Ping da morte – envio de um pacote ICMP Echo Request muito grande que causa indisponibilidade dos serviços de rede em vários sistemas operacionais devido a um *bug* ou erro de programação presente nos softwares vulneráveis.

Na prática, as seguintes situações podem ocorrer com probabilidade diferente de zero:

a) atividade intrusiva com detecção de atividade não intrusiva pelo IDS: esta situação é extremamente grave, pois gera a situação de falso-negativo, ou seja, o sistema não detecta um ataque ou intrusão;
b) atividade não intrusiva e detecção de atividade não intrusiva pelo IDS: situação denominada positivo-negativo, onde ocorre uma atividade normal e não intrusiva, portanto, o sistema corretamente não irá detectar ataque ou intrusão;
c) atividade não intrusiva e detecção de atividade intrusiva: situação denominada falso-positivo. O sistema alerta indicando uma intrusão que na verdade não ocorreu; e
d) atividade intrusiva e detecção de atividade intrusiva: situação ideal em que o sistema detecta corretamente o ataque ou intrusão.

A abordagem de detecção por anomalia apresenta como primeira grande vantagem, a capacidade de detectar novos tipos de ataques, visto que quaisquer novos ataques diferem de comportamento normal.

Segundo Silva e outros (2004), são vantagens da detecção por anomalias:

a) possibilidade de detectar ataques desconhecidos e complexos, não dependendo de uma base que armazene todos os ataques e vulnerabilidades possíveis, visto que a construção dos modelos de comportamento normal não requer que os padrões de ataque sejam rotulados, o que implica na possibilidade de uso de técnicas não supervisionadas na construção dos modelos;
b) produção de informações sobre padrões de ataques novos, para estudos e levantamentos estatísticos;
c) esforço de manutenção reduzido, sem a necessidade de atualizações freqüentes da base;
d) dependência de plataforma reduzida; e

e) detecção de abusos de privilégios facilitada.

A detecção por anomalias também apresenta problemas. A principal limitação é a alta taxa de alarmes falsos que ela produz. Isso ocorre porque nem toda atividade não usual é ilegítima ou representa um ataque. Assim, comportamentos legítimos do sistema podem ser equivocadamente identificados como anomalias, e assinalados como possíveis intrusões.

Outras desvantagens dessa técnica compreendem:

a) dificuldade de configuração e ajuste do limiar do objeto considerado anômalo ou não, em que os modelos do comportamento normal são construídos sobre os dados históricos de tráfego da rede;

b) baixo desempenho, principalmente devido a grande quantidade de cálculos complexos exigidos; e

c) dificuldade no tratamento de mudanças do comportamento, principalmente se as alterações forem sutis, podendo comprometer a eficácia da técnica (atacantes podem desviar-se lentamente do comportamento padrão do sistema, se descoberto, até que seu ataque seja considerado normal).

Em geral, esses métodos apresentam duas importantes questões. A primeira refere-se à construção de bases de assinaturas, para sistemas baseados em abusos, ou de modelos de comportamento normal, para sistemas baseados em anomalias, os quais devem ser construídos a partir dos dados rotulados.

A segunda é a capacidade de reconhecer atributos nos registros de conexões do ambiente de rede monitorado. Atributos são dados que caracterizam as conexões de tráfego em rede como sendo pertencente a uma determinada classe de ataque, para sistemas baseados em anomalias, ou dados que caracterizam uma determinada assinatura, para sistemas baseados em assinaturas.

Atributos de maior relevância são aqueles que contribuem mais significativamente para caracterizar um dado registro de conexão como sendo pertencente a uma determinada classe de ataque, para sistemas baseados em anomalias, ou para caracterizar uma determinada assinatura, para sistemas baseados em abusos (KAYACIK; ZINCIR-HEYWOOD; HEYWOOD, 2005).

Atributos de menor relevância são aqueles que, por sua pouca contribuição na caracterização do tráfego, podem ser desconsiderados em sistemas de detecção de intrusão. Mais detalhes sobre relevância de atributos em registros de conexões serão tratados na seção 5.3.

A figura 3 mostra um resumo da classificação dos IDS, onde as caixas em destaque mostram a classe a que pertence o IDS abordado no presente trabalho, ou seja, um IDS baseado em rede (NIDS), que utiliza técnica de detecção baseado em anomalias, através do uso de Redes Neurais Artificiais, com detecção a posteriori, e reatividade passiva.

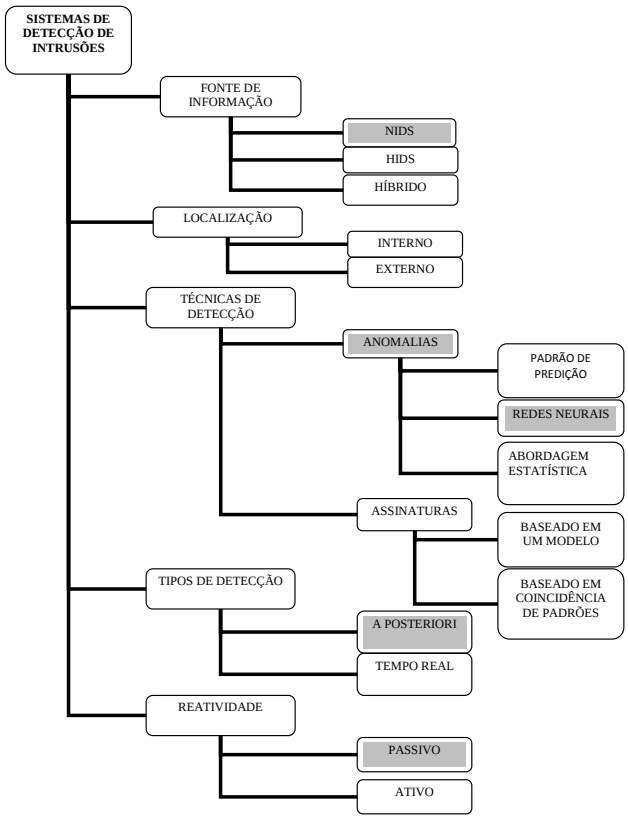

Figura 3 - Classificação dos IDS.
Fonte: elaborada pelo autor

A análise de fluxos de dados é uma técnica escalável, usada em redes de computadores com grande volume de tráfego para diversas finalidades. Na RFC 3917 são descritas possíveis aplicações da análise de fluxos, entre elas, engenharia de tráfego, caracterização de tráfego, monitoramento de qualidade de serviço, contabilidade de uso e detecção de intrusão e anomalias. Um fluxo de dados pode ser caracterizado como um conjunto de pacotes com características em comum. Alguns ataques de difícil detecção em IDS, baseados em assinaturas, podem ser facilmente detectados através da análise de fluxos. Isso se deve ao fato de que os datagramas de rede são correlacionados na análise de fluxos em detrimento da análise individual para cada datagrama realizada em IDS tradicionais.

Existem tecnologias distintas para a análise de fluxos de dados. Uma das mais populares é a tecnologia Netflow da Cisco. Fluxos Netflow se assemelham às conexões TCP. No entanto, os fluxos são unidirecionais, ou seja, uma conexão TCP gera no mínimo dois fluxos. Um fluxo *Netflow* pode ser definido como uma tupla contendo as informações: IP de origem, IP de destino, porta de origem, porta de destino, valor do campo Protocolo do Datagrama IP, byte de *Type of Servic (*ToS) e interface lógica de entrada no roteador.

2.4 ATAQUES E INTRUSÕES

Ataques e intrusões podem ser classificados em quatro categorias distintas: negação de serviço *denial of service (DoS*), remoto para usuário (R2L), usuário para super usuário (U2R) e reconhecimento (*probing*).

Negação de serviço é uma classe de ataques contra a disponibilidade de sistemas computacionais. O objetivo do atacante é causar a indisponibilidade do serviço alvo, seja através do consumo excessivo de recursos computacionais como memória, processador ou rede para atendimento a solicitações falsas, ou seja, através de indisponibilidade total do serviço por exploração de uma falha grave existente no mesmo.

Em fevereiro de 2000, uma seqüência de ataques de negação de serviço causou total indisponibilidade de serviços de empresas com alta visibilidade na Internet como Yahoo, e-Bay, Amazon.com dentre outras. Esse episódio marcou o surgimento, em situações práticas, dos chamados ataques de negação de serviço distribuído *Distributed Denial of Service* (DDOS). Variação dos ataques de negação de serviço envolvendo um número enorme de computadores e sistemas controlados remotamente, e utilizados como robôs para atacar determinado alvo. São exemplos de ataques e intrusões da classe DoS: *pod, neptune, mailbomb, land, udpstorm, teardrop, smurf, processtable, apache2* e *back.* No apêndice C podemos encontrar uma descrição resumida dos ataques de classe DoS existentes da base de registros de conexões (KDD Cup, 1999).

Ataques e intrusões da classe usuário para superusuário (*User to Root*) englobam todos os ataques em que o intruso possui acesso ao sistema como um usuário normal e consegue elevar seu nível de privilégio para o de um usuário especial (como o usuário root em sistemas Unix ou administrador em outras plataformas). Existem diversos tipos de ataques nesta classe sendo o mais comum os ataques que empregam técnicas de estouro de buffer (buffer *overflow*). São exemplos de ataques e intrusões da classe usuário para superusuário: *xterm, Perl, loadmodule, os, rootkit, sqlattack* e *buffer_overflow*. No apêndice F podemos encontrar uma descrição resumida dos ataques de classe usuário para superusuário, existentes da base de registros de conexões (KDD Cup, 1999).

Ataques e intrusões da classe remoto para local (*Remote to Local*) correspondem a situações em que o intruso possui conectividade com a máquina vítima, sem possuir uma conta de usuário, e explorando alguma vulnerabilidade existente, consegue obter acesso local à máquina. São exemplos de ataques e intrusões da classe remoto para usuário: *snmpguess, xlock, sendmail, xsnoop, warezclient, warezmaster, spy, worm, guess_passwd, named, httptunnel, phf, imap, ai_write* e *multihop.* No apêndice E podemos encontrar uma descrição resumida dos ataques de classe remoto para local, existentes da base de registros de conexões (KDD Cup, 1999).

Ataques e intrusões da classe reconhecimento (*Probing*) são normalmente empregados em uma etapa que antecede o ataque ou intrusão. Nesta categoria estão ferramentas e mecanismos que permitem ao invasor encontrar e conhecer melhor novas vítimas. Através das técnicas desta classe o invasor pode descobrir, por exemplo, quais sistemas e serviços estão ativos em determinado computador, dentre outras informações. Após este levantamento, o invasor precisa selecionar quais técnicas de ataque e intrusão terão sucesso para aquele perfil levantado do alvo. São exemplos de ataques e intrusões da classe reconhecimento: *ipsweep, nmap, aint, satan, mscan* e *portsweep.* No apêndice D podemos encontrar uma descrição resumida dos ataques de classe reconhecimento, existentes da base de registros de conexões (KDD Cup, 1999).

Do ponto de vista dos sistemas de detecção de intrusão, é importante detectar atividade relacionada a esta categoria, pois permite uma preparação e resposta adequada à próxima etapa que provavelmente será iniciada pelo intruso.

Quanto aos privilégios que um invasor pode obter em um sistema computacional através de algum método de transição de privilégios, podemos destacar:

a) acesso remoto via rede ®, que significa obter algum tipo de acesso através de rede de sistemas interconectados ao computador alvo;

b) acesso local à rede (L), que permite ao invasor ler e escrever no segmento de rede em que o sistema alvo está conectado;

c) acesso com usuário normal (U), que envolve a capacidade de executar comandos de usuário na máquina alvo; e

d) acesso como super-usuário (S), que significa ter a capacidade de executar comandos privilegiados (de administração) na vítima.

Alguns autores também abordam o acesso físico como uma forma de privilégio obtido por um invasor em um sistema operacional, que corresponde à capacidade

de acessar fisicamente o hardware e componentes do computador em referência. Essa forma de acesso não será abordada no presente trabalho.

2.5 TRÁFEGO DE REDES E ANOMALIAS

As características do tráfego em redes IP são agrupadas em quatro categorias:

a) **Características básicas**: características básicas podem ser derivadas de cabeçalhos de pacotes sem inspeção de sua carga (*payload)*;

b) **Características de conteúdo**: o domínio do conhecimento é usado para interpretar o *payload* dos pacotes TCP originais, de modo a se obter as características desejadas;

c) **Característica de tráfego baseada no tempo**: essas características são destinadas a capturar propriedades que somente se consolidam em uma janela temporal, sendo portanto características que não podem ser extraídas em um único instante de tempo. Um exemplo desse tipo de característica seria o número de conexões para um mesmo *host*, dentro de um intervalo de 2 segundos; e

d) **Características de tráfego baseadas em *Host***: Utiliza uma janela histórica estimada sobre o número de conexões, ao invés de utilizar o tempo como parâmetro, para avaliar ataques.

Anomalias na rede podem ser detectadas a partir de informações do *host*, como acesso a arquivos, número de chamadas de sistema (*system calls*), dados de processos do kernel do sistema, compreendendo o número de processos filhos cancelados por processos-pai e quantidade do tempo gasto pelo processo nos modos usuário e sistema, podem ser detectadas a partir de dados do sistema, tais como uso de memória e carga de CPU, e podem ser detectadas a partir de dados coletados do tráfego de rede, envolvendo informações estatísticas relacionadas às camadas TCP, IP, ICMP, UDP da pilha de protocolos de rede, como, por exemplo,

taxas de alteração do número de pacotes recebidos, número de pacotes recebidos com erro e número de pacotes descartados.

As anomalias podem ser classificadas em dois grandes grupos. O primeiro grupo envolve as *falhas na rede* assim como problemas de desempenho. O outro grupo de anomalias diz respeito à segurança da rede perante os ataques. Um dos principais exemplos deste último grupo, é o ataque do tipo *Denial of Service* (DoS). Essa intrusão causa a indisponibilidade de serviços na rede através do ataque ao provedor do serviço. O foco desse trabalho será concentrado no segundo grupo de anomalias descrito.

Num plano ideal teríamos um IDS com baixo número de falsos positivos e negativos apresentados pelos sistemas baseados em detecção por assinaturas, combinado com a dinâmica dos sistemas de detecção por anomalias. Esse IDS teria atualizações constantes do padrão de normalidade (*baseline*), cautelosas, precisas e que não corresse o risco de considerar como normais diversas atividades maliciosas. Estaria sempre atualizado contra ataques não conhecidos.

IDS por detecção de anomalias definem um *baseline* que serve de comparação constante. No entanto, estes últimos também têm problemas, como é o caso de definições de *baselines* pouco precisos.

3 REDES NEURAIS

A tentativa inicial de reproduzir o alto desempenho do cérebro humano em tarefas cognitivas extremamente complexas motivou o desenvolvimento dos modelos de Redes Neurais Artificiais (RNA). Tais modelos representam um tipo especial de processamento da informação, que consiste de muitas células primitivas que trabalham em paralelo e estão conectadas através de ligações diretas, cuja principal função é distribuir padrões de ativação, de maneira similar ao mecanismo básico do cérebro humano (HAYKIN, 2001).

Uma rede neural artificial é uma estrutura composta de unidades processadoras simples, distribuídas e paralelas, que tem o propósito de armazenar conhecimento por meio de mecanismos empíricos e torná-lo disponível para o uso. Ela se espelha no cérebro humano em dois aspectos:

a) o conhecimento é adquirido pela rede a partir do meio ambiente através de mecanismos de aprendizagem; e

b) a “força” das conexões entre as unidades processadoras são minimizadas ou maximizadas, de forma a armazenar melhor o conhecimento adquirido.

O poder computacional de uma rede neural advém de sua estrutura maciçamente paralela e distribuída e de sua capacidade de aprendizagem e generalização. O conceito de generalização deve ser visto como a habilidade da rede em fornecer valores adequados para entradas não disponíveis no processo de aprendizagem.

O primeiro trabalho realizado com IDS com auxilio de redes neurais foi divulgado na 13ª Conferência Nacional de Segurança de Computador, em Washington, Capital, em outubro de 1990 – Security in Computing.

As Redes Neurais Artificiais são o mais difundido e popular conjunto de métodos subsimbólicos, sendo em geral caixas-pretas por excelência, sendo esta uma de suas desvantagens (WANGENHEIM, 2007).

O termo neurônio refere-se a cada unidade processadora da Rede Neural e são implementados através de uma função matemática. O procedimento utilizado para realizar a aprendizagem da Rede Neural é feito por meio de um algoritmo de aprendizagem, cuja função é modificar a magnitude da “força” de ligação entre os neurônios, ou seja, modificar os pesos sinápticos das conexões de uma forma ordenada para alcançar um objetivo desejado.

Uma forma comum de aprendizado é o paradigma supervisionado onde é fornecida à Rede Neural uma entrada com a respectiva saída esperada, e então, os pesos sinápticos são ajustados de forma a produzir a saída ou resposta esperada. Esse processo deve se repetir até que a rede forneça as saídas esperadas, ou seja, haja a convergência da rede. Uma aplicação dessa propriedade é a classificação de padrões, da qual o objetivo é encontrar arbitrariamente a fronteira de disjunção entre as várias categorias ou classes preestabelecidas de objetos ou eventos.

No contexto de classificação de padrões, uma Rede Neural pode ser projetada não só para responder sobre a disjunção de objetos, mas também sobre a confiança ou crença na decisão tomada (ZHANG et al., 2001). Essa característica deverá ser aplicada quando o problema em questão tratar de padrões ambíguos, dando assim maior confiabilidade ao processo de classificação.

Caso um neurônio venha a falhar, a própria estrutura descentralizada das redes neurais favorece a continuidade do bom funcionamento do sistema. Assim, o dano causado à rede deve ser relativamente amplo para que a sua resposta final da rede seja seriamente comprometida. Em princípio, a degradação de uma Rede Neural ocorre suavemente e situações anormais apenas são causadas por sérios danos à mesma.

A figura 4 mostra um neurônio com a identificação de seus elementos básicos.

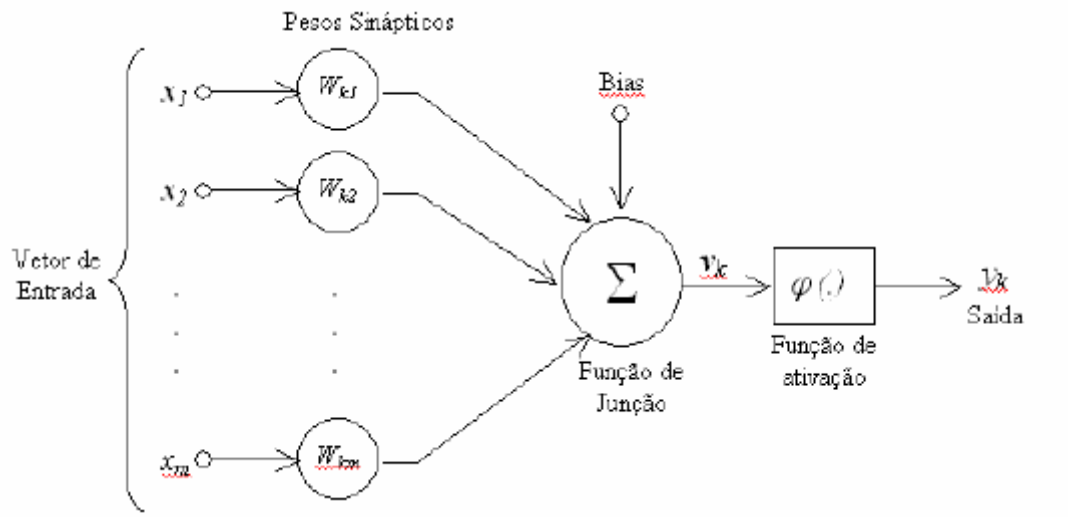

Figura 4 – Modelo de um neurônio artificial.
Fonte: elaborada pelo autor

Um sinal x_j na entrada j conectado a um neurônio k é multiplicado pelo peso sináptico w_{kj}, onde o índice k refere-se ao neurônio em questão e o índice j refere-se ao terminal de entrada.

A função de junção soma os sinais de entrada já ponderados pelos seus respectivos pesos sinápticos. Ela também é conhecida como combinador linear ou simplesmente somador. A função de ativação se faz necessária para restringir a amplitude de saída de um neurônio. Por convenção, as saídas dos neurônios devem estar restritas ao intervalo fechado [0,1] ou [-1,1]. Assim, a função de ativação ou função restritiva tem como objetivo adequar a saída do neurônio a um desses intervalos. Um fator que pode interferir na função de ativação é o bias, o qual pode ser identificado na figura 4, sendo b_k sua representação matemática. Este fator tem o efeito de aumentar ou diminuir a entrada da função de ativação, dependendo se ele é positivo ou negativo, respectivamente. Dessa forma, o bias interfere diretamente na sensibilidade do neurônio em análise às entradas.

Em termos matemáticos, um neurônio k é descrito pelas seguintes equações:

$u_k = \sum_{j=1}^{m} w_{kj}x_j$ e $y_k = \varphi(u_k + b_k) = \varphi(v_k)$,

tal que x_1, x_2, x_3,...x_m, são sinais de entrada; w_{k1}, w_{k2}, w_{k3},...w_{km}, são os pesos sinápticos do neurônio k; u_k é a saída da função de junção; b_k é o bias; $\varphi(\ .\ )$ é a função de ativação, e y_k é o sinal de saída do neurônio.

A equação que define o potencial de ativação v_k, é dada por:

$v_k = u_k + b_k$

3.1 ARQUITETURA DAS REDES NEURAIS

A forma como os neurônios estão organizados e interconectados define a arquitetura de uma rede neural e está intimamente ligada com o algoritmo de aprendizagem usado para treinar a rede.

As Redes Neurais Artificiais são organizadas em camadas que definem sua estrutura topológica ou sua arquitetura (maneira como os elementos de processamento são organizados). Assim sendo, existem redes neurais de camada simples (*perceptron*) constituídas por um grupo de neurônios arranjados em apenas uma camada, e as redes multicamadas (*feedforward*), formadas por várias camadas intermediárias ou pela combinação de várias redes de camadas simples. Nessa estrutura, a camada de entrada de nós de fonte é aquela na qual os padrões são apresentados à rede; as camadas intermediárias são responsáveis por grande parte do processamento, podendo ser consideradas como extratoras de características; e a camada de saída é aquela, onde o resultado final é concluído e apresentado.

Na forma mais simples de uma rede em camadas, temos como entradas os nós fontes que se comunica diretamente com a camada de saída em sentido único,

através de sinapses, que são unidades estruturais e funcionais elementares que medeiam às interações entre neurônios. Dessa forma, esta rede é sempre alimentada adiante, sendo por isso denominada de rede *feedforward* ou acíclica.

Neste trabalho foram usadas Redes Neurais *feedforward* multicamadas, as quais se distinguem das redes com camada única por apresentarem uma ou mais camadas intermediárias entre a entrada e a saída da rede neural. Estas camadas são denominadas camadas ocultas, cujos nós computacionais são chamados correspondentemente de neurônios ocultos. A razão para a presença de neurônios ocultos é proporcionar às redes neurais a capacidade de extrair estatísticas de ordem mais elevada sobre os dados de entrada, ou seja, a rede adquire uma perspectiva global apesar de sua conectividade local, devido a um conjunto extra de conexões sinápticas e da dimensão extra de interações neurais. A habilidade de os neurônios ocultos extraírem estatísticas de ordem elevada é particularmente valiosa quando o tamanho da camada de entrada (nós de fonte) for muito grande (HAYKIN, 2001). Pesos sinápticos e bias são parâmetros do neurônio artificial, os quais servem de fatores multiplicadores dos sinais de entrada e sinal externo de um neurônio, respectivamente.

Uma Rede Neural Recorrente difere das *feedforwards* pelo fato de apresentar pelo menos um laço de retro-alimentação. Os elementos de atraso unitário, representados por z^{-1}, são responsáveis por fornecer saídas já computadas em iterações passadas à Rede Neural Recorrente o que explica o termo retro-alimentação. O atraso no fornecimento de entradas à Rede Neural introduz memória na rede, proporcionando aos neurônios valores de entrada atuais e valores temporalmente anteriores a eles. As Redes Neurais Recorrentes geralmente apresentam um comportamento dinâmico não-linear, o que as torna mecanismos úteis na predição de valores inerentemente temporais (séries temporais).

Na prática, redes recorrentes são utilizadas principalmente para memórias associativas e para aproximar o mapeamento entrada-saída.

3.2 FASE DE APRENDIZADO DE UMA RNA

A propriedade primordial das redes neurais é aprender conforme o ambiente onde estão inseridas e assim melhorar seu desempenho. Para que uma rede neural possa aprender, se faz necessário apresentar um conjunto de exemplos à mesma de forma seqüencial e interativa. Esse processo é chamado de treinamento de redes neurais. Em cada interação, os pesos sinápticos e o bias são ajustados, através de um algoritmo de aprendizagem. Durante o processo de propagação, os pesos sinápticos da rede são todos fixos. Durante o passo para trás, por outro lado, os pesos sinápticos são todos ajustados de acordo com uma regra de correção de erro. Especificamente, a resposta real da rede é subtraída de uma resposta desejada (alvo) para produzir um sinal de erro, que é propagado para trás através da rede, contra a direção das conexões sinápticas. Esse processo deve se repetir até o objetivo da rede ser alcançado.

Teoricamente, uma rede neural torna-se mais instruída a cada passo do processo de aprendizagem. Segundo Haykin (2001), aprendizagem no contexto de redes neurais pode ser definida como um processo pelo qual os parâmetros livres (i.e., pesos sinápticos e bias) de uma rede neural são adaptados através de um processo de estimulação pelo ambiente no qual a rede está inserida. O tipo de aprendizagem é determinado pela forma que a modificação dos parâmetros ocorre. Então, segundo Haykin (2001), podem ser identificados três eventos principais no processo de aprendizagem que ocorrem de forma seqüencial:

a) a rede neural recebe os padrões de aprendizagem;
b) segundo os padrões de aprendizagem, os pesos sinápticos da rede neural são modificados; e
c) a rede neural responde de uma maneira nova ao problema para o qual se destina.

Quanto aos paradigmas de aprendizagem, há dois tipos básicos: o supervisionado, também conhecido como processo de aprendizado com auxílio de um professor, e o paradigma não-supervisionado ou auto-supervisionado.

No paradigma supervisionado, um “professor” indica à rede neural qual o resultado esperado para uma entrada específica da mesma. O valor esperado pelo professor é comparado com o valor fornecido pela rede neural, conseguindo-se, dessa forma, um erro estimado. Este erro deve ser usado pela rede como parâmetro de correção dos pesos sinápticos. Nesse paradigma, o professor é representado por um conjunto de vetores de entrada com seus respectivos valores de saída desejados. Estas informações compõem o padrão de treinamento ou exemplos usados no processo de aprendizado. Esse processo interativo deve prosseguir até que a taxa de acerto da rede em treinamento atinja um patamar satisfatório. Esta forma de aprendizado é bem conhecida e tem demonstrado excelentes resultados em aplicações reais.

Backpropagation é um método simples de treinamento no qual os pesos são atualizados de padrão em padrão, até formar uma época, isto é, uma apresentação completa do conjunto de treinamento inteiro que está sendo processado. Os ajustes dos pesos são realizados de acordo com os respectivos erros calculados para cada padrão apresentado à rede. A média aritmética destas alterações individuais de peso sobre o conjunto de treinamento é, portanto, uma estimativa da alteração real que resultaria da modificação dos pesos baseada na minimização da função de custo sobre o conjunto de treinamento inteiro (HAYKIN, 2001).

O algoritmo *backpropagation* permite um ajuste de pesos em cada uma das camadas da rede e é projetado para minimizar a soma do erro médio quadrático entre a saída calculada por uma arquitetura multicamadas e a saída desejada.

O processo de treinamento pelo algoritmo *backpropagation* começa com a definição de um conjunto arbitrário de pesos para as conexões da rede e envolve duas fases distintas. Na primeira, um vetor de treinamento com a respectiva saída desejada é

apresentado à rede e propagada através de suas camadas, computando uma saída para cada elemento de processamento. As saídas dos nós da última camada são, então, comparadas com as saídas desejadas e, a partir disso, são calculados os termos de erro. A segunda fase envolve um retrocesso, ou seja, uma passagem de volta através da rede a partir da última camada, durante a qual, o erro é repassado para cada elemento de processamento e os pesos correspondentes são alterados. Em um treinamento bem sucedido, o erro diminui com o aumento do número de iterações e o procedimento converge para um conjunto estável de pesos.

Seja E_p, a função do erro médio quadrático para o padrão p, d_{pj} o valor da saída desejado (para o padrão p e o nó j) e o_{pj}, o valor da saída obtido. Então, a função do erro médio quadrático (E_p) é dada por:

$$E_p = {}^{1}\!/_{2} \sum (d_{pj} - o_{pj})^2$$

Após o aprendizado da rede, ela classifica os padrões da área de interesse usando o conjunto dos pesos ajustados durante a fase de treinamento, reconhecendo um dado padrão como pertencente a uma determinada classe, quando a saída para essa classe for “alta”, e “baixa” para as demais classes.

Em linhas gerais, o critério de decisão adotado estabelece que, para um dado padrão de entrada, o elemento de processamento da camada de saída (associado a uma classe específica) que produzir o maior resultado numérico definirá a classe à qual o padrão será atribuído. Para algumas implementações de redes neurais, além dessa saída ser maior para uma das classes, ela deve ter um valor superior ao limite de tolerância, para que o padrão possa ser atribuído àquela classe.

Em termos práticos, dada uma rede *feedforward* de M camadas, o Algoritmo *Backpropagation* funciona da seguinte forma:

Dado:

ξ^{μ} = Vetor de entrada do padrão de treinamento μ apresentado à camada de entrada;

ξ^{μ}_{k} = Valor de entrada deste padrão para o neurônio k da camada de entrada;

ζ^{μ} = Vetor de saída esperado do padrão de treinamento μ apresentado à camada de entrada;

ζ^{μ}_{i} = Valor de saída deste padrão para o neurônio *i* da camada de saída;

W_{ij} = Peso da conexão dirigida do neurônio *j* para o neurônio *i*. Lê-se "peso da conexão que *i* recebe de *j*";

h^{μ}_{j} = Entrada total do neurônio *j* para o padrão μ;

V^{μ}_{j} = Sinal de saída (ativação) do neurônio interno (hidden) *j* para o padrão μ;

O^{μ}_{i} = Sinal de saída (ativação) do neurônio da camada de saída *i* para o padrão μ;

g = Função de ativação. Computa a ativação de um neurônio dada uma entrada h.

Passos do Algoritmo *Backpropagation*:

a) inicialize os pesos w^{m} das camadas m = 0,... M com valores aleatórios e pequenos;

b) escolha um padrão μ do conjunto de treinamento, com entrada da rede ξ^{μ} e saída desejada ζ^{μ}, e apresente o padrão à rede para todos os neurônios de entrada;

c) propague a ativação através das m = 1,... M camadas restantes da rede:

$$V_i^m = g(h_i^m) = g\left(\sum_j w_{ij}^m V_j^{m-1}\right)$$, até que as saídas da rede tenham sido calculadas,

d) calcule os δ para a camada de saída:

$$\delta_i^{\mu} = g'(h_i^{\mu}) \cdot (\zeta_i^{\mu} - O_i^{\mu})$$, para todo i,

e) itere pelas camadas anteriores, de trás para frente, calculando os δ através da retropropagação do erro:

$$\delta_p^{m,\mu} = g'(h_p^{m,\mu}) \cdot \sum_r w_{rp}^{m+1} \cdot \delta_r^{m+1,\mu}$$, para todo neurônio *p* da camada m – 1,

f) determine a variação dos pesos para todas as camadas:

$\Delta w_{pq}^{m} = \eta \sum_{\mu} \delta_{p}^{m} V_{q}^{m-1}$, para todas as conexões entre neurônios,

g) determine os novos pesos das conexões:

$$w_{pq}^{m} = w_{pq}^{m} + \Delta w_{pq}^{m} ,$$

h) retorne ao passo 2 e tome o próximo padrão.

O quadro 1 mostra os principais algoritmos de aprendizagem de redes neurais e suas aplicações mais comuns (HAYKIN, 2001).

Algoritmo de Aprendizagem	**Paradigma de Treinamento**	**Arquitetura**	**Aplicação Principal**
Adaptative Resonance Theory	Não-supervisionado	Recorrente	Agrupamento
Kohonen Feature Maps	Não-supervisionado	*Feedforward*	Agrupamento
ARTMAP	Supervisionado	Recorrente	Classificação
Recurrent Backpropagation	Supervisionado	Recorrente	Séries Temporais
Radial Basis Function Networks	Supervisionado	*Feedforward*	Classificação, Séries Temporais
Backpropagation	Supervisionado	*Feedforward*	Classificação

Quadro 1 – Principais algoritmos de aprendizagem de Redes Neurais.
Fonte: elaborado pelo autor

Classificadores são programas cujo trabalho básico é classificar um conjunto de dados de entrada em uma classe. Os dados de entrada são fornecidos em um vetor de características. Os classificadores podem ser classificadores com aprendizado supervisionado ou com aprendizado não-supervisionado (STEINER et al., 2007).

Há dois modelos de redes neurais utilizados na prática como classificadores passíveis de serem gerados através de aprendizado supervisionado. Ambos os modelos baseiam-se nos *Perceptrons feed-forward*, variando o número de camadas e a função de ativação e, por conseguinte, a regra de aprendizado: as Redes *Backpropagation* (Redes BP) e as Redes de Função de Base Radial (Redes RBF ou *Radial Base Function*).

As redes de Base Radial, conhecidas também por redes-RBF podem ser usadas, em teoria, para representar os mesmos tipos de problemas que uma rede *Backpropagation* equivalente. Segundo Haykin (2001), sempre existe uma rede RBF capaz de imitar precisamente uma rede BP específica, ou vice versa.

Por outro lado, a compreensão do algoritmo de aprendizado da rede RBF envolve uma matemática bem mais complexa. Além disso, para um mapeamento de entrada-saída não-linear, a rede BP requer um número menor de parâmetros que a rede RBF, para o mesmo grau de precisão (HAYKIN, 2001). Assim, foram utilizadas as Redes BP no presente trabalho.

Antes de tentar resolver um problema através do paradigma de redes neurais, é aconselhável que se faça uma análise para verificar se o domínio do problema pode ser tratado através de RNAs. As redes neurais são aplicáveis a problemas como: reconhecimento e classificação de padrões, predição de séries temporais e aproximação de funções complexas. Deve-se ter cuidado na seleção das variáveis relevantes ao problema em questão. Esta seleção envolve, além da identificação das variáveis intimamente relacionadas ao problema, a exclusão das variáveis não-confiáveis ao processo, ou cujo uso seja impraticável por razões técnicas ou econômicas.

4 SOLUÇÃO PROPOSTA PARA DETECÇÃO DE INTRUSÃO COM REDES NEURAIS

Para que uma Rede Neural possa cumprir seu propósito, fatores como disponibilidade e características dos dados, bem como topologia e algoritmo de treinamento devem ser criteriosamente analisados (HEADY et al., 1990). O problema em questão trata da detecção de intrusão por anomalias, utilizando Redes Neurais Artificiais. A capacidade de generalização das Redes Neurais permite, ao sistema de detecção, identificar padrões de ataques e intrusões que não foram apresentados anteriormente à rede, durante a etapa de aprendizagem. A generalização se refere ao fato de a rede neural produzir saídas adequadas para entradas que não estavam presentes durante o treinamento – aprendizagem (HAYKIN, 2001). Diz-se que uma rede generaliza bem quando o mapeamento de entrada-saída computado pela rede for correto (ou aproximadamente correto) para dados de teste não-utilizados para a criação ou treinamento da rede HAYKIN, 2001).

4.1 TRABALHOS RELACIONADOS

Soluções para sistemas detectores de intrusão com base na detecção de anomalias de tráfego em redes de computadores, e classificação dos ataques que ocasionaram tais anomalias, são abordadas em diversos trabalhos na literatura especializada atual. O desafio é atingir um sistema detector de intrusão altamente confiável, de fácil implementação e que obtenha alta taxa de acertos na detecção e classificação dos ataques presentes em redes, e que sejam altamente adaptável aos mais diversos tipos de redes reais de computadores.

O trabalho de Mafra e outros (2008) apresenta um modelo de sistema de detecção de intrusão que classifica o tráfego de rede por análise comportamental como normal ou anômalo. Para detecção de anomalias são utilizadas duas técnicas de

inteligência artificiais chamadas *Support Vector Machine* (SVM) e Redes Neurais de Kohonen (KNN).

Ainda conforme o trabalho acima citado, um problema presente em redes neurais é o tempo de treinamento das mesmas, que é normalmente efetuado *off-line*. Contudo, uma vez treinadas, os tempos de análise são consideravelmente eficientes.

Mafra e outros (2008) desenvolveram um sistema multicamadas chamado POLVO-IIDS, utilizando as Redes Neurais de Kohonen para classificar dados de forma genérica (comportamentos normal e anômalo), e Redes Neurais do tipo *Support Vector Machine* (SVM), sendo que para cada classe de ataque (DoS, Worm, Scan ou Normal) existe uma SVM especializada para detectar o tipo correspondente de ataque. Portanto, terá duas opções como saída: tráfego normal ou atividade maliciosa. O uso de Redes Reurais com aplicações mais especializadas, segundo Mafra e outros (2008), teve como objetivo obter-se maior precisão, pois cada SVM foi treinada para separar os dados em apenas duas classes.

Assim, no trabalho supracitado, o classificador faz uma pré-seleção do tráfego de entrada, através da análise de características contidas nos pacotes em um determinado período de tempo. Para minimizar a taxa de falsos positivos, foi utilizada uma outra Rede Neural na saída do classificador, rede essa com função mais especializada, pois é responsável por analisar determinado tipo de ataque e identificar com mais precisão o que é anomalia e o que é tráfego normal. Dessa forma, pretendeu-se reduzir a taxa de falsos positivos dos IDSs convencionais e melhorar a taxa de acerto. Para a realização de testes com o modelo apresentado, foi usado o tráfego KDD Cup (1999). Data disponível na Internet.

No trabalho de Haijun e outros (2007), foi elaborado um estudo comparativo entre o uso de *support vector machine* (SVM), redes neurais e outras técnicas de mineração de dados para a detecção de intrusão. Nos testes realizados, a taxa de detecção de

intrusão obtida com SVM e redes neurais não apresentaram diferenças significativas entre si, mas essas duas técnicas mostraram-se superiores às demais utilizadas.

Em Cândido Júnior e outros (2005) apresentaram as Redes Neurais Artificiais aplicadas ao reconhecimento de padrões obtidos em fluxos de dados para detecção de ataques e anomalias. A ferramenta Neuro-sig foi desenvolvida em ambiente UNIX com o intuito de gerar padrões para a rede neural, a partir de uma base de dados de registros de conexões. Assim, a ferramenta Neuro-sig foi desenvolvida com o propósito de auxiliar a geração de padrões para treinamento de redes neurais. A rede neural foi treinada a partir das informações fornecidas no arquivo de padrões, estando pronta para a classificação de ataques.

Os fluxos de dados foram coletados em uma instituição acadêmica. No treinamento de cada modelo foram utilizados fluxos obtidos em um período de duas semanas de tráfego. Adicionalmente, tráfego de uma semana foi utilizado para a validação da Rede Neural (CÂNDIDO JUNIOR et al., 2005).

Em Cândido Junior e outros (2005), optou-se por simular ataques de negativa de serviço. Em ataques dessa natureza é observado um número excessivamente grande de fluxos de dados por segundo. O evento “tráfego desconhecido” foi criado para classificação de um evento anômalo de difícil classificação. Esse evento é semelhante ao tráfego lícito, ou seja, não demanda nenhum tipo de ação quando detectado. Estes testes mostraram que Redes Neurais podem ser utilizadas com sucesso na detecção de ataques e anomalias em redes de computadores com um baixo índice de erros.

Dalmazo e outros (2008) classificaram os sistemas de detecção de intrusão em três componentes fundamentais: fonte de informação, análise e resposta. A fonte de informação foi representada por um coletor associado a um *host*, rede ou segmento de rede. A análise, como a parte do SDI que verifica os eventos derivados da fonte de informações, determinando quando estes eventos indicam que uma intrusão está

ocorrendo ou já ocorreu. E a resposta foi representada como o conjunto de ações que o IDS realiza quando detecta uma intrusão, por exemplo, a intervenção automatizada ou a geração de alertas e relatórios para a interpretação e intervenção humana.

O principal desafio do desenvolvimento de um IDS é escolher um método eficiente que identifique uma intrusão de maneira correta, sem gerar um número excessivo de falsas detecções.

Dalmazo e outros (2008) propuseram a utilização da teoria de séries temporais na fase de análise de um IDS, objetivando de identificar com maior confiança uma intrusão e diminuir o número de falsos positivos. Uma série temporal é um modelo matemático para representar amostragens periódicas que apresentam dependência entre as amostras.

Como resultado preliminar, os testes realizados com o Detector de Intrusões Baseado em Séries Temporais (DIBSeT), no trabalho supra citado, demonstraram que a utilização de séries temporais para a detecção de ataques (negação de serviço com ataque SYN e SMURF) apresentaram resultados satisfatórios quanto a identificação de um ataque, além de consumir pouco tempo de processamento.

Do ponto de vista algorítmico, o uso de séries temporais exige uma fase de preenchimento da série apenas no início da computação, seguida de um ajuste do modelo de previsão, o qual pode ser revisto sempre que o erro de predição começar a aumentar. O ajuste do modelo significa determinar quais os parâmetros do modelo ARIMA (número de termos auto-regressivos e de médias móveis, e número de integrações necessárias para tornar a série estacionária). Realizado os ajustes, o preditor possibilita realizar as previsões, ou seja, detectar ataques presentes no tráfego de rede (DALMAZO et al., 2008).

O trabalho de Ferreira e outros (2008) apresentou uma proposta de uso das Transformadas de Wavelets para detecção de anomalias, e classificação dos ataques através de Redes Neurais Artificiais. A transformada de Wavelet é uma técnica matemática com capacidade de realizar a decomposição de funções.

O emprego de Wavelets para a detecção de anomalias requer o uso de uma função que represente o comportamento da rede de forma mais realística possível. Além disso, as métricas utilizadas, tais como banda utilizada, número de conexões ativas, número de pacotes transmitidos/recebidos etc., para identificação do comportamento da rede, devem ser escolhidas mediante dois fatores importantes: facilidade de uso e independência dos sistemas operacionais em cada nó da rede (FERREIRA et al., 2008).

O algoritmo simulado detectou mudanças abruptas no comportamento da rede. Em seguida, uma rede neural, previamente treinada, foi utilizada para classificar os ataques detectados na etapa anterior. Assim, Ferreira e outros (2008) tiveram como proposta demonstrar que é possível utilizar transformadas de wavelet para detectar anomalias na rede.

Dessa forma, a utilização conjunta das transformadas de wavelet com as Redes Neurais Artificiais pode realizar a classificação dos ataques com maior precisão. Ferreira e outros (2008) consideraram que os resultados iniciais foram promissores.

4.2 DETECÇÃO DE INTRUSÃO POR ANOMALIAS COM REDES NEURAIS

A forma de detecção baseada em anomalias permite a identificação de uma tentativa de intrusão a partir do desvio do comportamento considerado normal. Com isso, torna-se possível a detecção de ataques cuja identificação formal de sua assinatura ainda se encontra sob investigação, oferecendo aos recursos da rede

mecanismos que permitam isolar por completo uma ameaça muito rapidamente, oferecendo maior robustez funcional se comparado à forma de detecção baseado em assinaturas.

Contudo, por dispensar o arquivo de assinaturas, a estratégia de detecção baseada em anomalias precisa estabelecer um padrão considerado como normal para sua utilização referencial. Tal comportamento é obtido a partir de um período de treinamento onde é observado o funcionamento sistêmico da rede e estabelecido um perfil que será utilizado como referência que fundamentalmente resume todos os padrões do período.

Atualmente, já foram desenvolvidas técnicas que abrangem o reconhecimento de padrões de comportamento da rede através de processamento on-line do sinal ou processamento de conjuntos de dados que fazem parte do histórico da rede. Esses métodos são capazes de detectar anomalias não conhecidas, mas trouxeram à tona o problema dos falsos positivos que existem devido à característica não-estacionária do tráfego de rede. Além do objetivo principal da rápida detecção e recuperação da rede, é necessário que os sistemas gerem o mínimo possível de falsos positivos.

Segundo Haykin (2001), o reconhecimento de padrões é formalmente definido como o processo pelo qual um padrão/sinal recebido é atribuído a uma classe dentre um número predeterminado de classes (categorias). Uma rede neural realiza o reconhecimento de padrões passando inicialmente por uma seção de treinamento, durante a qual se apresenta repetidamente à rede um conjunto de padrões de entrada junto com a categoria a qual cada padrão particular pertence. Mais tarde, apresenta-se à rede um novo padrão que não foi visto antes, mas que pertence à mesma população de padrões utilizada para treinar a rede. A rede é capaz de identificar a classe daquele padrão particular por causa da informação que ela extraiu dos dados de treinamento. O reconhecimento de padrões realizado por uma rede neural à de natureza estatística, com os padrões sendo representados por pontos em um espaço de decisão multidimensional. O espaço de decisão é dividido em regiões, cada uma das quais associada a uma classe. As fronteiras de decisão

são determinadas pelo processo de treinamento. A construção dessas fronteiras é tornada estatística pela variabilidade inerente que existe dentro das classes e entre as classes (HAYKIN, 2001).

O principal objetivo de um sistema de detecção de intrusos é ser capaz de alcançar alta taxa de acertos (calculada como sendo a porcentagem dos ataques detectados) e baixa taxa de alarmes falsos (calculada como a porcentagem de acessos legítimos que são classificados como invasões).

Um desempenho ruim de um modelo baseado em detecção de anomalias sugere que as fases de treinamento e testes não foram suficientes ou ainda que as características utilizadas na detecção ou os algoritmos de classificação devem ser refinados (LIMA, 2005). Diversos testes podem ser necessários até que um bom modelo de detecção de anomalias seja produzido. No presente trabalho, foram realizados diversos testes comparativos para se chegar aos parâmetros utilizados na rede neural, e nas etapas de treinamento e testes da rede, conforme descrito no capítulo 5.

Utilizando a capacidade de generalização das redes neurais, espera-se que o presente trabalho tenha como resultado a detecção de novos ataques, mantendo uma alta taxa de acertos.

4.3 DESCRIÇÃO DAS ETAPAS DE IMPLEMENTAÇÃO DO PROJETO

A seguir, serão apresentadas as principais etapas para implementação do sistema de detecção de intrusão baseado em anomalias de tráfego.

4.3.1 Apresentação do problema

Para a classificação dos dados pela RNA, será utilizado o simulador JNNS, conforme descrito na seção 4.4, e será necessário definir a priori, os dados de entrada e as classes de saída a serem mapeadas, o algoritmo de aprendizagem, o número de elementos em cada camada da Rede Neural, função de ativação de cada neurônio, taxa de aprendizagem, entre outras. A definição de cada um desses parâmetros será abordada mais adiante neste trabalho.

Segundo Espinhosa e Galo (2004), o número de elementos de processamento (nós) da primeira camada corresponde à dimensionalidade do vetor de atributos dos dados de entrada. Assim, no presente trabalho, camada de saída apresentará cinco elementos, correspondente às cinco classes a serem separadas. O problema maior está na definição do número de camadas escondidas e do número de nós que as compõem. Na prática, este problema tem sido geralmente resolvido por tentativa e erro e/ou pela experiência prévia do pesquisador no domínio de uma dada situação. A arquitetura da rede neural a ser utilizada, será tratada com mais detalhes no decorrer desta seção.

Os dados de entrada da rede neural, aqui utilizados para treinamento e testes, correspondem a subconjuntos da base de dados disponibilizada na competição internacional de mineração de dados *Knowledge Discovery and Data Mining Competition* - KDD Cup (1999), e será detalhado mais adiante nesta seção. Como os dados de entrada não estão no formato adequado a serem apresentados à RNA, foi necessário alterar a formatação de todos os registros de conexões da base de dados de entrada.

Os resultados obtidos na simulação da rede neural serão analisados na forma de percentual de detecção por cada classe, e comparado com os resultados obtidos na competição KDD Cup (1999), possibilitando, assim, avaliar a eficácia do método.

Melhorar a eficácia da Rede Neural na detecção de intrusões, utilizando heurísticas disponíveis na literatura sobre o tema, assim como selecionando os campos de maior relevância na base de dados dos registros de conexões, de modo a reduzir o número de neurônios de entrada da RNA, são questões abordadas neste trabalho.

4.3.2 Seleção e representação de dados

Conforme descrito na seção 2.3, o modelo proposto neste trabalho possui arquitetura baseada em rede, método de detecção que incorpora funcionalidades de anomalia e geração de respostas passivas aos eventos detectados. Os resultados gerados pelo processo de análise classificam os eventos analisados em padrões considerados intrusivos ou normais.

A escolha correta do conjunto de dados que mede o comportamento da rede é essencial para a eficiente detecção de anomalias. Os tipos de anomalias que podem ser reconhecidos dependem da natureza dos dados utilizados. A reunião de fontes de dados diferentes na correlação de alarmes também pode ajudar na diminuição da quantidade de alarmes falsos.

Há necessidade do tratamento de dados antes que estes possam fazer parte da rede neural. Alguns parâmetros identificados na definição do problema podem ser combinados em um só ou eliminados em caso de redundância. Visto que qualquer valor não pode ser apresentado à rede neural, técnicas de codificação e normalização dos dados deverão ser utilizadas nesta fase. Geralmente, as redes neurais só aceitam como entrada valores reais no intervalo fechado [0,1] ou [-1,1], ou, ainda, valores binários 0 ou 1. Na seção 4.5 será abordado com mais detalhes, a base de registros de conexões utilizada como entrada da RNA.

4.3.3 Seleção do modelo da Rede Neural

A seleção do modelo da Rede Neural está intimamente ligada ao tipo de problema a ser resolvido. Por exemplo, se os dados têm forte relacionamento temporal, uma melhor escolha para o modelo da rede seria do tipo *Recurrent Backpropagation*, em vez do modelo *Backpropagation* padrão. Este, por ser um modelo muito usado na classificação supervisionada de padrões sem relacionamento temporal, e por ser um método simples de treinamento, no qual os pesos são atualizados de padrão em padrão, até formar uma época, será o modelo utilizado neste trabalho.

O processo de treinamento pelo algoritmo *backpropagation* começa com a definição de um conjunto arbitrário de pesos para as conexões da rede. Em seguida, cada vetor de treinamento é apresentado à rede, e as respectivas saídas dos nós da última camada são comparadas com as saídas desejadas, e calculados os erros. Cada erro calculado é passado de volta através da Rede Neural, para cada elemento, camada por camada da RNA, e os pesos correspondentes são alterados. Todo esse processo foi realizado com o uso do simulador JNNS, detalhado na seção 4.4.

4.3.4 Especificação da arquitetura da Rede Neural

Para o desenho da rede foi necessário definir o tipo de entrada, a integração dessas entradas, o número de camadas, o número e os tipos de neurônios de entrada, de saída e intermediários, além do tipo de conectividade. O número de camadas intermediárias deve ser avaliado com cuidado, pois seu excesso pode comprometer o poder de generalização da rede e aumentar no tempo de treinamento, fazendo com que a rede apenas memorize os exemplos fornecidos sem extrair nenhum relacionamento dos dados.

Assim, para a Rede Neural de múltiplas camadas *MultiLayer Perceptron* (MLP) foi determinado o tamanho de vetor de entradas e de saída, e estimado o número de camadas e neurônios adequados para o problema.

Os procedimentos empregados para estabelecer a arquitetura da RNA utilizada neste trabalho consistiram em:

a) estabelecer o número de entradas da rede, que equivale ao número de campos existentes no registro de conexão da base de dados (KDDCUP, 1999). Inicialmente, utilizaram-se os 41 campos do registro de conexão, conforme detalhado na seção 4.3.4. Conforme será visto na seção 5.3, como forma de diminuir o tamanho da Rede Neural e melhorar seu desempenho, serão utilizados, dos 41 campos do registro de conexão, somente os de maior relevância na determinação das classes de saída;

b) empregar a heurística $N_{hidden} = 2N_{in} + 1$ (número de elementos ocultos é igual ao dobro do número de elementos de entrada mais uma unidade), com ajustes baseados em tentativa e erro:

- para uma rede, por exemplo, que receba como entrada apenas os 9 campos que compõem as características básicas do TCP/IP, a heurística determina que $N_{hidden} = 2x9 + 1 = 19$ neurônios nas camadas escondidas. Neste caso usa-se na prática uma rede com duas camadas escondidas, cada uma com 10 neurônios. Características básicas ou intrínsecas do TCP/IP são aquelas características essenciais e elementares obtidas ao se analisar um fluxo de pacotes de protocolo TCP/IP. Maiores detalhes sobre características básicas do TCP/IP serão tratados na seção 4.5,

- para uma rede que tenha 41 entradas, como a que foi utilizada no presente trabalho, onde cada entrada corresponde a um dos 41 campos do registro de conexão da base de teste/treinamento utilizada (conforme detalhado na seção 4.3.4), a heurística recomendaria 83 neurônios distribuídos em camadas escondidas. Ajustes, baseados em tentativa e erro, mostraram que é possível reduzir o número de neurônios ocultos na RNA, sem comprometer sua capacidade de generalização, diminuindo assim o tamanho da Rede Neural e seu custo de aprendizagem. Dessa forma, foram realizados ajustes

para determinar qual o menor número de neurônios atenderiam este problema. O número encontrado empiricamente variou entre 30 e 40 neurônios artificiais distribuídos em duas camadas,

c) estabelecer o número de camadas de saída, que corresponde ao número de classes ou padrões de ataques. Foram definidas as classes contendo padrão de tráfego normal (sem características de intrusões), classe com ataques do tipo DoS, com ataques do tipo probe, do tipo U2R e do tipo R2L; e

d) utilizar intervalos de valores de entrada compreendidos entre -1 e 1, um conjunto de treinamento com 30.170 registros, extraídos da base de registros de conexões nomeada de *Whole KDD* e disponível em KDD Cup (1999), com cerca de três milhões e oitocentos mil registros, e uma base de teste/validação com 35.366 registros, extraído da base nomeada de *Corrected KDD*, e disponível também em KDD Cup (1999), com cerca de duzentos mil registros de conexões. Foi utilizado o SQL para extração das amostras de treinamento e testes/validação das bases de dados da Competição KDDCup (1999). Será utilizada também uma RNA totalmente conectada, ou seja, cada neurônio estará conectado a todos os outros neurônios da camada seguinte.

4.3.5 Configuração dos parâmetros de treinamento

Nesta fase são definidos os parâmetros de aprendizado que, entre outros fatores, determinam a velocidade do treinamento e o poder de generalização da rede neural. Função de ativação e taxa de aprendizado são os parâmetros de treinamento mais gerais, ou seja, estão presentes em quase todos os tipos de redes neurais. A taxa de aprendizado é um parâmetro geral que determina o grau de atualização realizada nos pesos sinápticos, visando à saída alvo. A taxa de aprendizado é diretamente proporcional à velocidade de treinamento, mas pode acarretar na variação demasiada dos pesos sinápticos e assim causar uma maior imprecisão nos resultados gerados.

Foram realizados testes empíricos com uma RNA, com modelo e arquitetura já definidos nas seções anteriores, e chegou-se a uma taxa de aprendizagem de 0,2, e função de ativação Logística. Há também, diversos parâmetros de treinamento específicos a cada modelo de rede neural, como é o caso do fator de momento e tolerância do erro do modelo *Backpropagation*.

Em resumo, o processo de treinamento implicou em uma definição prévia dos parâmetros que são requeridos para a classificação. Nesse sentido, foi estabelecida uma taxa de aprendizagem, que corresponde ao quanto de erro é repassado aos nós a cada iteração. A função de ativação adotada foi a logística, usada para calcular o valor de ativação em cada nó das camadas intermediárias e de saída. O modo de atualização escolhido foi pela estrutura topológica, que leva em conta a organização das camadas na rede. Os pesos aleatórios iniciais foram definidos no intervalo entre -0,1 e 0,1.

4.3.6 Heurísticas para melhorar o desempenho do algoritmo *Backpropagation*

Segundo Haykin (2001), o projeto de uma rede neural utilizando o algoritmo de retropropagação é mais uma arte do que uma ciência, significando que muitos dos numerosos fatores envolvidos no projeto são resultados da experiência particular de cada um. Entretanto, ainda segundo o mesmo autor, existem métodos que melhoram significativamente o desempenho do algoritmo de retropopagação.

Algumas heurísticas abordadas por Haykin (2001) foram consideradas no presente trabalho, tais como:

a) o modo seqüencial da aprendizagem por retropopagação, envolvendo atualização de padrão em padrão, é computacionalmente mais rápida que o modo por lote, especialmente quando o conjunto de dados de treinamento for grande e altamente redundante, como é o caso da base de registros de conexões da competição KDDCup (1999) utilizada neste trabalho;

b) tornar aleatória, ou seja, embaralhar a ordem em que os exemplos são apresentados ao *perceptron* de múltiplas camadas de uma época para a seguinte. Idealmente, a aleatoriedade garante que os exemplos sucessivos apresentados à rede em uma época raramente pertençam à mesma classe; e

c) normalizar as variáveis de entrada da rede neural, sobre todo o conjunto de treinamento, de modo que os valores sejam próximos de zero ou sejam pequenos comparados ao desvio padrão. Isso evita que, por exemplo, as variáveis de entrada apresentem valores positivos de modo consistente, onde os pesos sinápticos de um neurônio da primeira camada oculta apenas cresçam ou decresçam juntos, e como o vetor peso desse neurônio deve mudar de direção, ele só pode fazer isso ziguezagueando seu caminho através da superfície de erro, o que é tipicamente lento e deve ser evitado.

4.3.7 Verificação do aprendizado da rede

Na fase de verificação do aprendizado da rede, registros de conexões com ataques inéditos e ataques já apresentados na fase de treinamento, foram mostrados à rede neural para verificar se ela realmente obteve um aprendizado satisfatório. Durante essa etapa, se a rede atinge um valor aceitável de erro, ela está treinada e pronta para o uso. Se a rede não obtém uma performance condizente, as fases de desenvolvimento anteriores deverão ser reavaliadas com o intuito de identificar a origem do problema.

A validação dos dados foi feita a cada *n* ciclos (variando a cada simulação), sendo que a partir daí foi disponibilizado o erro médio quadrático sobre esse conjunto de padrões de validação. Durante essa fase, o conjunto de validação foi utilizado para determinar o desempenho da rede com dados que não foram previamente utilizados no treinamento. Com base nesse erro e no resultado da aplicação da rede treinada à área estabelecida para validação, foram selecionados os parâmetros de rede mais adequados para classificar o conjunto de dados de entrada. Assim, a Rede Neural está pronta para o uso, pois passou no teste de verificação de aprendizado da rede

citado acima. Nesse momento, os pesos *sinápticos* guardam o “conhecimento” da rede e não são mais atualizados. Assim, a rede é capaz de classificar, agrupar e prever situações que lhe são fornecidas através de suas entradas.

4.4 FERRAMENTAS UTILIZADAS PARA SIMULAÇÕES DE RNAs

O uso de uma ferramenta disponível no mercado para simulação de redes neurais, justifica-se pelo custo e complexidade do desenvolvimento de um software próprio para manipulação de Redes Neurais e pela facilidade de obtenção de ferramentas abertas, que atendessem às necessidades do presente projeto.

MATLAB: a “toolbox” de redes neurais do software Matlab pode ser utilizada para testar o desempenho de algumas configurações de Redes Neurais Artificiais aplicadas à detecção de intrusão, utilizando-se subconjuntos de dados para treinamento e testes da RNA. A rede neural *MultiLayer Perceptron* (MLP) pode ser composta por duas ou mais camadas escondidas com elementos processadores, função de ativação, e faixa de valores de entrada das unidades neurais no intervalo [-1,1].

SNNS (*Stuttgart Neural Network Simulator*): trata-se de um simulador desenvolvido na Universidade de Stuttgart, pode ser usado para criar, modelar, treinar e testar redes neurais de diversos modelos (ZELL et al.,1995).

O SNNS possui outra vantagem: depois de treinada uma rede, pode-se gerar com o SNNS um arquivo em linguagem "C" contendo a rede treinada, utilizando o utilitário *snns2c* fornecido juntamente com a versão Unix/Linux. Esse arquivo compilado pode ser utilizado como programa *standalone* ou então como biblioteca (dll ou .so) “linkada” ao programa aplicativo que for utilizar-se da mesma. Isso é uma vantagem para as aplicações que se for desenvolver, pois permite que se integre a rede diretamente ao código da aplicação.

O SNNS possui duas versões: SNNS Padrão e JavaNNS. A versão padrão é fornecida em código-fonte e pode ser compilada para qualquer plataforma Unix/Linux. Ela possui uma série de utilitários que permitem a integração das redes neurais treinadas com o sistema em programas aplicativos. O *JavaNNS* é uma nova implementação em Java com finalidade exclusiva de ensino. Está disponível sob a forma de arquivo .jar para plataforma MS Windows e oferece apenas o ambiente de treinamento e teste interativo de redes neurais, não oferecendo nenhuma ferramenta de integração das redes em programas aplicativos.

JNNS (*Java neural Network Simulator):* baseada no núcleo do *Stuttgart Neural Network Simulator* (SNNS), a ferramenta aberta JNNS foi desenvolvida no WSI (Instituto de Ciência da Computação Wilhelm-Schickard) em Tübingen, Alemanha. A grande inovação desse simulador em relação ao SNNS é a nova interface gráfica baseada em tecnologia Java, o que lhe conferiu maior portabilidade e facilidade de uso. Essas ferramentas se equivalem quanto à usabilidade principal (desenvolvimento de redes neurais de vários tipos) e diferem em pontos específicos como exibição tridimensional das redes e geração automática de código C, tais pontos estão presentes apenas na versão SNNS.

Neste trabalho foi utilizado o simulador JNNS por ser um software de distribuição livre e possuir os principais recursos utilizados em simuladores comerciais, tais como: possibilidade de modelagem de redes *MultiLayer Perceptron* (MPL), suporta treinamento e testes de redes neurais com os principais algoritmos de aprendizagem utilizados, possibilita extrair resultados das simulações e gerar gráficos de erros nas saídas da rede neural durante as fases de treinamento e testes, permite a parametrização da RNA e das funções de aprendizagem de forma simples e exibe uma janela de *log* com todos os eventos ocorridos, dentre outras vantagens.

4.5 BASE DE REGISTROS DE CONEXÕES UTILIZADAS NO PROJETO

Neste trabalho, os dados utilizados para treinamento e testes das redes neurais, correspondem a subconjuntos da base de dados disponibilizada na competição internacional de mineração de dados *Knowledge Discovery and Data Mining Competition* - KDDCup (1999) e gerados em projeto de análise de sistemas de detecção de intrusão realizado. Essa base foi gerada através da captura, durante nove semanas, de todos os pacotes TCP/IP em uma rede real.

Uma rede de computadores, típicamente encontrada em agências governamentais norte-americanas, foi simulada e todo o tráfego de dados capturado. A rede estudada consiste em dois segmentos de rede padrão *Ethernet* interconectados por um roteador Cisco. Geradores de pacotes por software foram empregados gerando tráfego representativo de mais de 20 diferentes serviços de rede, como DNS, FTP, http, PING, POP, SMTP, Telnet, X, Finger etc. Para assegurar a precisão do modelo gerado, foram coletados por vários meses, em 50 redes diferentes da Força Aérea norte-americana, estatísticas referentes ao tráfego de rede e serviços presentes. Os geradores de pacotes foram então ajustados para gerar tráfego com o mesmo comportamento estatístico destas redes reais. Foi capturado tráfego equivalente ao funcionamento de nove semanas dessa rede, incluindo, de maneira identificada, padrões de ataques e intrusões pertencentes a quatro categorias distintas.

Os dados capturados nas nove semanas consistiram de todos os pacotes IP, TCP, UDP e ICMP capturados nessa rede em formato *tcpdum* que é uma ferramenta utilizada para monitorar os pacotes trafegados numa rede de computadores. Ela mostra os cabeçalhos dos pacotes que passam pela interface de rede.

Em 1999, os organizadores da competição internacional em algoritmos de mineração de dados KDDCUP selecionaram essa base de dados e o problema de detecção de intrusão como tema de seu concurso anual. Decidiram, entretanto, realizar uma etapa de pré-processamento na base de dados, reduzindo-a para uma base de dados de registros de conexão. Toda seqüência de pacotes entre um mesmo par de endereços IP e portas de origem e destino específicos, dentro de um

intervalo de tempo predefinido tiveram suas principais características extraídas e condensadas em um único registro de conexão com 41 campos. Logo, um pré-processamento inicial dos pacotes foi efetuado, agrupando-os em conexões. Uma conexão consiste na troca de um ou mais pacotes entre dois sistemas, para um determinado serviço, durante um intervalo definido.

Conexão foi considerada como uma seqüência de pacotes TCP começando e terminando em algum tempo bem definido, com fluxo de dados de um endereço IP de origem para um endereço IP alvo, sobre algum protocolo bem definido. Apesar do protocolo UDP não possuir o mesmo conceito de conexão que o TCP, esse protocolo estava presente no ambiente de rede simulado na competição do KDDCup (1999), e o objetivo do UDP é fornecer um mecanismo de entrega de datagramas sem nenhum tipo de verificação ou garantia de entrega. Do ponto de vista de um sistema de detecção de intrusão, entretanto, é interessante criar, mesmo para o UDP, o conceito de uma "conexão virtual", "fluxo" ou "sessão". Na prática, os sistemas de detecção de intrusos enxergarão uma "conexão virtual" sempre que uma seqüência de pacotes UDP forem trocados entre dois endereços IPs, dentro de um intervalo de tempo predefinido e mantendo portas de origem e destino fixas.

O programa *tcptrace* realiza a consolidação de vários pacotes em um arquivo capturado com *tcpdump* em registros de conexão individuais. Cada registro de conexão recebeu, ainda, um campo identificador classificando-a como uma conexão normal ou um ataque/intrusão.

A base de dados resultante possui aproximadamente 5.000.000 de registros de conexões para a base de treinamento, e cerca de 311.000 para a de teste/validação.

Diversos tipos de ataques a redes TCP/IP[2] foram inseridos e identificados na base de dados, juntamente com milhares de registros normais. Cada conexão é caracterizada por 41 variáveis, classificadas em 3 categorias principais:

[2] 22 na base de treinamento e 37 na base de teste/validação, sendo que, para esta última, 15 tipos não estavam presentes na base de treinamento sendo, portanto, novos ataques a serem apresentados à RNA

a) **características básicas do TCP/IP:** são informações essenciais e elementares obtidas ao se analisar um fluxo de pacotes de protocolo TCP/IP. Estão detalhadas no quadro 2.

NOME	DESCRIÇÃO	TIPO
Duration	Duração em segundos do fluxo	Contínua
Protocol_type	Tipo de protocolo usado no fluxo, i.e., tcp, udp, etc.	Discreta
Service	Serviço de rede sendo utilizado, i.e., http, telnet, ftp etc.	Discreta
Src_bytes	Número de bytes enviados da fonte para o destino	Contínua
Dst_bytes	Número de bytes enviados do destino para a fonte	Contínua
Flag	Status da conexão (normal ou erro)	Discreta
Land	1 se conexão é de/para o mesmo *host*; 0 caso contrário.	Discreta
Wrong_fragment	Número de fragmentos com erro.	Contínua
Urgent	Número de pacotes com flag urgente habilitado.	Contínua

Quadro 2 - Características intrínsecas de fluxos TCP/IP.
Fonte: elaborado pelo autor

b) **características sugeridas através de conhecimento da área (“*domain knowledge*”)**: são informações extraídas dos pacotes, as quais têm significado apenas com o auxílio de especialistas, para se chegar a conclusões de que padrões associados a conexões, podem representar um dado tipo de ataque. Exemplo seria o registro de tentativas de login com usuário privilegiado em determinada conexão ou o número de acessos a arquivos realizados. Estão detalhadas no quadro 3.

NOME	DESCRIÇÃO	TIPO
Hot	Número de indicadores chaves (“hot”).	Contínua
Num_failed_logins	Tentativas de login sem sucesso	Contínua
Logged_in	1 se login efetuado com sucesso; 0 caso contrário.	Discreta
Num_compromised	Número de condições de “comprometimento”	Contínua
Root_shell	1 se shell root foi obtido; 0 caso contrário	Discreta
Su_attempted	1 se comando “su root” foi tentado; 0 caso contrário	Discreta
Num_root	Número de acessos como root.	Contínua
Num_file_creations	Número de operações de criação de arquivos.	Contínua
Num_shells	Número de “shells prompts” obtidos.	Contínua
Num_access_files	Número de operações em arquivos de controle de acesso.	Contínua
Num_outbound_cmds	Número de comandos externos em uma sessão ftp.	Contínua
Is_hot_login	1 se o login pertence à lista “hot”; 0 caso contrário.	Discreta
Is_guest_login	1 se o login usou a conta guest; 0 caso contrário.	Discreta

Quadro 3 - Características de conexão por conhecimento especialista.
Fonte: elaborado pelo autor

c) **características obtidas através de uma janela de 2 segundos**: são características temporais obtidas usando uma janela de 2 segundos. Informações importantes referentes a certos ataques somente podem ser obtidas levando-se o tempo em consideração. Estão detalhadas no Quadro 4.

NOME	DESCRIÇÃO	TIPO
Count	Número de conexões iguais a esta para este mesmo “*host*” nos últimos 2 segundos.	Contínua
Srv_count	Número de conexões para o mesmo serviço que o usado nesta conexão nos últimos 2 segundos.	Contínua
Serror_rate	% de conexões que possuem erro SYN.	Contínua
Srv_serror_rate	% de conexões que possuem erro SYN para este serviço.	Contínua
Rerror_rate	% de conexões que possuem erro REJ.	Contínua

Srv_rerror_rate	% de conexões que possuem erro REJ para este serviço.	Contínua
Same_srv_rate	% de conexões para um mesmo serviço.	Contínua
Diff_srv_rate	% de conexões para serviços diferentes.	Contínua
Srv_diff_*host*_rate	% de conexões deste mesmo serviço para *host*s diferentes.	Contínua
Dst_*host*_count	Número de conexões com mesmo *host* de destino que esta.	Contínua
Dst_*host*_srv_count	Número de conexões com mesmo *host* de destino e mesmo serviço que esta.	Contínua
Dst_*host*_same_srv_count	% de conexões com o mesmo *host* de destino e mesmo serviço que esta.	Contínua
Dst_*host*_diff_srv_rate	% de conexões com o mesmo *host* de destino e services diferentes que esta.	Contínua
Dst_*host*_same_src_port_rate	% de conexões com o mesmo *host* de destino e a mesma porta de origem que a conexão atual.	Contínua
Dst_*host*_srv_diff_*host*_rate	% de conexões para o mesmo serviço vindo de diferentes *host*s.	Contínua
Dst_*host*_serror_rate	% de conexões para o mesmo *host* que o da conexão atual e que possui um erro S0.	Contínua
Dst_*host*_srv_serror_rate	% de conexões para o mesmo *host* e serviço que o da conexão atual e que possui um erro S0.	Contínua
Dst_*host*_rerror_rate	% de conexões para o mesmo *host* que apresentem flag RST.	Contínua
Dst_*host*_srv_rerror_rate	% de conexões para o mesmo *host* e serviço da conexão atual que apresentem flag RST.	Contínua

Quadro 4 - Características temporais: janela de 2 segundos.
Fonte: elaborado pelo autor

As variáveis que caracterizam cada conexão são tipificadas como discretas ou contínuas. Variáveis discretas assumem valores definidos, como 0 ou 1, normal ou erro, enquanto que as variáveis contínuas assumem uma ampla faixa de valores.

É importante ressaltar que, em certos casos, os perfis de tráfego definidos são específicos para cada hora do dia e para cada dia da semana, diferenciando dias úteis, fins de semana e feriados.

Os registros de conexões foram divididos em duas bases de dados distintas: a primeira, contendo o equivalente a sete semanas de operação da rede modelo (4.898.430 registros), foi denominada base de dados para treinamento dos algoritmos de detecção. A segunda base de dados, contendo o equivalente a duas semanas de funcionamento da rede (311.029 registros) foi denominada base de dados para testes e validação, totalizando assim as nove semanas de observação do tráfego da rede.

A tabela 1 apresenta a distribuição dos registros presentes nos dois subconjuntos, incluindo a categoria na qual cada registro está enquadrado. A base de dados de testes e validação possui 15 padrões de ataque que não existem na base de treinamento, assim como possui distribuição de probabilidade dos registros de conexões diferentes dos de treinamento. Os tipos de ataques descritos na tabela 2 e tabela 3 foram detalhados na seção 2.4.

Tabela 1 – Distribuição dos registros presentes na base de dados do KDDCUP99.

Dataset	***DoS***	***Probe***	***u2r***	***r2l***	***Normal***
"Corrected KDD" (Teste/Validação)	229853	4166	70	16347	60593
"Whole KDD" (Treinamento)	3883370	41102	52	1126	972780

Fonte: elaborada pelo autor

As tabelas 2 e 3 mostram com detalhes a distribuição das bases de treinamento e de teste/validação, disponíveis no KDD Cup (1999).

Tabela 2 - Distribuição da base de treinamento.

DISTRIBUIÇÃO DA BASE DE TREINAMENTO			
Ataques / Intrusões	**# Ocorrências**	**Percentual**	**Categoria**
spy.	2	0,00004%	R2L
perl.	3	0,00006%	U2R
phf.	4	0,00008%	R2L
Multihop.	7	0,00014%	R2L
ftp_write.	8	0,00016%	R2L
loadmodule.	9	0,00018%	U2R
rootkit.	10	0,00020%	U2R
imap.	12	0,00024%	R2L
warezmaster.	20	0,00041%	R2L
Land.	21	0,00043%	DOS
buffer_overflow.	30	0,00061%	U2R
guess_passwd.	53	0,00108%	R2L
pod.	264	0,00539%	DOS
teardrop.	979	0,01999%	DOS
warezclient.	1020	0,02082%	R2L
Back.	2203	0,04497%	DOS
nmap.	2316	0,04728%	Probing
portsweep.	10413	0,21258%	Probing
ipsweep.	12481	0,25480%	Probing
satan.	15892	0,32443%	Probing
normal.	972780	19,85902%	Normal
neptune.	1072017	21,88491%	DOS
smurf.	2807886	57,32216%	DOS
Total	**4898430**	**100,0000%**	

Fonte: elaborada pelo autor

Tabela 3 - Distribuição da base de teste/validação.

DISTRIBUIÇÃO DA BASE DE TESTES		
Ataque / Intrusão	**# Ocorrências**	**Percentual**
imap.	1	0,00032%
sqlattack.	2	0,00064%
loadmodule.	2	0,00064%
phf.	2	0,00064%
udpstorm.	2	0,00064%
perl.	2	0,00064%
worm.	2	0,00064%
ftp_write.	3	0,00096%
xsnoop.	4	0,00129%
xlock.	9	0,00289%
land.	9	0,00289%
teardrop.	12	0,00386%
xterm.	13	0,00418%
rootkit.	13	0,00418%
ps.	16	0,00514%
named.	17	0,00547%
sendmail.	17	0,00547%
multihop.	18	0,00579%
buffer_overflow.	22	0,00707%
nmap.	84	0,02701%
pod.	87	0,02797%
httptunnel.	158	0,05080%
ipsweep.	306	0,09838%
portsweep.	354	0,11382%
saint.	736	0,23663%
processtable.	759	0,24403%
apache2.	794	0,25528%
mscan.	1053	0,33855%
back.	1098	0,35302%
warezmaster.	1602	0,51506%
satan.	1633	0,52503%
snmpguess.	2406	0,77356%
guess_passwd.	4367	1,40405%
mailbomb.	5000	1,60757%
snmpgetattack.	7741	2,48884%
neptune.	58001	18,64810%
normal.	60593	19,48146%
smurf.	164091	52,75746%
Total	**311029**	**100,00000%**

Fonte: elaborada pelo autor

Assim, a monitoração e captura dos pacotes utilizando o software *tcpdump* foi realizada pelo projeto MIT/Darpa. A primeira etapa de pré-processamento, juntamente com a etapa de extração dos conjuntos de características (temporais, intrínsecas e especialistas) foi realizada pelos organizadores do KDD Cup (1999).

As demais etapas de processamento dos dados da base de registros de conexões foram parte integrante do escopo deste trabalho, e serão detalhadas adiante.

5 IMPLEMENTAÇÃO E RESULTADOS

Neste capítulo, será apresentada a implementação propriamente dita, da detecção de intrusão por anomalias utilizando Redes Neurais, envolvendo o processamento da base de dados dos registros de conexões, usando linguagem SQL, a implementação da Rede Neural usando o simulador JNNS, os resultados obtidos com todos os 41 campos dos registros de conexões, a comparação desses resultados com os da competição KDD Cup (1999), e os resultados obtidos usando-se apenas as características mais relevantes dos registros de conexões, visando diminuir o tamanho da RNA, diminuido o tempo de treinamento e, consequentemente, de classificação da mesma.

5.1 PROCESSAMENTO DOS REGISTROS DE CONEXÔES COM O USO DO SQL

A ferramenta SQL foi utilizada neste trabalho, com o objetivo de adequar os campos de cada registro de conexão da base de dados do KDDCUP99, para apresentá-los à Rede Neural. Transformação de registros não numéricos em valores numéricos, normalização de valores e criação de tabelas de saída com classes de ataques obtidas aleatoriamente na base de registro de conexões, são exemplos de tarefas realizadas pelo SQL. A figura 5 mostra, de forma genérica, as etapas de processamento aplicadas à base de registros de conexões (KDD Cup, 1999).

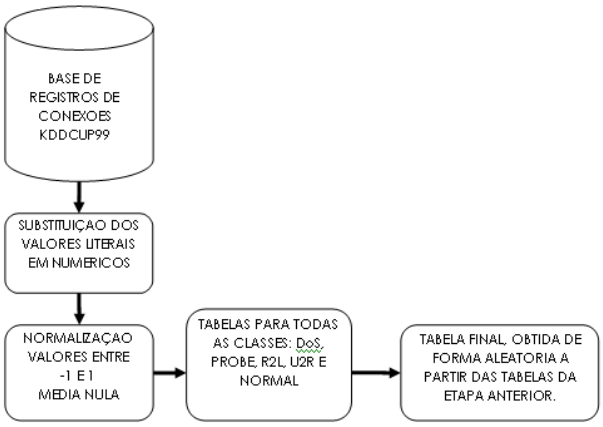

Figura 5 – Etapas gerais de processamento da base de dados.
Fonte: elaborada pelo autor

A figura 6 mostra, de maneira detalhada, a implementação das etapas de processamento dos scripts no SQL, utilizadas neste trabalho, para tratamento dos registros de conexões. As etapas são idênticas tanto para a base usada para treinamento quanto para a base usada para teste/validação da RNA. A base *kddcup.data* foi usada para treinamento, e a base *corrected* (com rótulos corrigidos) foi usada para fins de teste e validação da Rede Neural, ambas disponíveis em (KDD Cup, 1999). O Apêndice 1 mostra os códigos dos scripts utilizados em cada etapa.

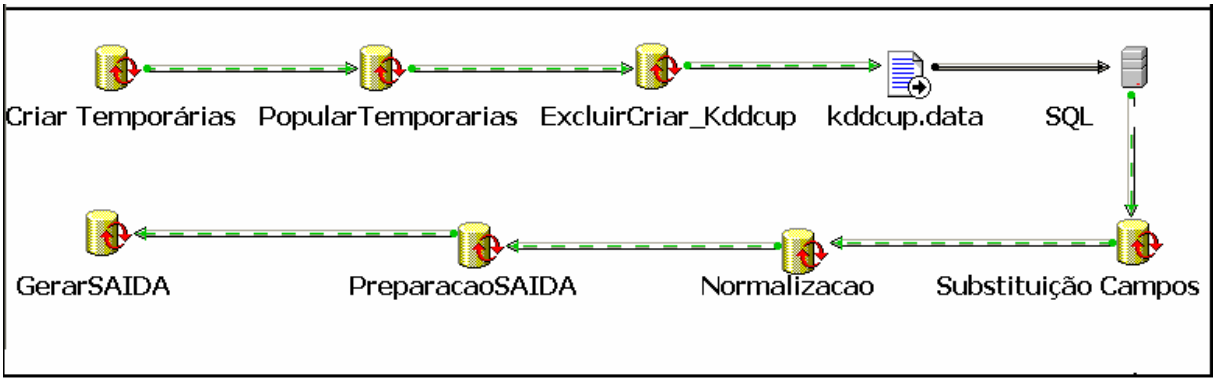

Figura 6 - Etapas de processamento dos Script's no SQL.
Fonte: elaborada pelo autor

A etapa "Criar Temporárias" e "Popular Temporárias" criam as tabelas temporárias e as populam, com a associação entre valores literais e numéricos, que posteriormente serão utilizadas para auxiliar na substituição de todos os campos com valores não numéricos em valores numéricos.

As três etapas seguintes são responsáveis por criar uma estrutura de tabela chamada tabela KDDCup e migrar à base de dados original da competição KDD Cup (1999), seja à base de treinamento ou a de teste/validação.

A etapa "Substituição Campos" faz a substituição dos campos não numéricos em valores numéricos da tabela KDDCup criada na etapa anterior, utilizando as tabelas temporárias criadas na primeira etapa.

A etapa “Normalização” ajusta todos os valores da tabela KDDCup – 99, agora todos numéricos, para o intervalo [-1;1], de modo a compatibilizar com o intervalo de valores aceitos nas entradas da rede neural. Isso porque as variáveis da base de dados da competição KDDCUP não estavam normalizadas e, portanto, não podiam ser apresentadas como entrada de uma rede neural. Por isso foi realizado esse processamento dos dados, que consistiu na transformação de todos os registros em valores numéricos, normalizados no referido intervalo.

As duas etapas seguintes preparam a estrutura e criam as tabelas de saída, as quais serão utilizadas como entrada da rede neural. Cada tabela de saída possui registros de uma classe de ataque, juntamente com registros normais (sem a presença de qualquer classe de ataque), obtidos aleatoriamente na tabela KDDCup.

Nesta etapa foram incluídos os cabeçalhos de cada registro, tanto para os valores de entrada quanto para os de saída da RNA (no caso da base de treinamento usada no aprendizado supervisionado).

Assim, o SQL foi escolhido como ferramenta, pela sua robustez na manipulação de grandes volumes de dados, bem como sua facilidade de uso e implementação de *script's*. Nos trabalhos de Batista e outros (2006), Braga e outros (2004) e Santos (2007), são apresentados estudos envolvendo manipulação de dados, voltados tanto para detecção de intrusão quanto para outras finalidades.

5.2 RESULTADOS OBTIDOS COM TODOS OS 41 CAMPOS DOS REGISTROS DE CONEXÕES

As saídas geradas pelo processamento da base de registros de conexões no SQL foram apresentadas à RNA, primeiro para treinamento da mesma depois para fins de testes e validação.

Testes comparativos indicaram um melhor desempenho da rede com as características apresentadas na tabela 4.

Tabela 4 – Parâmetros utilizados na RNA

PARÂMETRO	VALOR
Algoritmo de treinamento	Backpropagation
N° nós de fonte na camada de entrada	41
N° neurônios na camada de saída	5
N° de camadas intermediárias	2
N° de neurônios em cada camada intermediária	15
Função de ativação das camadas intermediárias	Logística
Função de ativação da camada de saída	Logística
Taxa de Aprendizagem	0,2
Modelo da Rede Neural	Multilayer *Perceptron* Feed-forward
N° de épocas de treinamento	100
N° de amostras do conjunto de treinamento	30170
N° de amostras do conjunto de teste/validação	35366
Base de Treinamento	Whole KDD
Base de Teste/Validação	Corrected KDD
Erro mínimo	1
Intervalo de valores de saída dos neurônios	de -1,0 a 1,0
Intervalo de pesos aleatórios iniciais	de -1,0 a 1,0

Fonte: elaborada pelo autor

Abaixo, exemplificamos um registro de conexão já processado com o uso do SQL para ser apresentado à RNA:

INPUT PATTERN 1: -1.000 0.000 -0.970 0.250 -0.989 -0.990 0.000 -1.000 0.000 -1.000 0.000 1.000 -1.000 -1.000 0.000 -1.000 -1.000 0.000 0.000 0.000 0.000 0.000 -0.984 -0.984 -1.000 -1.000 -1.000 -1.000 1.000 -1.000 -1.000 -0.425 1.000 1.000 -1.000 -0.980 -0.920 -1.000 -1.000 -1.000 -1.000 # OUTPUT PATTERN 1: 1.000 0.000 0.000 0.000 0.000

A Rede Neural foi simulada usando o *Java Neural Network Simulator*, na versão 1.1 do software e está mostrada na figura 7.

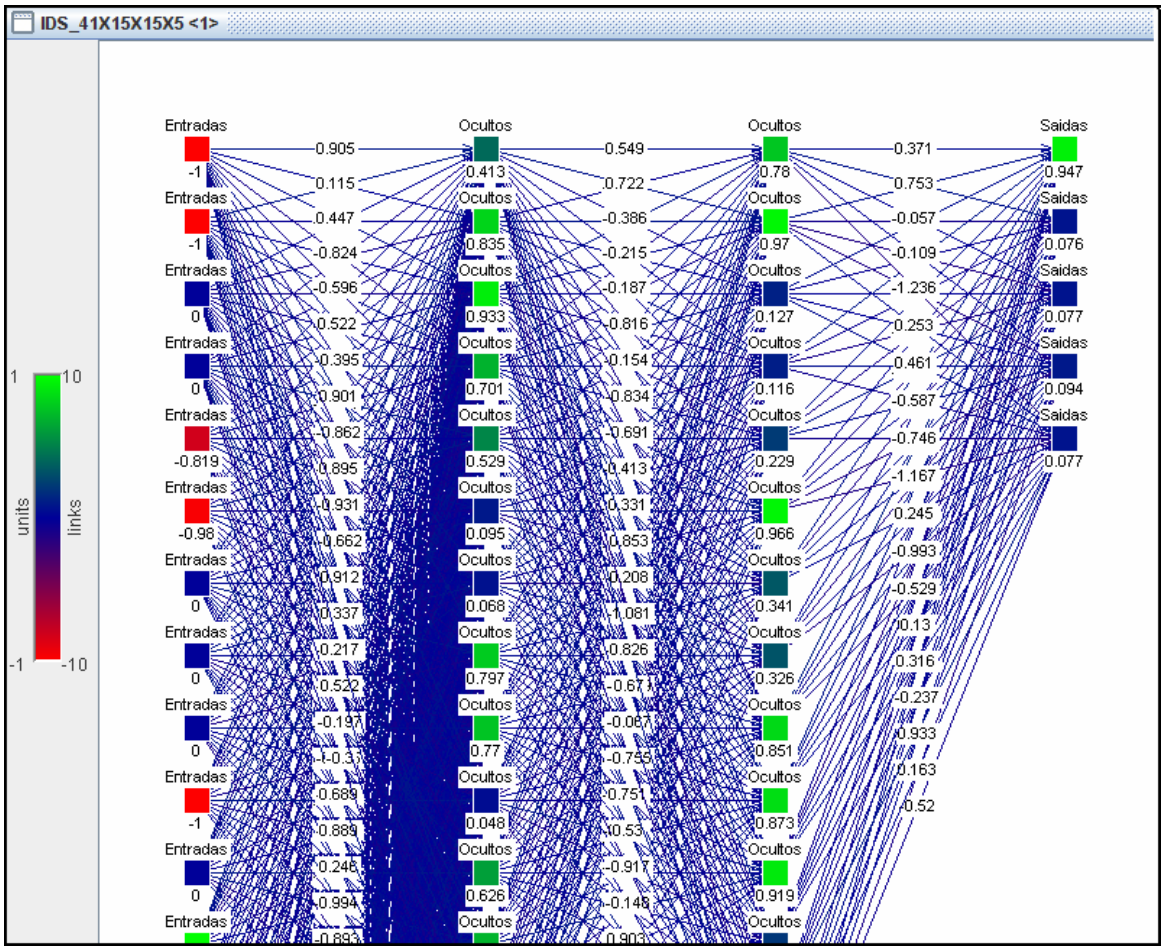

Figura 7 - Rede Neural do tipo MLP 41x15x15x5 com algoritmo de aprendizagem *Backpropagation.*
Fonte: elaborada pelo autor

Cada uma das cinco saídas da rede neural representa um padrão de ataque, ou tráfego normal, para cada padrão de entrada apresentado à rede. Assim, o neurônio de saída de número 72, que representa o tráfego normal, vai para nível 1, quando o registro de conexão apresentado à rede neural for reconhecido como um padrão de tráfego normal pela RNA. De modo semelhante, o neurônio de saída de número 73, que representa o ataque da classe DoS, vai para nível 1, quando o registro de conexão apresentado à rede neural for reconhecido como um ataque da classe DoS. Para as demais saídas, temos: saída 74 da RNA representando ataques da classe

probe, saídas 75 representando ataque da classe R2L, saídas 76 representando ataque da classe U2R.

A figura 8 mostra a soma do quadrado dos erros apresentado pela RNA durante o processo de aprendizagem, indicando assim que o referido erro estabilizou durante o processo. No cálculo da soma dos quadrados dos erros, a diferença entre a saída gerada pela RNA, e a real classe a que pertence o padrão de entrada, é computada, para cada unidade de saída. Em seguida, é calculada a soma dos quadrados das diferenças, para todas as saídas da rede neural.

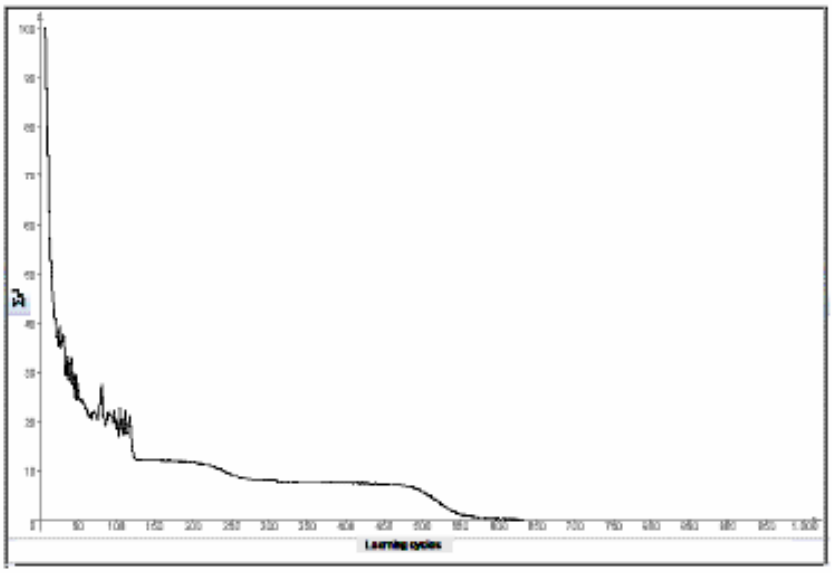

Figura 8 - Erro da Rede Neural.
Fonte: elaborada pelo autor

Para a simulação da Rede Neural, foi utilizada uma estação Intel Celeron 1,60GHz, memoria RAM de 1Gb com MS Windows XP Professional e Service Pack 3, onde o tempo médio de treinamento da rede simulada foi 2,85 horas, para as 30.170 amostras do conjunto de treinamento, sendo que o tempo médio de propagação seguida de retro-propagação de um ciclo foi de 3,4 ms. O tempo médio de teste/validação da rede foi de 3,34 horas para as 35.366 amostras do conjunto de teste/validação.

No apêndice B é apresentado um exemplo de resultados obtidos com o uso da ferramenta de simulação de RNA.

Cada saída da rede neural, representando uma classe de ataque/intrusão, foi comparada com a classe real da base de registro de conexões, utilizando o MSExcel, conforme exemplificado na tabela 5, e o resultado da comparação de cada registro (classe real da intrusão versus classificação feita pela RNA) é avaliado e apresentado na tabela 6, indicando, assim, a quantidade de previsões corretas e incorretas feitas pela rede neural. As previsões corretas, na tabela 6, correspondem à interseção de uma coluna com uma linha de mesma classe, ou seja, quando a classe real de um registro de conexão (coluna) corresponde à classe que a Rede Neural determinou quando esse mesmo registro foi apresentado à entrada da RNA (linha). Por outro lado, as previsões incorretas são todas aquelas onde a classe real de um registro de conexão (coluna) corresponde a qualquer outra classe de saída da Rede Neural (linha) diferente daquela a que o registro de conexão realmente pertence. As quantidades de previsões corretas estão ressaltadas em negrito na tabela 6.

Tabela 5 – Saídas da RNA e respectivas classes reais para cinco padrões.

Padrão	RNA Saída [72]	RNA Saída [73]	RNA Saída [74]	RNA Saída [75]	RNA Saída [76]	Valor Real Classe Normal	Valor Real Classe DoS	Valor Real Classe Probe	Valor Real Classe R2L	Valor Real Classe U2R
1	0	0	1	0	0	0	0	1	0	0
2	1	0	0	0	0	1	0	0	0	0
3	1	0	0	0	0	1	0	0	0	0
4	0	1	0	0	0	0	1	0	0	0
5	1	0	0	0	0	1	0	0	0	0

Fonte: elaborada pelo autor

Tabela 6 - Quantidades de previsões corretas e incorretas por classe.

VALOR DE SAÍDA DA REDE NEURAL		VALOR REAL DA BASE DE REGISTRO DE CONEXÕES				
		NORMAL	DoS	PROBE	R2L	U2R
SAÍDA 72	NORMAL	**7952**	263	129	1708	7
SAÍDA 73	DoS	66	**9078**	0	0	0
SAÍDA 74	PROBE	37	3	**339**	2	0
SAÍDA 75	R2L	0	0	0	**6**	0
SAÍDA 76	U2R	8	1	0	143	**0**

Fonte: elaborada pelo autor

A tabela 7 mostra o percentual de detecção por classe, indicando que a detecção de tráfego normal, e a dos ataques dos tipos DoS e probe, tiveram excelente percentual de detecção pela rede neural, mas em contrapartida, ataques das classes R2L e U2R não tiveram percentual de detecção significativo pela RNA. Ainda, pode-se verificar na referida tabela, que o sistema apresentou baixo percentual de falsos positivos, ou seja, apenas 1,38% do tráfego normal, foi classificado erradamente como pertencente a uma das classes de ataques (falsos positivos), e o restante, 98,62% do tráfego normal, foram corretamente classificados como normal pela rede neural.

Tabela 7 – Percentual de detecção por classe.

NORMAL	98,62%
DoS	97,14%
PROBE	72,44%
R2L	0,32%
U2R	0,00%

Fonte: elaborada pelo autor

As duas classes U2R e R2L foram classificadas com percentagens de êxito de detecção relativamente baixas, devido ao reduzido número de amostras no conjunto de treinamento. Em outras palavras, não foi possível, com a quantidade de ataques das classes U2R e R2L, presentes no conjunto de dados de treinamento, estabelecer a identidade das mesmas no interior da Rede Neural. Logo, quaisquer ataques, sejam ataques já apresentados à rede neural na etapa de treinamento (conhecidos) ou novos ataques (desconhecidos), pertencentes a essas duas classes, poderão não ser corretamente classificados pela RNA. Assim, a RNA classificou a maioria dos ataques de classes U2R e R2L como sendo de classe NORMAL, pois o conjunto de características do registro de conexão, que representava um ataque como pertencente às classes U2R e R2L, não foi suficientemente caracterizado pela RNA durante a etapa de aprendizagem.

É importante lembrar que o desempenho de reconhecimento da rede treinada, ou seja, sua capacidade de generalização (um termo emprestado da psicologia) está relacionada com a capacidade da rede de estabelecer uma associação correta entre a classe real a que pertence um registro de conexão e com a classe indicada pela RNA, como resultado do reconhecimento feito pela Rede Neural deste registro de conexão. Assim, se uma determinada classe não foi suficientemente caracterizada durante o treinamento da rede, sua capacidade de generalização para todos os ataques já apresentados ou não à RNA na etapa de treinamento, pertencentes a esta classe, fica comprometida.

Na tabela 8, pode-se fazer uma comparação entre os resultados obtidos neste trabalho (segunda coluna da tabela), e o resultado obtido pelo vencedor da

competição do KDDCup (terceira coluna da tabela), o Dr. Pfahringer (2000), do Instituto Austríaco de Investigação para a Inteligência Artificial, que utilizou árvores de decisão em seu trabalho. Nessa comparação, percebe-se uma similaridade nos resultados, para os tipos de ataques, cujas classes foram suficientemente caracterizadas durante o treinamento da Rede Neural (Normal, DoS e Probe).

Tabela 8 - Comparação dos resultados

Tipos de Ataques	Resultados deste Trabalho	Resultados KDDCUP
NORMAL	98.62%	99.50%
DoS	97.14%	97.10%
PROBE	72.44%	83.30%
R2L	0.32%	8.40%
U2R	0.00%	13.20%

Fonte: elaborada pelo autor

Uma ou outra abordagem, usando Árvores de Decisão ou Redes Neurais, é vastamente utilizada na literatura, envolvendo problemas de classificação de padrões, com vantagens e desvantagens para cada uma dessas abordagens (PAULA, 2002). Assim, encontramos na literatura duas abordagens básicas para os problemas de classificação: a abordagem simbólica, baseada em árvores de decisão e a abordagem conexionista, baseada, principalmente, em Redes Neurais.

A vantagem da Rede Neural na classificação de padrões é a grande tolerância a falhas, devido ao seu paralelismo inerente, o que torna a implementação das Redes Neurais como solução prática, em sistemas de alta disponibilidade, algo vantajoso.

O paralelismo não é somente um conceito ao processamento de informação em redes neurais, mas é também a fonte de sua flexibilidade. Além disso, o paralelismo pode ser maciço (centenas de milhares de neurônios), o que dá às redes neurais uma forma notável de robustez. Como a computação está distribuída sobre muitos neurônios, normalmente não importa muito se os estados de alguns neurônios da

rede se desviarem de seus valores esperados. Entradas ruidosas ou incompletas ainda podem ser reconhecidas, uma rede danificada pode ainda ser capaz de funcionar satisfatoriamente, e a aprendizagem não precisa ser perfeita. O desempenho da rede se degrada suavemente dentro de certo limite (HAYKIN, 2001).

Além disso, a modelagem direta do problema, ou seja, a possibilidade de apresentar à RNA os dados de entrada, sem a necessidade de interpretação de como são processados os dados no interior da rede – para os casos onde não se faz necessária a visualização de forma trivial – o conhecimento adquirido no processo de aprendizado pode ser uma vantagem no uso das Redes Neurais, pois isso implica em simplicidade na implementação. Por outro lado, nos casos onde se faz necessária a visualização do conhecimento adquirido no processo de aprendizado, ou seja, quando se deseja saber quais as regras que levaram à tomada de certa decisão, a técnica de Árvores de Decisão é a mais adequada.

Assim, as Redes Neurais, embora possuindo vantagens sobre a abordagem usando árvores de decisão, como paralelismo e capacidade de generalização, quando as classes são suficientemente caracterizadas durante o processo de treinamento, encerram uma enorme dificuldade no que diz respeito à interpretação do seu funcionamento. Embora o seu desempenho seja por vezes verdadeiramente impressionante, é normalmente difícil a análise dos valores dos pesos da rede em termos de se conhecer como é efetuada a classificação e, portanto, qual a essência do fenômeno. Como é evidente, este é um fato pouco ou nada limitativo quando o objetivo é apenas a classificação.

Por outro lado, embora as Redes Neurais apresentem essa enorme dificuldade no que diz respeito à sua interpretação, esse problema pode, em princípio, ser contornado. Mantendo uma baixa complexidade nas redes colocadas em cada nó, é possível facilitar a interpretação de cada uma delas, possibilitando, assim, uma explicação global para o problema.

Na seção seguinte, será abordada uma forma de reduzir o tamanho da Rede Neural, através da utilização de apenas alguns campos dos registros de conexões, considerados essenciais para a determinação dos ataques presentes na base de registros de conexões. Com o uso dos campos essenciais, haverá a diminuição do tamanho da rede e, como conseqüência, a diminuição do tempo de treinamento e classificação, visando obter um melhor desempenho da rede neural, sem contudo comprometer os resultados obtidos na classificação dos ataques.

5.3 RESULTADOS OBTIDOS COM APENAS 29 CAMPOS, CONSIDERADOS DE MAIOR RELEVÂNCIA NOS REGISTROS DE CONEXÕES

De modo a obter um melhor desempenho da Rede Neural, no processo de detecção de intrusão, foram selecionadas apenas as características de maior relevância para a determinação das classes de ataques presentes nos registros de conexões da base KDD Cup (1999).

Um melhor desempenho da rede neural na detecção de intrusão possibilita o uso futuro dessa técnica na detecção em tempo real de ataques em redes de computadores. As características relevantes dos registros de conexões da base KDD Cup (1999), foram determinadas por Kayacik, Zincir-Heywood e Heywood (2005), dentre as 41 características existentes em cada registro, através de uma abordagem baseada no ganho de informação de cada característica para cada classe. Com base na entropia de um recurso, o ganho de informação mede a relevância de um recurso para uma dada classe, ou seja, o seu papel na determinação da classe de um ataque.

Uma dada característica de um registro de conexão é considerada relevante se, em outras palavras, for extremamente útil para uma determinação rigorosa das classes de ataque a que aquele registro de conexão pertence (BOUZIDA et al., 2005). Em (KAYACIK; ZINCIR-HEYWOOD; HEYWOOD, 2005), uma característica relevante é expressa em termos de ganho de informação, que é maior para características mais discriminatórias.

A figura 9 mostra o máximo de ganho de informações para cada característica do registro de conexão presente na base (KDDCUP, 1999). Há 10 características com ganho de informação máximo menor do que 0,001, e dessa forma, contribuem muito pouco para a detecção de intrusões. Além disso, as características 20 e 21 (outbound command count for FTP sessions e hot login, respectivamente), não mostram quaisquer ganhos de informação, por isso, eles não têm qualquer relevância para a detecção de intrusões. As características determinadas como relevantes no processo de classificação dos registros de conexões em classes de ataques, ou seja, aquelas que podem discriminar com precisão essas classes, foram utilizadas na rede neural, a qual passou a apresentar 29 entradas, para as 29 características consideradas relevantes por Kayacik, Zincir-Heywood e Heywood (2005).

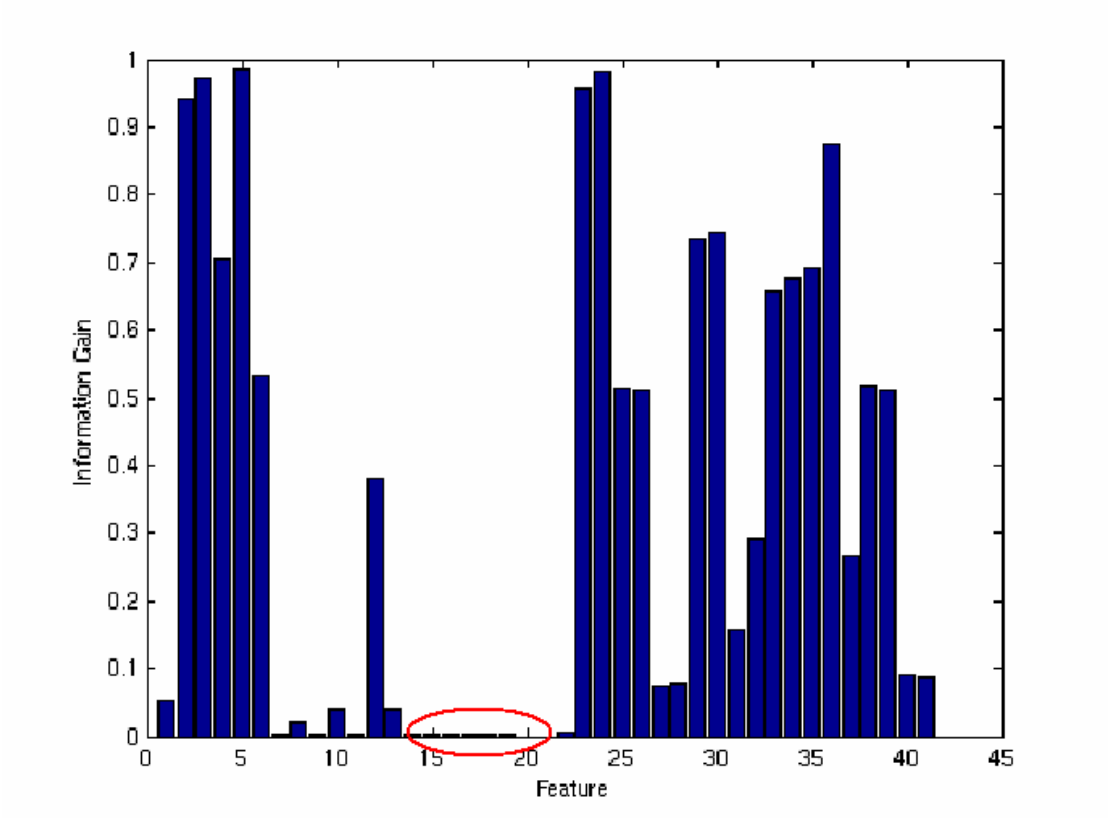

Figura 9 - Máximo ganho de informações para cada característica do registro de conexão.
Fonte: ZINCIR-HEYWOOD, 1999.

Para uma rede que tenha 29 entradas, como a que será utilizada para classificação, usando somente as características consideradas relevantes da base de registros de conexões, a heurística recomendaria 59 neurônios distribuídos em camadas escondidas. Testes com este número de neurônios demonstraram que a rede era capaz de aprender em excesso os vetores de entrada, prejudicando a sua capacidade de generalização. Quando a rede aprende um número excessivo de exemplos, acaba memorizando os dados do treinamento, ou seja, uma rede treinada em excesso perde a capacidade de generalização dos padrões de entrada-saída similares (há excesso de ajuste, um fenômeno chamado de *overfitting*, ou seja, a super-adaptação dos parâmetros da rede aos padrões de treinamento). Assim, quando se tem um conjunto de dados pequeno deve-se tomar cuidado com a utilização de um número excessivo de épocas, para evitar que a Rede Neural "memorize" os dados de treinamento sem, no entanto, aprender sobre o comportamento de entrada e saída desses dados.

Ajustes, baseados em tentativa e erro, foram realizados para determinar qual o menor número de neurônios que atenderiam a esse problema. Empiricamente, o número encontrado variou entre 20 e 30 neurônios artificiais distribuídos em duas camadas. Assim, foi utilizada uma Rede Neural com 24 neurônios ocultos, distribuídos em duas camadas ocultas.

A tabela 9 mostra os resultados obtidos, utilizando-se como entrada da rede neural apenas os parâmetros de maior relevância para a classificação dos ataques, segundo Kayacik, Zincir-Heywood e Heywood (2005)

Tabela 9 - Resultados utilizando-se as características relevantes

Tipos de Ataques	Resultados com Todas as Características	Resultados com Características Relevantes	Ganho no Percentual de Detecção com uso das Características Relevantes
NORMAL	**98,62%**	**99,60%**	**0,98%**
DoS	**97,14%**	**97,07%**	**-0,07%**
PROBE	**72,44%**	**68,64%**	**-3,80%**
R2L	**0,32%**	**2,96%**	**2,64%**
U2R	**0,00%**	**0,00%**	**0,00%**

Fonte: elaborada pelo autor

Dessa forma, os resultados mostraram-se satisfatórios, pois estão bem próximos daqueles encontrados anteriormente, quando foram usadas todas as 41 características dos registros de conexões. Além disso, houve uma redução de 48 minutos (24%), em média, no tempo de classificação da Rede Neural, devido à diminuição no tamanho da rede.

A Rede neural, por apresentar menor número de neurônios de entrada e nas camadas ocultas, foi capaz de realizar a classificação dos ataques de forma mais rápida e eficaz, o que contribui para o uso dessa técnica em trabalhos futuros, na classificação em tempo real dos padrões de ataques em redes de computadores.

6 CONCLUSÕES

A seguir, serão apresentadas as conclusões alcançadas com o presente trabalho e propostas para trabalhos futuros.

6.1 CONSIDERAÇÕES FINAIS

Como apresentado neste trabalho, o sistema de detecção de intrusão por anomalias, utilizando Redes Neurais Artificiais, procurou alcançar uma baixa taxa de erros no processo de detecção de padrões de ataques, além de ser capaz de detectar intrusões ainda não conhecidas pela rede, mas pertencentes a uma das classes de ataques caracterizadas durante o treinamento da rede neural.

O modelo elaborado neste trabalho possui arquitetura baseada em rede, e método de detecção que incorpora funcionalidades de anomalia. Os resultados gerados pelo processo de análise classificaram os eventos analisados em padrões considerados intrusivos ou normais.

Logo, a RNA classificou os tipos de ataques presentes na base de registros de conexões KDD Cup (1999), assim como classificou novos ataques com base nas características apresentadas, para que o analista (elemento humano) pudesse tomar alguma ação contra o novo ataque, por similaridade a algum já conhecido.

A fonte de dados de ataques utilizada foi a base de registros de conexões da competição KDD Cup (1999). Como ferramenta de simulação de Redes Neurais, foi utilizado o Java Neural Network Simulator (JNNS), uma ferramenta de distribuição livre e com os recursos necessários ao desenvolvimento deste trabalho.

Foi utilizada neste trabalho uma rede neural com o número de entradas correspondente a todas as características presentes nas conexões, que são as características básicas, as características por conhecimento especialista, e as temporais. A rede foi projetada para classificar os registros de conexões, utilizando uma saída para cada classe, de modo a determinar se o registro da entrada corresponde a uma conexão normal (livre de ataques), ou aos ataques das classes *Denial of Service* (DoS), *probe*, *User to Root* (U2R) ou *Remote to Local* (R2L).

Um desempenho ruim do modelo de detecção de anomalias sugere que as fases de treinamento e testes não foram suficientes ou ainda que as características utilizadas na detecção, ou os algoritmos de classificação, devem ser refinados. Diversos testes foram necessários até que um bom modelo de detecção de anomalias fosse produzido. Em um treinamento bem sucedido, o erro diminui com o aumento do número de iterações e o procedimento converge para um conjunto estável de pesos. A rede neural simulada neste trabalho atingiu um valor aceitável de erro, estando, portanto, treinada e pronta para o uso.

Os resultados usando todas as características da base de registros de conexões, foram considerados satisfatórios, quando comparados com os dados obtidos pelo vencedor da competição KDDCup (1999). A baixa percentagem de êxito de detecção para as classes U2R e R2L ocorreram devido ao reduzido número de amostras dessas duas classes no conjunto de treinamento, não tendo sido portanto, suficiente para um bom treinamento da rede neural para essas duas classes de ataques.

Os resultados obtidos com a utilização somente das características consideradas essenciais dos registros de conexões, para a classificação dos ataques, foram satisfatórios, pois se aproximaram bastante dos resultados obtidos anteriormente, com o uso de todos os campos do registro de conexão. Dessa forma, com o uso de uma rede neural com menos nós de fonte e menos neurônios nas camadas ocultas, associados à heurísticas que dão maior eficácia às Redes Neurais na classificação de padrões, é possível utilizar-se dessa técnica no desenvolvimento de ferramentas

mais eficazes, baseadas em Redes Neurais, para detecção em tempo real, de ataques em redes de computadores reais.

Dificuldades foram encontradas na modelagem adequada dos dados de interesse para entrada e processamento, segundo a arquitetura da rede neural, e na interpretação dos dados obtidos com a classificação dos padrões.

6.2 PROPOSTAS PARA TRABALHOS FUTUROS

A utilização de um conjunto de redes neurais, ou seja, várias redes, cada uma especializada em detectar uma classe de ataques, ou até mesmo especializada em categorias específicas das variáveis de entrada, pode aumentar a precisão da resposta do sistema, diminuindo a quantidade de falso-positivos.

Refinar a codificação do programa, realizar testes com dados reais de conexões de rede e implementar a aplicação em tempo real são atividades a serem exploradas em trabalhos futuros.

Para o treinamento da rede neural com tráfego e ataques reais é necessário, entre outros requisitos, o desenvolvimento de uma ferramenta que gere padrões para a rede neural, com base em arquivos de fluxos coletados por alguma ferramenta de captura de pacotes e análise de fluxos. Dessa forma, primeiramente os fluxos serão armazenados em disco, utilizados para gerar um arquivo de dados. A partir dos dados armazenados será possível selecionar intervalos de tempo para a geração do arquivo de padrões na segunda fase. A rede neural poderá, então, ser treinada a partir das informações fornecidas no arquivo de padrões. Depois de treinada, a rede neural poderá ser agregada a uma ferramenta para monitoramento de fluxos em tempo real.

Em aplicações reais de IDS, a amostragem de pacotes é geralmente utilizada por ser computacionalmente eficiente, exigindo menos recursos para implementação em equipamentos de rede, e é atualmente implementado em roteadores de alto tráfego, através de ferramentas como a Netflow, da Cisco. Assim, muitos provedores de serviço de internet (ISP) estão utilizando a amostragem de pacotes para obter informações de tráfegos de rede diretamente de roteadores.

Em Brauckhoff e outros (2006) e Mai e outros (2006), são abordados os impactos da amostragem de pacotes sobre os resultados da detecção por anomalias de tráfego em redes de computadores. A amostragem é inerentemente um processo de perdas, onde muitos pacotes são descartados sem inspeção. Assim, é importante o estudo das técnicas de amostragem utilizadas, e o efeito de cada uma delas na deterioração do desempenho de algoritmos usados para detecção de anomalias em tráfego de redes. Além disso, outros desafios surgem no desenvolvimento de um IDS para detecção em tempo real de anomalias de tráfego em redes, tais como: tempo resposta do IDS e dificuldade de treinamento da rede neural com tráfego real.

O desenvolvimento desses IDS, para detecção em tempo real, se depara com outro grande problema: a dificuldade de treinamento com tráfego real e ataques reais. Amostras de tráfego real podem apresentar algum tipo de tráfego malicioso (ruído) não identificado. A aplicação desse tráfego malicioso no treinamento da rede neural pode afetar o valor dos pesos dos neurônios, ocasionando erros na detecção. É difícil classificar uma quantidade grande de tráfego real, identificando todos os ataques existentes como pacotes mal formados, pacotes fragmentados etc.

A figura 10 apresenta a proposta de um IDS para detecção de intrusão em tempo real, indicando os principais pontos objetos de pesquisa.

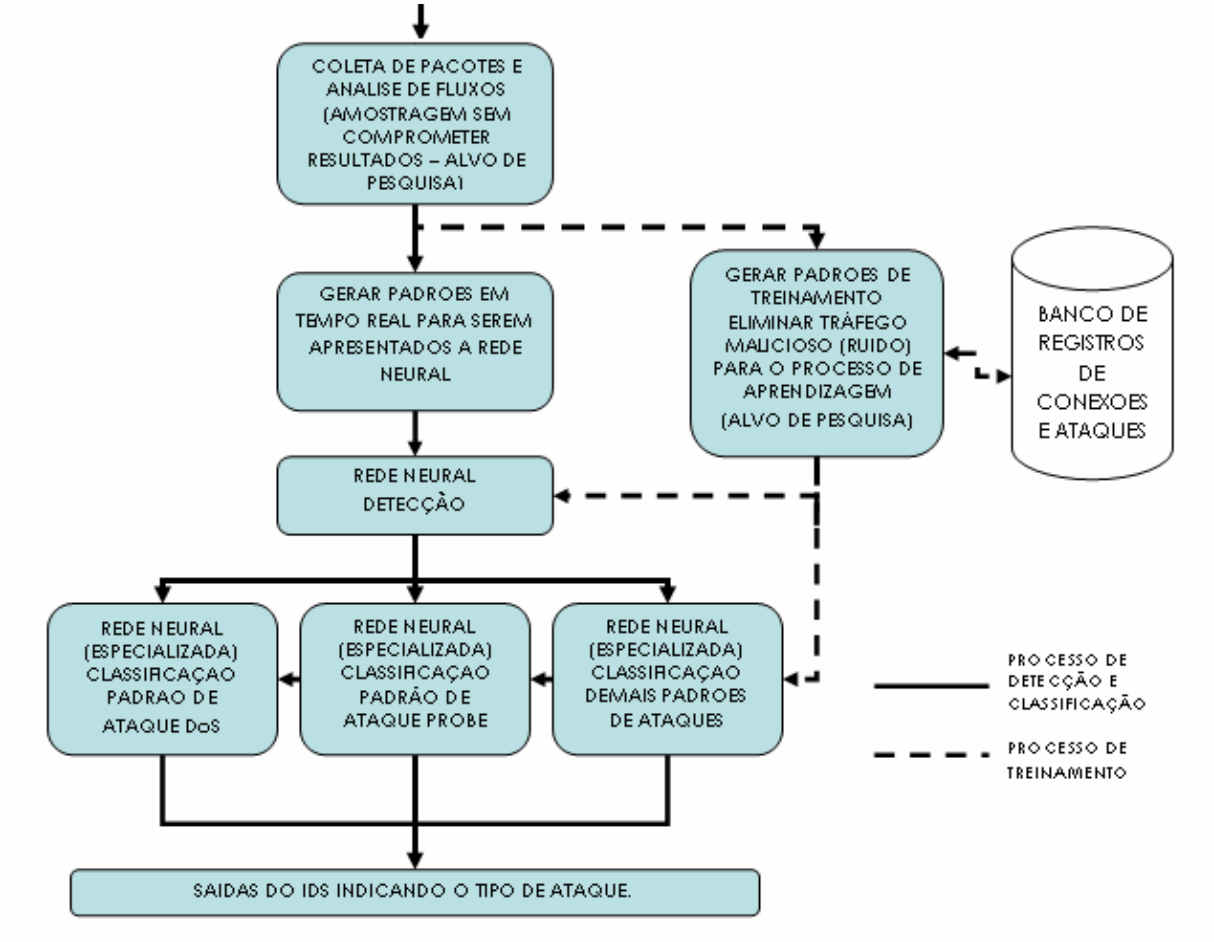

Figura 10 - Proposta de IDS para detecção em tempo real.
Fonte: elaborada pelo autor

REFERÊNCIAS

BATISTA, Luana et al. Mineração de Dados para Tratamento de Alarmes em Distribuição de Energia Elétrica. In: SEMINÁRIO TÉCNICO DE PROTEÇÃO E CONTROLE, 8., 2005, Recife. **Anais eletrônicos**... Recife, 2005. Disponível em: <http://www.dsc.ufcg.edu.br/~luana/articles/STPC05/ST_17.pdf>. Acesso em: mar. 2008.

BOUZIDA, Yacine et al. **Efficient intrusion detection using principal component analysis**. France, 2005. Article. Departement RSM GET. Disponível em: <http://www.rennes.enst-bretagne.fr/~fcuppens/articles/sar04.pdf>. Acesso em: jun. 2008.

BRAGA, Bruno de Rocha et al. **IDSMinder:** data mining de modelos de detecção de intrusão. Rio de Janeiro: UFRJ, 2004. Disponível em: <http://clusterminer.nacad.ufrj.br/TechReport/RT06.pdf>. Acesso em: out. 2008.

BRAUCKHOFF, Daniela et al. **Impact of packet sampling on snomaly detection metrics**. Swiss Federal Institute of Technology (Zurich). Boston University (Boston), 2006. Disponível em: <http://www.cert.org/flocon/2006/presentations/packet_sample_anomoly2006_ppt.pdf>. Acesso em: maio 2008.

CANDIDO JUNIOR, A. et al. Uso de redes neurais para detecção de anomalias em fluxos de dados. In: SIMPÓSIO DE GUERRA ELETRÔNICA, 7., 2005, São José do Rio Preto, SP. **Anais...** São José do Rio Preto, SP, 2005. Disponível em: <http://www.acmesecurity.org/publicacoes/artigos/acme-artigo-sige-2005-arnalmadr.pdf/view>. Acesso em: 17 mar. 2008.

CASWELL, Brian et al. **Snort 2**: sistema de detecção de intruso. Rio de Janeiro: Altas Books, 2003.

DALMAZO, Bruno Lopes et al. Detecção de intrusões baseada em séries temporais. In: SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS, 8., 2008, Santa Maria, RS. **Anais**... Santa Maria, RS: UFSM/INPE, FAPERGS, 2008. Disponível em:

<http://sbseg2008.inf.ufrgs.br/proceedings/data/pdf/st02_01_resumo.pdf>. Acesso em: 17 mar. 2008.

ELKAN, Charles. **Results of the KDD'99 classifier learning contest**. 1999. Disponível em: <http://www-cse.ucsd.edu/~elkan/clresults.html.>. Acesso em: 17 mar. 2008.

ESPINHOSA, Miriam Cristina; GALO, M. L. Bueno Trindade. O uso de redes neurais artificiais na análise ambígua entre classes de água e plantas aquáticas. **Bol. Ciênc. Geod.,** Curitiba, v. 10, n. 2, p. 193-213, jul./dez. 2004. Dsiponível em <http://ojs.c3sl.ufpr.br/ojs2/index.php/bcg/article/view/1533/1287> Acesso em: mar. 2008.

FERREIRA, W. Tavares et al. Uma proposta de utilização da transformada de wavelet e redes neurais para detecção de ataques em redes ad hoc sem fio. In: Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 8., 2008, Mato Grosso. **Anais**... Mato Grosso. 2008. p. 247-248. Disponível em: <http://sbseg2008.inf.ufrgs.br/proceedings/data/pdf/st01_02_resumo.pdf>. Acesso em: 01 set. 2008.

FRANCESQUINI, Emilio de Camargo. **Técnicas de detecção de intrusos para redes móveis sem fio**. São Paulo: USP, 2004. Disponível em: <http://grenoble.ime.usp.br/movel/tecnicasIntruso.pdf>. Acesso em: mar. 2008.

HAIJUN, X. et al. Ad hoc-based features selection and support vector machine classifier for intrusion detection. In proceedings of 2007 IEEE Conference on Neural Networks, vol. 3, 2007.

HAYKIN, Simon. **Redes Neurais**: princípios e prática. 2. ed. Porto Alegre Bookman, 2001.

HEADY, R et al. **The architecture of a network level intrusion detection system**. Technical Report, Computer Science Department, University of New Mexico, Agosto de 1990.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. **Autenticação de assinaturas utilizando algoritmos de aprendizado de máquina**. São Leopoldo, RS: Universidade do Vale do Rio dos Sinos (UNISINOS), 2006. Disponível em: <http://osorio.wait4.org/Signatures/SBIA-SBRN-WCI-Sign-21208.pdf> Acesso em: jun. 2008.

KAYACIK, H. Gunes; ZINCIR-HEYWOOD, Malcolm; HEYWOOD, M. **Selecting features for intrusion detection: a feature relevance analysis on KDD 99 intrusion detection datasets**. Proceedings of the Third Annual Conference on Privacy, Security and Trust, Canada, 2005. Disponível em: <http://projects.cs.dal.ca/projectx>. Acesso em: 17 mar. 2008.

KDD Cup 1999. **UCI KDD repository,** 1999. Disponível em: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>. Acesso em: 10 fev. 2008.

LIMA, Igor Vinícius Mussoi de. **Uma abordagem simplificada de detecção de intrusão baseada em redes neurais artificiais**. 2005. 94 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Santa Catarina, Florianópolis, 2005. Disponível em: <http://www.inf.ufsc.br/~bosco/grupo/MestradoIgor.pdf>. Acesso em: maio 2008.

MAFRA, Paulo M. et al. POLVO-IIDS: um sistema de detecção de intrusão inteligente baseado em anomalias. In: SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS, 8., 2008, Florianópolis. **Anais**... Florianópolis, 2008. Disponível em: <http://sbseg2008.inf.ufrgs.br/proceedings/data/pdf/st02_02_artigo.pdf>. Acesso em: 10 fev. 2008.

MAI, Jianning. et al. **Is sampled data sufficient for anomaly detection?** Califórnia, 2006. Disponível em: <http://www.imconf.net/imc-2006/papers/p17-mai.pdf> Acesso em: abr. 2008.

MONZON, Maikon Weyh. **Aplicação de redes neurais em sistemas de detecção de intrusão**. 2007. 53 f. Trabalho de conclusão de Curso (Graduação - Ciência da

Computação) – Centro Universitário Feevale, Novo Hamburgo, 2007. Disponível em: <http://nead.feevale.br/tc/files/1002.pdf> Acesso em: jun. 2008.

PAULA, Maurício Braga de. **Indução Automática de Árvores de Decisão**. Florianópolis, Universidade Federal de Santa Catarina, novembro de 2002.

PFAHRINGER, Bernhard. **Winning the KDD99 classification cup**: bagged boosting. Viena: Austrian Research Institute, 2000. Disponível em: <http://www.sigkdd.org/explorations/issues/1-2-2000-01/pfahringer.pdf>. Aceso em: abr. 2008.

SANTOS, Fabrício Maravalha. **Detecção de intrusão**: uma abordagem com mineração de dados. 2007. 104 f. Trabalho de conclusão de curso (Graduação – Ciência da Computação) - Universidade de Brasília, Brasília. 2007. Disponível em: < http://monografias.cic.unb.br/dspace/bitstream/123456789/93/1/MONOGRAFIA-santosmaravalha.pdf>. Acesso em: abr. 2008.

SHYU, M. et al. Network Intrusion detection through adaptive sub-eigenspace modeling in multiagent systems. **ACM Transactions on Autonomous and Adaptive Systems**, v. 2, n. 3, September 2007. Article 9.

SILVA, Lília de Sá; MONTES, Antonio; SILVA José Demisio Simões da. Uma solução híbrida para detecção de anomalias em redes. In: WORCAP, 4., 2004, São José dos Campos. **Anais**... São José dos Campos: Instituto Nacional de Pesquisas Espaciais, 2004. Disponível em: <http://hermes2.dpi.inpe.br:1905/col/lac.inpe.br/worcap/2004/10.04.17.04/doc/lilia.pdf > Acesso em: maio 2008.

SILVA, M. et al. **Estudo de Sistemas de Detecção e Prevenção de Intrusões – Uma Abordagem Open Source**. Portugal, Escola Superior de Tecnologia e Gestão de Leiria, 2001.

STEINER, Maria Teresinha Arns et al. **Extração de regras de classificação a partir de redes neurais para auxílio à tomada de decisão na concessão de crédito bancário**. Pesquisa operacional [online]. 2007, v. 27, n. 3, p. 407-426. Disponível

em: <C:\Users\Raquel\Documents\Normatização\Referências Eusan\r28.mht>. Acesso em: abr. 2008.

WANGENHEIM, Aldo Von. **Técnicas subsimbólicas**: redes neurais. Universidade Federal de Santa Catarina. 2008. Disponível em: < C:\Users\Raquel\Documents\Normatização\Referências Eusan\r29.mht >. Acesso em: 17 mar. 2008.

ZARPELÃO, Bruno Bogaz. **Detecção de anomalias e geração de alarmes em redes de computadores**. 2004. 54 f. Trabalho de Conclusão de Curso (Graduação – Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2004. Disponível em: <http://www2.dc.uel.br/nourau/document/?view=90>. Acesso em: abr. 2008.

ZELL, A. et al. **SNNS: Stuttgart Neural Network Simulator**. User Manual, Version 4.1. Stuttgart, University of Stuttgart, 1995.

ZHANG, Z. et al. **Hide: A Hierarchical Network Intrusion Detection System Using Statistical Preprocessing ad Neural Network Classification**, Proceedings of the 2001 IEEE, Workshop on Information Assurance and Security United States Military Academy, West Point, NY. 2001.

APÊNDICE A - SCRIPTS USADOS NO SQL

A seguir serão apresentados os scripts usados no SQL, em cada etapa de processamento implementada, conforme abordado na seção 5.1 do presente trabalho.

```
/*****************************************************************
 * SCRIPT PARA CRIAR AS TABELAS TEMPORARIAS
 *****************************************************************/
USE bd_kdd
GO
-- CRIANDO A TABELA DE FLAG
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[FLAG]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
   DROP TABLE FLAG
GO
CREATE TABLE FLAG(
   id_flag INT NOT NULL,
   desc_flag VARCHAR(30) NOT NULL )
GO
-- CRIANDO A CHAVE PRIMARIA DA TABELA DE FLAG
ALTER TABLE FLAG WITH NOCHECK ADD
CONSTRAINT PK_FLAG PRIMARY KEY CLUSTERED
( id_flag )
GO
-- CRIANDO A TABELA DE CATEGORIA
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[CATEGORIA]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
```

```
   DROP TABLE CATEGORIA
GO
CREATE TABLE CATEGORIA(
id_categoria INT NOT NULL,
desc_label VARCHAR(30) NOT NULL,
desc_categoriaataque VARCHAR(10) NOT NULL
      CODIGO_categoriaataque VARCHAR(10) NOT NULL )
GO
-- CRIANDO A CHAVE PRIMARIA DA TABELA DE CATEGORIA
ALTER TABLE CATEGORIA WITH NOCHECK ADD
CONSTRAINT PK_CATEGORIA PRIMARY KEY CLUSTERED
( id_categoria )
GO
-- CRIANDO A TABELA DE PROTOCOLO
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[PROTOCOLO]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
   DROP TABLE PROTOCOLO
GO
CREATE TABLE PROTOCOLO(
id_protocolo INT NOT NULL,
desc_protocolo VARCHAR(30) NOT NULL )
GO
-- CRIANDO A CHAVE PRIMARIA DA TABELA DE PROTOCOLO
ALTER TABLE PROTOCOLO WITH NOCHECK ADD
CONSTRAINT PK_PROTOCOLO PRIMARY KEY CLUSTERED
( id_protocolo )
GO
```

```
-- CRIANDO A TABELA DE SERVICO
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[SERVICO]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
   DROP TABLE SERVICO
GO
CREATE TABLE SERVICO(
id_servico INT NOT NULL,
desc_servico VARCHAR(30) NOT NULL )
GO
-- CRIANDO A CHAVE PRIMARIA DA TABELA DE SERVICO
ALTER TABLE SERVICO WITH NOCHECK ADD
CONSTRAINT PK_SERVICO PRIMARY KEY CLUSTERED
( id_servico )
GO

/*************************************************************
 * SCRIPT PARA POPULAR AS TABELAS TEMPORARIAS
 *************************************************************/
USE bd_kdd
GO
-- POPULANDO A TABELA DE FLAG
INSERT INTO FLAG
SELECT id_flag = 1, desc_flag = 'REJ' UNION
SELECT id_flag = 2, desc_flag = 'RSTR' UNION
SELECT id_flag = 3, desc_flag = 'S3' UNION
SELECT id_flag = 4, desc_flag = 'S1' UNION
SELECT id_flag = 5, desc_flag = 'OTH' UNION
```

```
SELECT id_flag = 6, desc_flag = 'SF' UNION

SELECT id_flag = 7, desc_flag = 'S2' UNION

SELECT id_flag = 8, desc_flag = 'S0' UNION

SELECT id_flag = 9, desc_flag = 'RSTOS0' UNION

SELECT id_flag = 10, desc_flag = 'SH' UNION

SELECT id_flag = 11, desc_flag = 'RSTO'


-- POPULANDO A TABELA DE CATEGORIA

INSERT INTO CATEGORIA
SELECT id_categoria = 1, desc_label = 'warezmaster.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION

SELECT id_categoria = 2, desc_label = 'guess_passwd.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION

SELECT id_categoria = 3, desc_label = 'multihop.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION

SELECT id_categoria = 4, desc_label = 'satan.', desc_categoriaataque = 'probe', CODIGO_categoriaataque = '0 0 1 0 0' UNION

SELECT id_categoria = 5, desc_label = 'back.', desc_categoriaataque = 'dos', CODIGO_categoriaataque = '0 1 0 0 0' UNION

SELECT id_categoria = 6, desc_label = 'buffer_overflow.', desc_categoriaataque = 'u2r', CODIGO_categoriaataque = '0 0 0 0 1' UNION

SELECT id_categoria = 7, desc_label = 'loadmodule.', desc_categoriaataque = 'u2r', CODIGO_categoriaataque = '0 0 0 0 1' UNION

SELECT id_categoria = 8, desc_label = 'teardrop.', desc_categoriaataque = 'dos', CODIGO_categoriaataque = '0 1 0 0 0' UNION

SELECT id_categoria = 9, desc_label = 'perl.', desc_categoriaataque = 'u2r', CODIGO_categoriaataque = '0 0 0 0 1' UNION

SELECT id_categoria = 10, desc_label = 'rootkit.', desc_categoriaataque = 'u2r', CODIGO_categoriaataque = '0 0 0 0 1' UNION

SELECT id_categoria = 11, desc_label = 'imap.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION
```

```
SELECT id_categoria = 12, desc_label = 'spy.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION

SELECT id_categoria = 13, desc_label = 'land.', desc_categoriaataque = 'dos', CODIGO_categoriaataque = '0 1 0 0 0' UNION

SELECT id_categoria = 14, desc_label = 'ftp_write.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION

SELECT id_categoria = 15, desc_label = 'pod.', desc_categoriaataque = 'dos', CODIGO_categoriaataque = '0 1 0 0 0' UNION

SELECT id_categoria = 16, desc_label = 'smurf.', desc_categoriaataque = 'dos', CODIGO_categoriaataque = '0 1 0 0 0' UNION

SELECT id_categoria = 17, desc_label = 'warezclient.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION

SELECT id_categoria = 18, desc_label = 'neptune.', desc_categoriaataque = 'dos', CODIGO_categoriaataque = '0 1 0 0 0' UNION

SELECT id_categoria = 19, desc_label = 'nmap.', desc_categoriaataque = 'probe', CODIGO_categoriaataque = '0 0 1 0 0' UNION

SELECT id_categoria = 20, desc_label = 'ipsweep.', desc_categoriaataque = 'probe', CODIGO_categoriaataque = '0 0 1 0 0' UNION

SELECT id_categoria = 21, desc_label = 'phf.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION

SELECT id_categoria = 22, desc_label = 'portsweep.', desc_categoriaataque = 'probe', CODIGO_categoriaataque = '0 0 1 0 0' UNION

SELECT id_categoria = 23, desc_label = 'normal.', desc_categoriaataque = 'normal', CODIGO_categoriaataque = '1 0 0 0 0' UNION

SELECT id_categoria = 24, desc_label = 'sqlattack.', desc_categoriaataque = 'u2r', CODIGO_categoriaataque = '0 0 0 0 1'UNION

SELECT id_categoria = 25, desc_label = 'udpstorm.', desc_categoriaataque = 'dos', CODIGO_categoriaataque = '0 1 0 0 0'   UNION

SELECT id_categoria = 26, desc_label = 'worm.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION

SELECT id_categoria = 27, desc_label = 'xsnoop.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION
```

```
SELECT id_categoria = 28, desc_label = 'xlock.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION

SELECT id_categoria = 29, desc_label = 'xterm.', desc_categoriaataque = 'u2r', CODIGO_categoriaataque = '0 0 0 0 1' UNION

SELECT id_categoria = 30, desc_label = 'ps.', desc_categoriaataque = 'u2r', CODIGO_categoriaataque = '0 0 0 0 1' UNION

SELECT id_categoria = 31, desc_label = 'named.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION

SELECT id_categoria = 32, desc_label = 'sendmail.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION

SELECT id_categoria = 33, desc_label = 'httptunnel.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION

SELECT id_categoria = 34, desc_label = 'saint.', desc_categoriaataque = 'probe', CODIGO_categoriaataque = '0 0 1 0 0' UNION

SELECT id_categoria = 35, desc_label = 'processtable.', desc_categoriaataque = 'dos', CODIGO_categoriaataque = '0 1 0 0 0' UNION

SELECT id_categoria = 36, desc_label = 'apache2.', desc_categoriaataque = 'dos', CODIGO_categoriaataque = '0 1 0 0 0' UNION

SELECT id_categoria = 37, desc_label = 'mscan.', desc_categoriaataque = 'probe', CODIGO_categoriaataque = '0 0 1 0 0' UNION

SELECT id_categoria = 38, desc_label = 'snmpguess.', desc_categoriaataque = 'r2l', CODIGO_categoriaataque = '0 0 0 1 0' UNION

SELECT id_categoria = 39, desc_label = 'mailbomb.', desc_categoriaataque = 'dos', CODIGO_categoriaataque = '0 1 0 0 0'

-- POPULANDO A TABELA DE PROTOCOLO

INSERT INTO PROTOCOLO

SELECT id_protocolo = 1, desc_protocolo = 'icmp' UNION

SELECT id_protocolo = 2, desc_protocolo = 'tcp' UNION

SELECT id_protocolo = 3, desc_protocolo = 'udp'

-- POPULANDO A TABELA DE SERVICO

INSERT INTO SERVICO
```

```
SELECT id_servico = 1, desc_servico = 'imap4' UNION
SELECT id_servico = 2, desc_servico = 'auth' UNION
SELECT id_servico = 3, desc_servico = 'http' UNION
SELECT id_servico = 4, desc_servico = 'efs' UNION
SELECT id_servico = 5, desc_servico = 'http_2784' UNION
SELECT id_servico = 6, desc_servico = 'red_i' UNION
SELECT id_servico = 7, desc_servico = 'nnsp' UNION
SELECT id_servico = 8, desc_servico = 'rje' UNION
SELECT id_servico = 9, desc_servico = 'sql_net' UNION
SELECT id_servico = 10, desc_servico = 'supdup' UNION
SELECT id_servico = 11, desc_servico = 'bgp' UNION
SELECT id_servico = 12, desc_servico = 'daytime' UNION
SELECT id_servico = 13, desc_servico = 'netstat' UNION
SELECT id_servico = 14, desc_servico = 'http_443' UNION
SELECT id_servico = 15, desc_servico = 'aol' UNION
SELECT id_servico = 16, desc_servico = 'sunrpc' UNION
SELECT id_servico = 17, desc_servico = 'systat' UNION
SELECT id_servico = 18, desc_servico = 'ldap' UNION
SELECT id_servico = 19, desc_servico = 'kshell' UNION
SELECT id_servico = 20, desc_servico = 'shell' UNION
SELECT id_servico = 21, desc_servico = 'Z39_50' UNION
SELECT id_servico = 22, desc_servico = 'X11' UNION
SELECT id_servico = 23, desc_servico = 'other' UNION
SELECT id_servico = 24, desc_servico = 'echo' UNION
SELECT id_servico = 25, desc_servico = 'mtp' UNION
SELECT id_servico = 26, desc_servico = 'csnet_ns' UNION
SELECT id_servico = 27, desc_servico = 'discard' UNION
```

```
SELECT id_servico = 28, desc_servico = 'urh_i' UNION
SELECT id_servico = 29, desc_servico = 'eco_i' UNION
SELECT id_servico = 30, desc_servico = 'netbios_ssn' UNION
SELECT id_servico = 31, desc_servico = 'uucp' UNION
SELECT id_servico = 32, desc_servico = 'courier' UNION
SELECT id_servico = 33, desc_servico = 'pm_dump' UNION
SELECT id_servico = 34, desc_servico = 'telnet' UNION
SELECT id_servico = 35, desc_servico = 'link' UNION
SELECT id_servico = 36, desc_servico = 'tim_i' UNION
SELECT id_servico = 37, desc_servico = 'pop_3' UNION
SELECT id_servico = 38, desc_servico = 'finger' UNION
SELECT id_servico = 39, desc_servico = 'pop_2' UNION
SELECT id_servico = 40, desc_servico = 'ntp_u' UNION
SELECT id_servico = 41, desc_servico = 'vmnet' UNION
SELECT id_servico = 42, desc_servico = 'netbios_ns' UNION
SELECT id_servico = 43, desc_servico = 'domain_u' UNION
SELECT id_servico = 44, desc_servico = 'uucp_path' UNION
SELECT id_servico = 45, desc_servico = 'nntp' UNION
SELECT id_servico = 46, desc_servico = 'http_8001' UNION
SELECT id_servico = 47, desc_servico = 'domain' UNION
SELECT id_servico = 48, desc_servico = 'name' UNION
SELECT id_servico = 49, desc_servico = 'ftp_data' UNION
SELECT id_servico = 50, desc_servico = 'hostnames' UNION
SELECT id_servico = 51, desc_servico = 'printer' UNION
SELECT id_servico = 52, desc_servico = 'urp_i' UNION
SELECT id_servico = 53, desc_servico = 'private' UNION
SELECT id_servico = 54, desc_servico = 'gopher' UNION
```

```
        SELECT id_servico = 55, desc_servico = 'time' UNION
        SELECT id_servico = 56, desc_servico = 'ctf' UNION
        SELECT id_servico = 57, desc_servico = 'iso_tsap' UNION
        SELECT id_servico = 58, desc_servico = 'smtp' UNION
        SELECT id_servico = 59, desc_servico = 'IRC' UNION
        SELECT id_servico = 60, desc_servico = 'harvest' UNION
        SELECT id_servico = 61, desc_servico = 'tftp_u' UNION
        SELECT id_servico = 62, desc_servico = 'netbios_dgm' UNION
        SELECT id_servico = 63, desc_servico = 'ssh' UNION
        SELECT id_servico = 64, desc_servico = 'exec' UNION
        SELECT id_servico = 65, desc_servico = 'whois' UNION
        SELECT id_servico = 66, desc_servico = 'ftp' UNION
        SELECT id_servico = 67, desc_servico = 'ecr_i' UNION
        SELECT id_servico = 68, desc_servico = 'remote_job' UNION
        SELECT id_servico = 69, desc_servico = 'klogin' UNION
        SELECT id_servico = 70, desc_servico = 'login' UNION
        SELECT id_servico = 71, desc_servico = 'icmp'


/*****************************************************************
 * SCRIPT PARA EXCLUIR E CRIAR TABELA KDDCUP
 *****************************************************************/
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[KDDCUP]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)
   DROP TABLE KDDCUP
GO
CREATE TABLE KDDCUP (
   id_kddcup INT NOT NULL  IDENTITY(1,1),
```

Duration FLOAT(15) NOT NULL,

Protocol_type CHAR(30) NOT NULL,

Service CHAR(30) NOT NULL,

Src_bytes CHAR(30) NOT NULL,

Dst_bytes FLOAT(1) NOT NULL,

Flag FLOAT(15) NOT NULL,

Land FLOAT(15) NOT NULL,

Wrong_fragment FLOAT(15) NOT NULL,

Urgent FLOAT(15) NOT NULL,

Hot FLOAT(15) NOT NULL,

Num_failed_logins FLOAT(15) NOT NULL,

Logged_in FLOAT(15) NOT NULL,

Num_compromised FLOAT(15) NOT NULL,

Root_shell FLOAT(15) NOT NULL,

Su_attempted FLOAT(15) NOT NULL,

Num_root FLOAT(15) NOT NULL,

Num_file_creations FLOAT(15) NOT NULL,

Num_shells FLOAT(15) NOT NULL,

Num_access_files FLOAT(15) NOT NULL,

Num_outbound_cmds FLOAT(15) NOT NULL,

Is_hot_login FLOAT(15) NOT NULL,

Is_guest_login FLOAT(15) NOT NULL,

Kount FLOAT(15) NOT NULL,

Srv_count FLOAT(15) NOT NULL,

Serror_rate FLOAT(15) NOT NULL,

Srv_serror_rate FLOAT(15) NOT NULL,

Rerror_rate FLOAT(15) NOT NULL,

```
   Srv_rerror_rate FLOAT(15) NOT NULL,
   Same_srv_rate FLOAT(15) NOT NULL,
   Diff_srv_rate FLOAT(15) NOT NULL,
   Srv_diff_host_rate FLOAT(15) NOT NULL,
   Dst_host_count FLOAT(15) NOT NULL,
   Dst_host_srv_count FLOAT(15) NOT NULL,
   Dst_host_same_srv_count FLOAT(15) NOT NULL,
   Dst_host_diff_srv_rate FLOAT(15) NOT NULL,
   Dst_host_same_src_port_rate FLOAT(15) NOT NULL,
   Dst_host_srv_diff_host_rate FLOAT(15) NOT NULL,
   Dst_host_serror_rate FLOAT(15) NOT NULL,
   Dst_host_srv_serror_rate FLOAT(15) NOT NULL,
   Dst_host_rerror_rate FLOAT(15) NOT NULL,
   Dst_host_srv_rerror_rate FLOAT(15) NOT NULL,
   COD_Cat CHAR(15)  NULL,
   Categoria CHAR(15) NULL,
   flag_processado BIT NOT NULL DEFAULT 0 )
/*****************************************************************
 * SCRIPT PARA SUBSTITUIÇÃO DOS CAMPOS
 *****************************************************************/
UPDATE KDDCUP SET Protocol_type = (SELECT id_protocolo FROM
PROTOCOLO WHERE desc_protocolo = rtrim(ltrim(Protocol_type)) )
FROM KDDCUP K

UPDATE KDDCUP SET Service = (SELECT id_servico FROM SERVICO WHERE
desc_servico = rtrim(ltrim(Service)) )
FROM KDDCUP K
```

```
UPDATE KDDCUP SET COD_Cat =  (SELECT CODIGO_CategoriaATAQUE FROM CATEGORIA WHERE desc_label = rtrim(ltrim(Categoria)) )

FROM KDDCUP K

UPDATE KDDCUP SET Categoria =  (SELECT id_categoria FROM CATEGORIA WHERE desc_label = rtrim(ltrim(Categoria)) )

FROM KDDCUP K

UPDATE KDDCUP SET Src_bytes = (SELECT id_flag FROM FLAG WHERE desc_flag = rtrim(ltrim(Src_bytes)) )

FROM KDDCUP K


/*****************************************************************

 * SCRIPT PARA NORMALIZAÇÃO DOS VALORES

 *****************************************************************/

DECLARE @MIN NUMERIC(10,5),

        @MAX NUMERIC(10,5)

-- NORMALIZANDO O CAMPO DURATION

SET @MIN = (SELECT MIN (Duration) FROM KDDCUP)

SET @MAX = (SELECT MAX (Duration) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Duration = ROUND((2*(Duration)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Duration = 0

-- NORMALIZANDO O CAMPO PROTOCOLO

SET @MIN = (SELECT MIN(CAST(LTRIM(RTRIM(Protocol_type)) AS INT)) FROM KDDCUP)

SET @MAX = (SELECT MAX(CAST(LTRIM(RTRIM(Protocol_type)) AS INT)) FROM KDDCUP)

IF (@MAX-@MIN) <> 0
```

```
UPDATE KDDCUP SET Protocol_type = ROUND((2*(Protocol_type)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Protocol_type = 0

-- NORMALIZANDO O CAMPO SERVICE

SET @MIN = (SELECT MIN(CAST(LTRIM(RTRIM(Service)) AS INT)) FROM KDDCUP)

SET @MAX = (SELECT MAX(CAST(LTRIM(RTRIM(Service)) AS INT)) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Service = ROUND((2*(Service)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Service = 0


-- NORMALIZANDO O CAMPO SRC_BYTES

SET @MIN = (SELECT MIN(CAST(LTRIM(RTRIM(Src_bytes)) AS INT)) FROM KDDCUP)

SET @MAX = (SELECT MAX(CAST(LTRIM(RTRIM(Src_bytes)) AS INT)) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Src_bytes = ROUND((2*(Src_bytes)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Src_bytes = 0

-- NORMALIZANDO O CAMPO Dst_bytes

SET @MIN = (SELECT MIN (Dst_bytes) FROM KDDCUP)

SET @MAX = (SELECT MAX (Dst_bytes) FROM KDDCUP)

IF (@MAX-@MIN) <> 0
```

```
UPDATE KDDCUP SET Dst_bytes = ROUND((2*(Dst_bytes)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Dst_bytes = 0

-- NORMALIZANDO O CAMPO Flag

SET @MIN = (SELECT MIN (Flag) FROM KDDCUP)

SET @MAX = (SELECT MAX (Flag) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Flag = ROUND((2*(Flag)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Flag = 0

-- NORMALIZANDO O CAMPO Land

SET @MIN = (SELECT MIN (Land) FROM KDDCUP)

SET @MAX = (SELECT MAX (Land) FROM KDDCUP)


IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Land = ROUND((2*(Land)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Land = 0

-- NORMALIZANDO O CAMPO Wrong_fragment

SET @MIN = (SELECT MIN (Wrong_fragment) FROM KDDCUP)

SET @MAX = (SELECT MAX (Wrong_fragment) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Wrong_fragment = ROUND((2*(Wrong_fragment)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Wrong_fragment = 0
```

```
-- NORMALIZANDO O CAMPO Urgent

SET @MIN = (SELECT MIN (Urgent) FROM KDDCUP)

SET @MAX = (SELECT MAX (Urgent) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Urgent = ROUND((2*(Urgent)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Urgent = 0

-- NORMALIZANDO O CAMPO Hot

SET @MIN = (SELECT MIN (Hot) FROM KDDCUP)

SET @MAX = (SELECT MAX (Hot) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Hot = ROUND((2*(Hot)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Hot = 0


-- NORMALIZANDO O CAMPO Num_failed_logins

SET @MIN = (SELECT MIN (Num_failed_logins) FROM KDDCUP)

SET @MAX = (SELECT MAX (Num_failed_logins) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Num_failed_logins = ROUND((2*(Num_failed_logins)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Num_failed_logins = 0

-- NORMALIZANDO O CAMPO Logged_in

SET @MIN = (SELECT MIN (Logged_in) FROM KDDCUP)

SET @MAX = (SELECT MAX (Logged_in) FROM KDDCUP)

IF (@MAX-@MIN) <> 0
```

```
UPDATE KDDCUP SET Logged_in = ROUND((2*(Logged_in)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Logged_in = 0

-- NORMALIZANDO O CAMPO Num_compromised

SET @MIN = (SELECT MIN (Num_compromised) FROM KDDCUP)

SET @MAX = (SELECT MAX (Num_compromised) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Num_compromised = ROUND((2*(Num_compromised)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Num_compromised = 0

-- NORMALIZANDO O CAMPO Root_shell

SET @MIN = (SELECT MIN (Root_shell) FROM KDDCUP)

SET @MAX = (SELECT MAX (Root_shell) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Root_shell = ROUND((2*(Root_shell)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Root_shell = 0

-- NORMALIZANDO O CAMPO Su_attempted

SET @MIN = (SELECT MIN (Su_attempted) FROM KDDCUP)

SET @MAX = (SELECT MAX (Su_attempted) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Su_attempted = ROUND((2*(Su_attempted)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Su_attempted = 0
```

```
-- NORMALIZANDO O CAMPO Num_root

SET @MIN = (SELECT MIN (Num_root) FROM KDDCUP)

SET @MAX = (SELECT MAX (Num_root) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Num_root = ROUND((2*(Num_root)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Num_root = 0

-- NORMALIZANDO O CAMPO Num_file_creations

SET @MIN = (SELECT MIN (Num_file_creations) FROM KDDCUP)

SET @MAX = (SELECT MAX (Num_file_creations) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Num_file_creations = ROUND((2*(Num_file_creations)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Num_file_creations = 0


-- NORMALIZANDO O CAMPO Num_shells

SET @MIN = (SELECT MIN (Num_shells) FROM KDDCUP)

SET @MAX = (SELECT MAX (Num_shells) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Num_shells = ROUND((2*(Num_shells)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Num_shells = 0

-- NORMALIZANDO O CAMPO Num_access_files

SET @MIN = (SELECT MIN (Num_access_files) FROM KDDCUP)

SET @MAX = (SELECT MAX (Num_access_files) FROM KDDCUP)
```

```
IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Num_access_files = ROUND((2*(Num_access_files)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Num_access_files = 0

-- NORMALIZANDO O CAMPO Num_outbound_cmds

SET @MIN = (SELECT MIN (Num_outbound_cmds) FROM KDDCUP)

SET @MAX = (SELECT MAX (Num_outbound_cmds) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Num_outbound_cmds = (2*(Num_outbound_cmds)-@MIN-@MAX)/(@MAX-@MIN)

ELSE

UPDATE KDDCUP SET Num_outbound_cmds = 0

-- NORMALIZANDO O CAMPO Is_hot_login

SET @MIN = (SELECT MIN (Is_hot_login) FROM KDDCUP)

SET @MAX = (SELECT MAX (Is_hot_login) FROM KDDCUP)


IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Is_hot_login = ROUND((2*(Is_hot_login)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Is_hot_login = 0

-- NORMALIZANDO O CAMPO Is_guest_login

SET @MIN = (SELECT MIN (Is_guest_login) FROM KDDCUP)

SET @MAX = (SELECT MAX (Is_guest_login) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Is_guest_login = ROUND((2*(Is_guest_login)-@MIN-@MAX)/(@MAX-@MIN),3)
```

```
ELSE

UPDATE KDDCUP SET Is_guest_login = 0

-- NORMALIZANDO O CAMPO Count

SET @MIN = (SELECT MIN (kount) FROM KDDCUP)

SET @MAX = (SELECT MAX (kount) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET kount = ROUND((2*(kount)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET kount = 0

-- NORMALIZANDO O CAMPO Srv_count

SET @MIN = (SELECT MIN (Srv_count) FROM KDDCUP)

SET @MAX = (SELECT MAX (Srv_count) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Srv_count = ROUND((2*(Srv_count)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Srv_count = 0

-- NORMALIZANDO O CAMPO Serror_rate

SET @MIN = (SELECT MIN (Serror_rate) FROM KDDCUP)

SET @MAX = (SELECT MAX (Serror_rate) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Serror_rate = ROUND((2*(Serror_rate)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Serror_rate = 0

-- NORMALIZANDO O CAMPO Srv_serror_rate

SET @MIN = (SELECT MIN (Srv_serror_rate) FROM KDDCUP)
```

```
SET @MAX = (SELECT MAX (Srv_serror_rate) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Srv_serror_rate = ROUND((2*(Srv_serror_rate)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Srv_serror_rate = 0

-- NORMALIZANDO O CAMPO Rerror_rate

SET @MIN = (SELECT MIN (Rerror_rate) FROM KDDCUP)

SET @MAX = (SELECT MAX (Rerror_rate) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Rerror_rate = ROUND((2*(Rerror_rate)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Rerror_rate = 0

-- NORMALIZANDO O CAMPO Srv_rerror_rate

SET @MIN = (SELECT MIN (Srv_rerror_rate) FROM KDDCUP)

SET @MAX = (SELECT MAX (Srv_rerror_rate) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Srv_rerror_rate = ROUND((2*(Srv_rerror_rate)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Srv_rerror_rate = 0

-- NORMALIZANDO O CAMPO Same_srv_rate

SET @MIN = (SELECT MIN (Same_srv_rate) FROM KDDCUP)

SET @MAX = (SELECT MAX (Same_srv_rate) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Same_srv_rate = ROUND((2*(Same_srv_rate)-@MIN-@MAX)/(@MAX-@MIN),3)
```

```
ELSE

UPDATE KDDCUP SET Same_srv_rate = 0

-- NORMALIZANDO O CAMPO Diff_srv_rate

SET @MIN = (SELECT MIN (Diff_srv_rate) FROM KDDCUP)

SET @MAX = (SELECT MAX (Diff_srv_rate) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Diff_srv_rate = ROUND((2*(Diff_srv_rate)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Diff_srv_rate = 0

-- NORMALIZANDO O CAMPO Srv_diff_host_rate

SET @MIN = (SELECT MIN (Srv_diff_host_rate) FROM KDDCUP)

SET @MAX = (SELECT MAX (Srv_diff_host_rate) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Srv_diff_host_rate = ROUND((2*(Srv_diff_host_rate)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Srv_diff_host_rate = 0

-- NORMALIZANDO O CAMPO Dst_host_count

SET @MIN = (SELECT MIN (Dst_host_count) FROM KDDCUP)

SET @MAX = (SELECT MAX (Dst_host_count) FROM KDDCUP)


IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Dst_host_count = ROUND((2*(Dst_host_count)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Dst_host_count = 0

-- NORMALIZANDO O CAMPO Dst_host_srv_count
```

```
SET @MIN = (SELECT MIN (Dst_host_srv_count) FROM KDDCUP)

SET @MAX = (SELECT MAX (Dst_host_srv_count) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Dst_host_srv_count = ROUND((2*(Dst_host_srv_count)-
@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Dst_host_srv_count = 0

-- NORMALIZANDO O CAMPO Dst_host_same_srv_count

SET @MIN = (SELECT MIN (Dst_host_same_srv_count) FROM KDDCUP)

SET @MAX = (SELECT MAX (Dst_host_same_srv_count) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Dst_host_same_srv_count =
ROUND((2*(Dst_host_same_srv_count)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Dst_host_same_srv_count = 0

-- NORMALIZANDO O CAMPO Dst_host_diff_srv_rate

SET @MIN = (SELECT MIN (Dst_host_diff_srv_rate) FROM KDDCUP)

SET @MAX = (SELECT MAX (Dst_host_diff_srv_rate) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Dst_host_diff_srv_rate =
ROUND((2*(Dst_host_diff_srv_rate)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Dst_host_diff_srv_rate = 0


-- NORMALIZANDO O CAMPO Dst_host_same_src_port_rate

SET @MIN = (SELECT MIN (Dst_host_same_src_port_rate) FROM KDDCUP)

SET @MAX = (SELECT MAX (Dst_host_same_src_port_rate) FROM KDDCUP)

IF (@MAX-@MIN) <> 0
```

```
UPDATE KDDCUP SET Dst_host_same_src_port_rate = ROUND((2*(Dst_host_same_src_port_rate)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Dst_host_same_src_port_rate = 0

-- NORMALIZANDO O CAMPO Dst_host_srv_diff_host_rate

SET @MIN = (SELECT MIN (Dst_host_srv_diff_host_rate) FROM KDDCUP)

SET @MAX = (SELECT MAX (Dst_host_srv_diff_host_rate) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Dst_host_srv_diff_host_rate = ROUND((2*(Dst_host_srv_diff_host_rate)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Dst_host_srv_diff_host_rate = 0

-- NORMALIZANDO O CAMPO Dst_host_serror_rate

SET @MIN = (SELECT MIN (Dst_host_serror_rate) FROM KDDCUP)

SET @MAX = (SELECT MAX (Dst_host_serror_rate) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Dst_host_serror_rate = ROUND((2*(Dst_host_serror_rate)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Dst_host_serror_rate = 0

-- NORMALIZANDO O CAMPO Dst_host_srv_serror_rate

SET @MIN = (SELECT MIN (Dst_host_srv_serror_rate) FROM KDDCUP)

SET @MAX = (SELECT MAX (Dst_host_srv_serror_rate) FROM KDDCUP)


IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Dst_host_srv_serror_rate = ROUND((2*(Dst_host_srv_serror_rate)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE
```

```
UPDATE KDDCUP SET Dst_host_srv_serror_rate = 0

-- NORMALIZANDO O CAMPO Dst_host_rerror_rate

SET @MIN = (SELECT MIN (Dst_host_rerror_rate) FROM KDDCUP)

SET @MAX = (SELECT MAX (Dst_host_rerror_rate) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Dst_host_rerror_rate = ROUND((2*(Dst_host_rerror_rate)-
@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Dst_host_rerror_rate = 0

-- NORMALIZANDO O CAMPO Dst_host_srv_rerror_rate

SET @MIN = (SELECT MIN (Dst_host_srv_rerror_rate) FROM KDDCUP)

SET @MAX = (SELECT MAX (Dst_host_srv_rerror_rate) FROM KDDCUP)

IF (@MAX-@MIN) <> 0

UPDATE KDDCUP SET Dst_host_srv_rerror_rate =
ROUND((2*(Dst_host_srv_rerror_rate)-@MIN-@MAX)/(@MAX-@MIN),3)

ELSE

UPDATE KDDCUP SET Dst_host_srv_rerror_rate = 0


/*************************************************************

 * SCRIPT PARA PREPARAÇÃO DAS TABELAS DE SAÍDA - SC

 *************************************************************/

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[SC1]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)

   DROP TABLE SC1

GO

CREATE TABLE SC1 (

   id_sc1 INT NOT NULL IDENTITY(1,1),

   TEXTO CHAR(30) NOT NULL,
```

Duration FLOAT(3) NOT NULL,

Protocol_type FLOAT(3) NOT NULL,

Service FLOAT(3) NOT NULL,

Src_bytes FLOAT(3) NOT NULL,

Dst_bytes FLOAT(3) NOT NULL,

Flag FLOAT(3) NOT NULL,

Land FLOAT(3) NOT NULL,

Wrong_fragment FLOAT(3) NOT NULL,

Urgent FLOAT(3) NOT NULL,

Hot FLOAT(3) NOT NULL,

Num_failed_logins FLOAT(3) NOT NULL,

Logged_in FLOAT(3) NOT NULL,

Num_compromised FLOAT(3) NOT NULL,

Root_shell FLOAT(3) NOT NULL,

Su_attempted FLOAT(3) NOT NULL,

Num_root FLOAT(3) NOT NULL,

Num_file_creations FLOAT(3) NOT NULL,

Num_shells FLOAT(3) NOT NULL,

Num_access_files FLOAT(3) NOT NULL,

Num_outbound_cmds FLOAT(3) NOT NULL,

Is_hot_login FLOAT(3) NOT NULL,

Is_guest_login FLOAT(3) NOT NULL,

count FLOAT(3) NOT NULL,

Srv_count FLOAT(3) NOT NULL,

Serror_rate FLOAT(3) NOT NULL,

Srv_serror_rate FLOAT(3) NOT NULL,

Rerror_rate FLOAT(3) NOT NULL,

```
   Srv_rerror_rate FLOAT(3) NOT NULL,
   Same_srv_rate FLOAT(3) NOT NULL,
   Diff_srv_rate FLOAT(3) NOT NULL,
   Srv_diff_host_rate FLOAT(3) NOT NULL,
   Dst_host_count FLOAT(3) NOT NULL,
   Dst_host_srv_count FLOAT(3) NOT NULL,
   Dst_host_same_srv_count FLOAT(3) NOT NULL,
   Dst_host_diff_srv_rate FLOAT(3) NOT NULL,
   Dst_host_same_src_port_rate FLOAT(3) NOT NULL,
   Dst_host_srv_diff_host_rate FLOAT(3) NOT NULL,
   Dst_host_serror_rate FLOAT(3) NOT NULL,
   Dst_host_srv_serror_rate FLOAT(3) NOT NULL,
   Dst_host_rerror_rate FLOAT(3) NOT NULL,
   Dst_host_srv_rerror_rate FLOAT(3) NOT NULL,
   TEXTO2 CHAR(30) NULL,
   COD_CAT CHAR(15) NOT NULL,
   Categoria int NULL )
GO
CREATE UNIQUE CLUSTERED INDEX [IX_SC1] ON [dbo].[SC1]([id_sc1]) ON [PRIMARY]
-- CRIANDO AS TABELAS DE SAIDA - SC2
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[SC2]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
   DROP TABLE SC2
GO
CREATE TABLE SC2 (
   id_sc2 INT NOT NULL IDENTITY(1,1),
   TEXTO CHAR(30) NOT NULL,
```

```
Duration INT NOT NULL,
Protocol_type CHAR(30) NOT NULL,
Service CHAR(30) NOT NULL,
Src_bytes CHAR(30) NOT NULL,
Dst_bytes INT NOT NULL,
Flag INT NOT NULL,
Land INT NOT NULL,
Wrong_fragment INT NOT NULL,
Urgent INT NOT NULL,
Hot INT NOT NULL,
Num_failed_logins INT NOT NULL,
Logged_in INT NOT NULL,
Num_compromised INT NOT NULL,
Root_shell INT NOT NULL,
Su_attempted INT NOT NULL,
Num_root INT NOT NULL,
Num_file_creations INT NOT NULL,
Num_shells INT NOT NULL,
Num_access_files INT NOT NULL,
Num_outbound_cmds INT NOT NULL,
Is_hot_login INT NOT NULL,
Is_guest_login INT NOT NULL,
Count INT NOT NULL,
Srv_count INT NOT NULL,
Serror_rate NUMERIC(5,2) NOT NULL,
Srv_serror_rate NUMERIC(5,2) NOT NULL,
Rerror_rate NUMERIC(5,2) NOT NULL,
```

```
    Srv_rerror_rate NUMERIC(5,2) NOT NULL,
    Same_srv_rate NUMERIC(5,2) NOT NULL,
    Diff_srv_rate NUMERIC(5,2) NOT NULL,
    Srv_diff_host_rate NUMERIC(5,2) NOT NULL,
    Dst_host_count NUMERIC(5,2) NOT NULL,
    Dst_host_srv_count INT NOT NULL,
    Dst_host_same_srv_count NUMERIC(5,2) NOT NULL,
    Dst_host_diff_srv_rate NUMERIC(5,2) NOT NULL,
    Dst_host_same_src_port_rate NUMERIC(5,2) NOT NULL,
    Dst_host_srv_diff_host_rate NUMERIC(5,2) NOT NULL,
    Dst_host_serror_rate NUMERIC(5,2) NOT NULL,
    Dst_host_srv_serror_rate NUMERIC(5,2) NOT NULL,
    Dst_host_rerror_rate NUMERIC(5,2) NOT NULL,
    Dst_host_srv_rerror_rate NUMERIC(5,2) NOT NULL,
    TEXTO2 CHAR(30) NULL,
    COD_CAT CHAR(15) NOT NULL,
    Categoria int NULL )
GO
CREATE UNIQUE CLUSTERED INDEX [IX_SC2] ON [dbo].[SC2]([id_sc2]) ON [PRIMARY]
-- CRIANDO AS TABELAS DE SAIDA - SC3
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[SC3]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
    DROP TABLE SC3
GO
CREATE TABLE SC3 (
    id_sc3 INT NOT NULL IDENTITY(1,1),
    TEXTO CHAR(30) NOT NULL,
```

```
Duration INT NOT NULL,
Protocol_type CHAR(30) NOT NULL,
Service CHAR(30) NOT NULL,
Src_bytes CHAR(30) NOT NULL,
Dst_bytes INT NOT NULL,
Flag INT NOT NULL,
Land INT NOT NULL,
Wrong_fragment INT NOT NULL,
Urgent INT NOT NULL,
Hot INT NOT NULL,
Num_failed_logins INT NOT NULL,
Logged_in INT NOT NULL,
Num_compromised INT NOT NULL,
Root_shell INT NOT NULL,
Su_attempted INT NOT NULL,
Num_root INT NOT NULL,
Num_file_creations INT NOT NULL,
Num_shells INT NOT NULL,
Num_access_files INT NOT NULL,
Num_outbound_cmds INT NOT NULL,
Is_hot_login INT NOT NULL,
Is_guest_login INT NOT NULL,
Count INT NOT NULL,
Srv_count INT NOT NULL,
Serror_rate NUMERIC(5,2) NOT NULL,
Srv_serror_rate NUMERIC(5,2) NOT NULL,
Rerror_rate NUMERIC(5,2) NOT NULL,
```

```
   Srv_rerror_rate NUMERIC(5,2) NOT NULL,
   Same_srv_rate NUMERIC(5,2) NOT NULL,
   Diff_srv_rate NUMERIC(5,2) NOT NULL,
   Srv_diff_host_rate NUMERIC(5,2) NOT NULL,
   Dst_host_count NUMERIC(5,2) NOT NULL,
   Dst_host_srv_count INT NOT NULL,
   Dst_host_same_srv_count NUMERIC(5,2) NOT NULL,
   Dst_host_diff_srv_rate NUMERIC(5,2) NOT NULL,
   Dst_host_same_src_port_rate NUMERIC(5,2) NOT NULL,
   Dst_host_srv_diff_host_rate NUMERIC(5,2) NOT NULL,
   Dst_host_serror_rate NUMERIC(5,2) NOT NULL,
   Dst_host_srv_serror_rate NUMERIC(5,2) NOT NULL,
   Dst_host_rerror_rate NUMERIC(5,2) NOT NULL,
   Dst_host_srv_rerror_rate NUMERIC(5,2) NOT NULL,
   TEXTO2 CHAR(30) NULL,
   COD_CAT CHAR(15) NOT NULL,
   Categoria int NULL )
GO
CREATE UNIQUE CLUSTERED INDEX [IX_SC3] ON [dbo].[SC3]([id_sc3]) ON [PRIMARY]
-- CRIANDO AS TABELAS DE SAIDA - SC3
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[SC4]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
   DROP TABLE SC4
GO
CREATE TABLE SC4 (
   id_sc4 INT NOT NULL IDENTITY(1,1),
   TEXTO CHAR(30) NOT NULL,
```

```
Duration INT NOT NULL,
Protocol_type CHAR(30) NOT NULL,
Service CHAR(30) NOT NULL,
Src_bytes CHAR(30) NOT NULL,
Dst_bytes INT NOT NULL,
Flag INT NOT NULL,
Land INT NOT NULL,
Wrong_fragment INT NOT NULL,
Urgent INT NOT NULL,
Hot INT NOT NULL,
Num_failed_logins INT NOT NULL,
Logged_in INT NOT NULL,
Num_compromised INT NOT NULL,
Root_shell INT NOT NULL,
Su_attempted INT NOT NULL,
Num_root INT NOT NULL,
Num_file_creations INT NOT NULL,
Num_shells INT NOT NULL,
Num_access_files INT NOT NULL,
Num_outbound_cmds INT NOT NULL,
Is_hot_login INT NOT NULL,
Is_guest_login INT NOT NULL,
Count INT NOT NULL,
Srv_count INT NOT NULL,
Serror_rate NUMERIC(5,2) NOT NULL,
Srv_serror_rate NUMERIC(5,2) NOT NULL,
Rerror_rate NUMERIC(5,2) NOT NULL,
```

```
   Srv_rerror_rate NUMERIC(5,2) NOT NULL,
   Same_srv_rate NUMERIC(5,2) NOT NULL,
   Diff_srv_rate NUMERIC(5,2) NOT NULL,
   Srv_diff_host_rate NUMERIC(5,2) NOT NULL,
   Dst_host_count NUMERIC(5,2) NOT NULL,
   Dst_host_srv_count INT NOT NULL,
   Dst_host_same_srv_count NUMERIC(5,2) NOT NULL,
   Dst_host_diff_srv_rate NUMERIC(5,2) NOT NULL,
   Dst_host_same_src_port_rate NUMERIC(5,2) NOT NULL,
   Dst_host_srv_diff_host_rate NUMERIC(5,2) NOT NULL,
   Dst_host_serror_rate NUMERIC(5,2) NOT NULL,
   Dst_host_srv_serror_rate NUMERIC(5,2) NOT NULL,
   Dst_host_rerror_rate NUMERIC(5,2) NOT NULL,
   Dst_host_srv_rerror_rate NUMERIC(5,2) NOT NULL,
   TEXTO2 CHAR(30) NULL,
   COD_CAT CHAR(15) NOT NULL,
   Categoria int NULL )
GO
CREATE UNIQUE CLUSTERED INDEX [IX_SC5] ON [dbo].[SC4]([id_sc4]) ON [PRIMARY]
-- CRIANDO AS TABELAS DE SAIDA - SC5
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[SC5]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
   DROP TABLE SC5
GO
CREATE TABLE SC5 (
   id_sc5 INT NOT NULL IDENTITY(1,1),
   TEXTO CHAR(30) NOT NULL,
```

```
Duration INT NOT NULL,
Protocol_type CHAR(30) NOT NULL,
Service CHAR(30) NOT NULL,
Src_bytes CHAR(30) NOT NULL,
Dst_bytes INT NOT NULL,
Flag INT NOT NULL,
Land INT NOT NULL,
Wrong_fragment INT NOT NULL,
Urgent INT NOT NULL,
Hot INT NOT NULL,
Num_failed_logins INT NOT NULL,
Logged_in INT NOT NULL,
Num_compromised INT NOT NULL,
Root_shell INT NOT NULL,
Su_attempted INT NOT NULL,
Num_root INT NOT NULL,
Num_file_creations INT NOT NULL,
Num_shells INT NOT NULL,
Num_access_files INT NOT NULL,
Num_outbound_cmds INT NOT NULL,
Is_hot_login INT NOT NULL,
Is_guest_login INT NOT NULL,
Count INT NOT NULL,
Srv_count INT NOT NULL,
Serror_rate NUMERIC(5,2) NOT NULL,
Srv_serror_rate NUMERIC(5,2) NOT NULL,
Rerror_rate NUMERIC(5,2) NOT NULL,
```

```
   Srv_rerror_rate NUMERIC(5,2) NOT NULL,
   Same_srv_rate NUMERIC(5,2) NOT NULL,
   Diff_srv_rate NUMERIC(5,2) NOT NULL,
   Srv_diff_host_rate NUMERIC(5,2) NOT NULL,
   Dst_host_count NUMERIC(5,2) NOT NULL,
   Dst_host_srv_count INT NOT NULL,
   Dst_host_same_srv_count NUMERIC(5,2) NOT NULL,
   Dst_host_diff_srv_rate NUMERIC(5,2) NOT NULL,
   Dst_host_same_src_port_rate NUMERIC(5,2) NOT NULL,
   Dst_host_srv_diff_host_rate NUMERIC(5,2) NOT NULL,
   Dst_host_serror_rate NUMERIC(5,2) NOT NULL,
   Dst_host_srv_serror_rate NUMERIC(5,2) NOT NULL,
   Dst_host_rerror_rate NUMERIC(5,2) NOT NULL,
   Dst_host_srv_rerror_rate NUMERIC(5,2) NOT NULL,
   TEXTO2 CHAR(30) NULL,
   COD_CAT CHAR(15) NOT NULL,
   Categoria int NULL )
GO
CREATE UNIQUE CLUSTERED INDEX [IX_SC5] ON [dbo].[SC5]([id_sc5]) ON
[PRIMARY]
/*****************************************************************
 * SCRIPT PARA POPULAR AS TABELAS DE SAÍDA - SC
 *****************************************************************/

-- POPULANDO AS TABELAS DE SAIDA
DECLARE @QTD_REGISTROS_SAIDA INT,
      @INTERACOES INT,
```

```
    @PERCENTCONCLUSAO NUMERIC(18,10),
    @A INT,
    @B INT,
    @C INT,
    @D INT,
    @E INT
SET @QTD_REGISTROS_SAIDA = 10000 -- DEFININDO A QUANTIDADE DE REGISTROS DE CADA TABELA
SET @INTERACOES = 0
SET @PERCENTCONCLUSAO = 0
SET @A = 1
SET @B = 1
SET @C = 1
SET @D = 1
SET @E = 1
-- EXCLUINDO OS VALORES DAS TABELAS DE SAIDA
TRUNCATE TABLE SC1
TRUNCATE TABLE SC2
TRUNCATE TABLE SC3
TRUNCATE TABLE SC4
TRUNCATE TABLE SC5
WHILE (((SELECT COUNT(ID_SC1) FROM SC1) < @QTD_REGISTROS_SAIDA) OR ((SELECT COUNT(ID_SC2) FROM SC2) < @QTD_REGISTROS_SAIDA) OR
    ((SELECT COUNT(ID_SC3) FROM SC3) < @QTD_REGISTROS_SAIDA) OR ((SELECT COUNT(ID_SC4) FROM SC4) < @QTD_REGISTROS_SAIDA) OR
    ((SELECT COUNT(ID_SC5) FROM SC5) < @QTD_REGISTROS_SAIDA))
BEGIN
  -- COLOCANDO A LINHA NA TEMPORARIA DE FORMA ALEATORIA
```

```
  SELECT TOP 1 * INTO #TEMP FROM kddcup WHERE id_kddcup >= (RAND() *
(SELECT MAX(id_kddcup) FROM kddcup)) AND flag_processado = 0

  -- VERIFICANDO QUAL TABELA IRÁ RECEBER O VALOR

  IF (((SELECT CATEGORIA FROM #TEMP) IN (SELECT ID_CATEGORIA FROM
CATEGORIA WHERE (desc_categoriaataque = 'normal' or desc_categoriaataque =
'probe'))) AND ((SELECT COUNT(ID_SC1) FROM SC1) <
@QTD_REGISTROS_SAIDA))

  BEGIN

    INSERT INTO SC1

    SELECT TEXTO=('#INPUT PATTERN '+ RTRIM(CONVERT (CHAR(7),
@A))+':'), Duration, Protocol_type, Service, Src_bytes,

      Dst_bytes, Flag, 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_hot_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_count, 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,

      TEXTO2=('OUTPUT PATTERN '+ RTRIM(CONVERT (CHAR(7), @A))+':'),
COD_CAT, Categoria

    FROM #TEMP

    SET @A=(@A + 1)

  END

  ELSE IF (((SELECT CATEGORIA FROM #TEMP) IN (SELECT ID_CATEGORIA
FROM CATEGORIA WHERE (desc_categoriaataque = 'normal' or
desc_categoriaataque = 'dos'))) AND ((SELECT COUNT(ID_SC2) FROM SC2) <
@QTD_REGISTROS_SAIDA))

  BEGIN

    INSERT INTO SC2
```

```
SELECT TEXTO=('#INPUT PATTERN '+ RTRIM(CONVERT (CHAR(7), @B))+':'), Duration, Protocol_type, Service, Src_bytes,

Dst_bytes, Flag, 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_hot_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_count, 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,

TEXTO2=('OUTPUT PATTERN '+ RTRIM(CONVERT (CHAR(7), @B))+':'), COD_CAT, Categoria

FROM #TEMP

SET @B=(@B + 1)

END

ELSE IF (((SELECT CATEGORIA FROM #TEMP) IN (SELECT ID_CATEGORIA FROM CATEGORIA WHERE (desc_categoriaataque = 'normal' or desc_categoriaataque = 'u2r'))) AND ((SELECT COUNT(ID_SC3) FROM SC3) < @QTD_REGISTROS_SAIDA))

BEGIN

INSERT INTO SC3

SELECT TEXTO=('#INPUT PATTERN '+ RTRIM(CONVERT (CHAR(7), @C))+':'), Duration, Protocol_type, Service, Src_bytes,

Dst_bytes, Flag, 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_hot_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_count, 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,

TEXTO2=('OUTPUT PATTERN '+ RTRIM(CONVERT (CHAR(7), @C))+':'), COD_CAT, Categoria

FROM #TEMP

SET @C=(@C + 1)

END

ELSE IF (((SELECT CATEGORIA FROM #TEMP) IN (SELECT ID_CATEGORIA FROM CATEGORIA WHERE (desc_categoriaataque = 'normal' or desc_categoriaataque = 'r2l'))) AND ((SELECT COUNT(ID_SC4) FROM SC4) < @QTD_REGISTROS_SAIDA))

BEGIN

INSERT INTO SC4

SELECT TEXTO=('#INPUT PATTERN '+ RTRIM(CONVERT (CHAR(7), @D))+':'), Duration, Protocol_type, Service, Src_bytes,

Dst_bytes, Flag, 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_hot_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_count, 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,

TEXTO2=('OUTPUT PATTERN '+ RTRIM(CONVERT (CHAR(7), @D))+':'), COD_CAT, Categoria

FROM #TEMP

SET @D=(@D + 1)

END

ELSE IF (((SELECT CATEGORIA FROM #TEMP) IN (SELECT ID_CATEGORIA FROM CATEGORIA WHERE (desc_categoriaataque = 'normal' or

```
desc_categoriaataque = 'dos' or desc_categoriaataque = 'r2l' or desc_categoriaataque = 'u2r' or desc_categoriaataque = 'probe'))) AND ((SELECT COUNT(ID_SC5) FROM SC5) < @QTD_REGISTROS_SAIDA))

-- ELSE IF ((SELECT COUNT(ID_SC5) FROM SC5) < @QTD_REGISTROS_SAIDA)

  BEGIN

    INSERT INTO SC5

   SELECT TEXTO=('#INPUT PATTERN '+ RTRIM(CONVERT (CHAR(7), @E))+':'), Duration, Protocol_type, Service, Src_bytes,

      Dst_bytes, Flag, 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_hot_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_count, 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,

      TEXTO2=('OUTPUT PATTERN '+ RTRIM(CONVERT (CHAR(7), @E))+':'), COD_CAT, Categoria

    FROM #TEMP

    SET @E=(@E + 1)

  END


  -- GARANTINDO QUE A LINHA NÃO SEJA PROCESSADA NOVAMENTE

  UPDATE KDDCUP SET flag_processado = 1 WHERE id_kddcup = (SELECT id_kddcup FROM #TEMP)


  -- EXCLUINDO A TEMPORARIA A CADA INTERACAO

  DROP TABLE #TEMP
```

```
  -- GARANTINDO QUE AS INTERACOES SO VAO ACONTECER ATE O NUMERO DE REGISTROS DA TABELA


  IF (@INTERACOES >= (SELECT MAX(id_kddcup) FROM kddcup))

     BREAK

  ELSE

  BEGIN

     SET @INTERACOES = (@INTERACOES + 1)

     SET @PERCENTCONCLUSAO = (CAST(@INTERACOES AS NUMERIC(18,1))/CAST((SELECT MAX(id_kddcup) FROM kddcup) AS NUMERIC(18,1)))


     PRINT '% DE CONCLUSAO: '+CAST(@PERCENTCONCLUSAO AS CHAR(20))

     CONTINUE

  END

END
```

APÊNDICE B – EXEMPLO DE RESULTADOS DO SIMULADOR DE RNA

A seguir será apresentado um exemplo de resultados obtidos no simulador de Redes Neurais Artificiais, conforme abordado na seção 5.2 do presente trabalho.

SNNS result file V1.4-3D

generated at Sat Jun 07 10:07:10 2008

No. of patterns : 12900

No. of input units : 41

No. of output units : 5

startpattern : 1

endpattern : 12900

input patterns included

teaching output included

#1.1

0 0 0.857 0 -1 -1 -1 -1 -1 0 -1 1 -1 -1 -1 -1 -1 -1 -1 0 -1 -1 0 0 -1 -1 -1 -1 1 -1

-1 0.73 0 -0.98 -0.94 -1 -1 -1 -1 -1 -1

0 0 0 0 1

0.08928 0.07216 0.00224 0.07406 0.90198

#2.1

0 0 -0.057 0 -1 0 -1 -1 1 0 -1 1 0 1 -1 0 0 -1 -1 0 -1 -1 0 0 -1 -1 -1 -1 1 -1

-1 -0.86 0 -0.56 -0.66 -0.88 -1 -1 -1 -0.88 -0.5

0 0 0 0 1

0.08903 0.0051 0.03107 0.05022 0.99194

#3.1

-1 0 0.371 0 -1 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 -1 -1 0 0 -1 -1 -1 -1 1 -1

-1 -0.99 0 1 -1 1 -0.46 -1 -1 -1 -1

0 0 0 0 1

0.06395 0.01999 0.09681 0.05325 0.90172

#4.1

0 0 -0.057 0 -1 0 -1 -1 -1 0 -1 1 0 1 -1 0 0 -1 0 0 -1 -1 0 0 -1 -1 -1 -1 1 -1

-1 1 0 0.1 -0.94 -1 -1 -0.04 0.72 -0.96 -0.98

0 0 0 0 1

0.03711 0.07676 0.01107 0.02007 0.98403

#5.1

0 0 -0.057 0 -1 0 -1 -1 0 0 -1 1 0 1 -1 0 -1 -1 0 0 -1 -1 0 0 -1 -1 -1 -1 1 -1

-1 -0.25 0 -0.18 -0.9 -0.98 -0.96 -1 -1 -0.66 0.04

0 0 0 0 1

0.04281 0.01599 0.01233 0.05641 0.99411

#6.1

-1 0 -0.943 0 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 0 -1 -1 0 0 -1 -1 -1 -1 1 -1

-1 1 1 1 -1 -1 -1 -1 -1 -1 -1

1 0 0 0 0

0.96573 0.03545 0.00109 0.01696 0.04

#7.1

-1 0 -0.943 0 -1 0 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 0 -1 -1 0 0 -1 -1 -1 -1 1 -1

-1 -0.45 1 1 -1 -0.98 -0.92 -1 -1 -1 -1

1 0 0 0 0

0.94119 0.04374 0.00081 0.04495 0.05414

#8.1

-1 1 0.486 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 -1 -1 0 0 -1 -1 -1 -1 1 -1

-1 1 0 1 -0.98 -1 -1 -1 -1 -1 -1

1 0 0 0 0

0.96202 0.01069 0.01286 0.02432 0.03184

#9.1

-1 1 0.486 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 -1 -1 0 0 -1 -1 -1 -1 1 -1

-1 1 0 1 -0.98 -1 -1 -1 -1 -1 -1

1 0 0 0 0

0.96202 0.01069 0.01286 0.02432 0.03184

#10.1

0 0 -0.057 0 -1 0 -1 -1 -1 0 -1 1 0 -1 -1 0 0 0 -1 0 -1 -1 0 0 -1 -1 -1 -1 1 -1

-1 0.94 0 -0.74 -0.94 -1 -1 -1 -1 -1 -1

0 0 0 0 1

0.01002 0.03485 0.02256 0.09628 0.98293

APÊNDICE C – PADRÕES DE ATAQUE DO TIPO NEGAÇÃO DE SERVIÇO

Existem 11 diferentes padrões de ataque e intrusão da classe negação de serviço na base de dados do Kddcup 1999 utilizada neste trabalho. Uma breve descrição da operação de cada um destes ataques será apresentada neste apêndice.

3.1 *Apache2*

Ataques *Apache2* têm como alvos servidores *Web* (servidores que interagem com navegadores Internet utilizando protocolo HTTP) *Apache*. O ataque consiste no envio de requisições iniciais do protocolo HTTP contendo um alto número de cabeçalhos HTTP. Enquanto uma requisição HTTP normal possui no máximo 20 cabeçalhos HTTP, uma requisição gerada para um ataque *Apache2* possui, tipicamente, mais de 1000 cabeçalhos HTTP. O conteúdo de cada cabeçalho não é importante. Servidores *Apache* recebendo estas solicitações anômalas aumentam seu consumo de recursos computacionais como processador e memória reduzindo drasticamente o seu desempenho e tempo de resposta para usuários legítimos. Em alguns casos, o servidor vítima deste ataque pode se tornar completamente inoperante, parando de responder a quaisquer solicitações de usuários.

3.1 Back

Também direcionado contra servidores *Web* Apache, o ataque *Back* consiste em envio de solicitação HTTP contendo URL com um alto número, tipicamente mais de 100, de caracteres “/” (barra invertida). Ao receber uma solicitação com este URL anômalo, servidores *Web* Apache2 aumentam sua utilização de CPU afetando o desempenho geral do sistema vítima. O processamento usualmente é normalizado com o fim do ataque.

3.3 *Land*

O padrão de ataque *Land* tem como alvo determinadas implementações do conjunto de protocolos TCP/IP. O único sistema vulnerável a este padrão de ataque existente no ambiente montado pelo projeto DARPA/MIT era o sistema operacional SunOS 4.1. *Land* consiste no envio de um pacote *TCP-Syn* (usado para iniciar a negociação de inicio de conexão TCP) com endereço de origem igual ao endereço de destino. Ao receber este pacote o sistema operacional SunOS 4.1 ficava completamente inoperante. Para recuperar este sistema é necessário desligar e ligar o computador. É um padrão de ataque facilmente detectado por sistemas de detecção de intrusão uma vez que um pacote TCP/IP com endereços de origem e destino idênticos jamais deveria trafegar em redes TCP/IP.

3.4 *MailBomb*

MailBomb consiste no envio de milhares de mensagens de e-mail para um único usuário com o objetivo de sobrecarregar o processamento de servidor de e-mail (baseados no protocolo SMTP). Implementações típicas deste ataque enviam 10000 mensagens de 1Kbyte (10 Mbytes de dados) para um único usuário continuamente. O efeito deste ataque está reduzido a uma sobrecarga de processamento no servidor SMTP e ao recebimento de milhares de mensagens inúteis na caixa postal do usuário. Identificação de um ataque *Mailbomb* é relativamente subjetiva e depende de uma definição de quantas mensagens de um mesmo remetente para um mesmo usuário, em um determinado intervalo de tempo, podem ser consideradas normais para cada organização.

3.2 SynFlood

Ataques *SynFlood*, também conhecidos como *Neptune* devido ao nome dado ao primeiro programa que implementou esta técnica de intrusão, abusam de característica normal e padrão do protocolo TCP. Antes de qualquer comunicação baseada no protocolo TCP começar é necessário o estabelecimento de uma

conexão virtual entre as partes envolvidas na comunicação. Esta “conexão” é estabelecida durante um “*handshake*” ou negociação inicial. O computador destino de uma negociação de uma nova conexão TCP aloca memória para a nova conexão assim que recebe o primeiro pacote TCPSyn do cliente. Se não for recebido o 3º pacote do processo (TCP-Ack) após um determinado período o computador destino assume que algum problema ocorreu e libera os recursos pré-alocados. O ataque SynFlood explora esta alocação inicial de recursos enviando um enorme número de pacotes TCP-Syn ao servidor sem, entretanto, completar a conexão. O servidor alocará recursos para conexões falsas comprometendo recursos que poderiam estar disponíveis para conexões legítimas e, em casos extremos, ficando completamente inoperante devido à sobrecarga de solicitações. Algumas implementações TCP/IP quando vítimas de um ataque *Synflood* acabam incorrendo em uma situação de exceção e paralisando completamente o funcionamento do sistema operacional. Um sistema de detecção de intrusão para redes pode detectar este ataque pela monitoração e comparação do número de pacotes TCP-Syn enviados para determinado computador contra um determinado limiar máximo. Um sistema de detecção de intrusão para *host* também pode ser eficaz em detectar este tipo de ataque através da monitoração do crescimento das áreas de memória reservadas pelo sistema operacional para estruturas de controle de conexões TCP.

3.6 *Ping of Death*

Ataque negação de serviço que consiste no envio de pacotes ICMP-Echo (gerados normalmente através do utilitário “*ping*” para diagnóstico) com tamanhos superiores a 64000 bytes. Como pacotes ICMP-Echo são normalmente bem inferiores a este valor diversos sistemas operacionais e implementações de TCP/IP disponíveis não conseguiam tratar esta anomalia e reagiam de forma inesperada, onde as reações mais comuns eram reinicialização do computador ou total paralisação de todas as operações. Sistemas de detecção de intrusão podem facilmente detectar este padrão de ataque monitorando o tamanho de pacotes ICMP-Echo.

3.7 *ProcessTable*

O ataque *ProcessTable* tem como alvo, diversas variantes do sistema operacional Unix. O princípio básico para o funcionamento deste ataque, está relacionado a como sistemas operacionais Unix lidam com conexões e solicitações recebidas pela rede. Tipicamente um novo processo é carregado em memória para lidar com a nova conexão e/ou requisição recebida. Para usuários regulares o sistema operacional limita a quantidade de processos que cada usuário pode carregar em memória. Este limite não é feito, entretanto, para o usuário "*root*" (usuário com poder de administrador), que normalmente é o usuário utilizado por processos associados a funções de rede do sistema operacional. Um sistema vítima deste ataque tem sua tabela interna de controle de processos ativos sobrecarregada impedindo a ativação de novos processos legítimos.

3.8 *Smurf*

Em um ataque *Smurf* o atacante envia pacotes *ICMP-Echo* com endereço de destino de "broadcast" (todas as máquinas em uma determinada rede) e endereço de origem igual ao do sistema alvo do ataque. O sistema vítima terá uma sobrecarga ao receber milhares de respostas (pacotes *ICMPEcho-Reply*) de solicitações que ele não realizou. Um sistema de detecção de intrusão pode facilmente detectar um ataque *Smurf* observando a quantidade de pacotes *Echo-Reply* recebidos em relação a quantidade de pacotes *Echo-Request* enviados. Em uma situação ideal normal estes valores devem ser iguais ou bem próximos.

3.9 *Syslog*

Ataque de negação de serviço que tem como alvo o serviço *Syslog* do sistema operacional Solaris. *Syslog* é o serviço de registro de eventos e auditoria da maioria dos sistemas operacionais Unix. Quando o serviço *Syslog* recebe um registro de evento para ser armazenado de um computador remoto ele tenta resolver qual nome DNS está associado ao endereço IP do computador remoto. Se esta resolução não

for possível uma falha no Syslog causa sua indisponibilidade. O serviço só restaurará sua operação normal após intervenção de um administrador do sistema. Este é um ataque que explora uma falha ou "*bug*" em um serviço de rede. Esta falha ou "*bug*" está relacionada ao não tratamento de situações excepcionais ou não previstas em condições normais.

3.10 *TearDrop*

Ataque que explora falha no tratamento de pacotes fragmentados em diversas implementações de TCP/IP em vários sistemas operacionais. Um pacote TCP/IP pode encontrar um enlace entre a origem e destino que não suporte trafegar pacotes maiores do que determinado tamanho (parâmetro conhecido como MTU ou "*Maximum Transmission Unit*"). Quando isto ocorre este pacote é fragmentado em um ou mais pacotes. É função do destinatário do pacote realizar a operação de reagrupar estes fragmentos no pacote original. Algumas implementações de TCP/IP não conseguiam realizar este reagrupamento se houvesse alguma sobreposição de dados entre os diversos fragmentos recebidos. Estas implementações causavam a reinicialização do sistema operacional se uma situação não esperada como esta fosse encontrada.

3.11 *UDPStorm*

Ataque de negação de serviço que causa congestionamento e alterações no desempenho de redes TCP/IP. Alguns serviços UDP, como "*chargen*", "*echo*" e outros, geram resposta automática sem qualquer verificação de origem e autenticidade da solicitação original recebida. Um atacante que envie um pacote com endereço de origem alterado para o da vítima e envie para um serviço desta classe pode gerar um processo repetitivo de comunicação entre o prestador do serviço e a máquina alvo.

APÊNDICE D - PADRÕES DE ATAQUE DO TIPO RECONHECIMENTO

Cinco padrões de ataques da classe Reconhecimento ou "*Probing*" estão presentes na base de dados MIT/KddCup 1999. São eles:

4.1 *IPSweep*

IPSweep é um ataque utilizado na etapa de reconhecimento e seleção de alvos vulneráveis. Consiste em uma varredura ou busca por computadores e sistemas acessíveis e que possam ser atacados por outros métodos em uma próxima etapa. *IPSweep* utilizado neste trabalho envia pacotes *ICMP-Echo* (utilitário *Ping*) para todos os endereços de uma determinada rede e constrói uma relação de respostas recebidas. Um sistema de detecção de intrusão que monitore uma seqüência de pacotes *ICMP-Echo* para endereços de destino em uma faixa continua e vindos de um mesmo endereço de origem será capaz de detectar este padrão.

4.2 *Mscan*

MScan é um ataque de reconhecimento que utiliza consulta a servidores DNS e realiza uma busca nos endereços IP encontrados procurando por vulnerabilidades específicas. Com código fonte disponível na Internet, esta ferramenta de ataque pode ser customizada para encontrar diversas vulnerabilidades em potenciais nas máquinas selecionadas através do DNS. Para o projeto MIT/DARPA foram realizados ataques *MScan* para todos os computadores pertencentes ao domínio eyrie.af.mil procurando pelas seguintes vulnerabilidades: statd, imap,pop, computadores IRIX com contas de usuários sem senha, bind, diversas vulnerabilidades cgi-bin em serviços *Web*, NFS e serviços X.

4.3 *Nmap*

Nmap é uma ferramenta de varredura e reconhecimento capaz de realizar busca por computadores, serviços e vulnerabilidades utilizando diversos mecanismos. Sua principal função é determinar quais portas TCP ou UDP estão ativas em determinado computador, conseqüentemente, determinando quais serviços estão ativos. A ferramenta permite ainda a identificação exata ("*fingerprinting*") de versão dos softwares que implementam os serviços descobertos. Esta informação é útil para um atacante determinar quais vulnerabilidades podem ser exploradas em uma 2ª etapa do ataque. Um aspecto da ferramenta *Nmap* que pode dificultar sua detecção é a capacidade de configurar diversos aspectos relativos à velocidade e periodicidade em que as varreduras são realizadas.

4.4 *Saint*

"*Security Administrator Integrated Network Tools*" (*Saint*) é uma ferramenta desenvolvida para administradores de redes, sistemas e profissionais de segurança da informação. Esta ferramenta tem como objetivo levantar o maior número de informações sobre serviços ativos em computadores remotos. Apesar de não ter sido desenvolvida com intenção de ser utilizada para ataques e intrusões, a enorme quantidade de informações levantadas pode ser útil para um invasor/atacante.

4.5 *Satan*

"*Security Administrator Tool for Analyzing Networks*" (*Satan*) pode ser considerado o predecessor da ferramenta *Saint* descrita anteriormente. O princípio das duas ferramentas é essencialmente o mesmo, sendo que a diferença entre elas reside apenas nos serviços e vulnerabilidades pesquisados. Um sistema de detecção de intrusão de rede pode detectar facilmente uma varredura utilizando o *Satan* devido ao perfil regular de tráfego gerado pela ferramenta.

APÊNDICE E - PADRÕES DE ATAQUE DO TIPO REMOTO PARA LOCAL

Padrões de ataque da categoria “Remoto para local” são extremamente perigosos pois normalmente permitem que um atacante com apenas a capacidade de enviar pacotes TCP/IP para a vítima, mas sem nenhum outro tipo de acesso, obtenha acesso não autorizado como um usuário legítimo. Nove padrões desta categoria estão presentes na base de dados MIT/KddCup 1999. São eles :

5.1 *Dictionary*

Padrão de ataque e intrusão clássico em que um atacante tenta obter acesso não autorizado a determinado computador ou serviço através de inúmeras tentativas de descobrir a conta e senha de um usuário válido. A principal vulnerabilidade explorada por este ataque é humana e reside no fato de que os usuários tipicamente utilizam senhas fáceis de serem lembradas. Todo e qualquer serviço de rede que empregue autenticação de usuário é vulnerável a este tipo de ataque. Diversas ferramentas (“*Lophtcrack*”, “*John the Ripper*” etc) existem para realizar automaticamente as tentativas de autenticação. Estas ferramentas fazem uso de dicionários de palavras no idioma do usuário para agilizar a descoberta da senha, daí o nome do padrão de ataque. Se o usuário utilizar uma palavra existente no dicionário a ferramenta será capaz de identificar sua senha mais rapidamente do que em tentativas por força bruta (tentativa de todas as combinações de caracteres alfanuméricos até um determinado tamanho máximo de senha). No projeto MIT/DARPA um script chamando “*NetGuess*” realizava entre 10 e 100 tentativas de autenticação nos serviços ftp, pop e telnet utilizando um arquivo de dicionário e permutações simples envolvendo o nome do usuário. Para que sistemas de detecção de intrusão possam detectar este tipo de ataque eles precisam possuir conhecimento prévio dos protocolos empregados por cada aplicação que possua autenticação remota. Basta então monitorar a quantidade de erros de autenticação em determinado intervalo de tempo para acusar uma intrusão deste tipo.

5.2 *FTP-Write*

Padrão de ataque que explora falha de configuração em serviços FTP que fornecem acesso para usuários não autenticados (anônimos). Se o diretório raiz de um servidor FTP tiver como proprietário o usuário "*ftp*" ou seu proprietário estiver no mesmo grupo do usuário "ftp" é possível que um usuário anônimo tenha acesso para escrita de arquivos no servidor FTP e até mesmo obtenha acesso remoto ao servidor.

5.3 *Guest*

Variante do padrão de ataque *Dictionary* descrito anteriormente, no ataque *Guest* é realizado a tentativa de acesso não autorizado utilizando apenas a conta de usuário "*guest*". Esta conta de usuário existe em diversos sistemas operacionais modernos, e usualmente é habilitado em instalação padrão destes sistemas.

5.4 *Imap*

Imap é um ataque que explora uma falha de estouro de "*buffer*" ("*buffer overflow*") em serviço Imap de servidores RedHat Linux 4.2 e que permite ao atacante a execução de código arbitrário no servidor afetado. Uma ferramenta disponibilizada na Internet com instruções de uso e denominada "*Impack 1.03 Attack Toolkit*" foi utilizada no projeto DARPA/MIT. A existência de ferramentas como esta tornam o ataque ainda mais perigoso uma vez que usuários sem conhecimentos técnicos sobre estouro de "*buffer*" podem seguir as instruções de uso e realizar com sucesso um ataque deste tipo.

5.5 Named

Ataque que explora uma falha de estouro de “*buffer*” em determinadas versões de servidores DNS (*Domain Name System*) Bind. Uma consulta de informações de DNS reverso (obter informações sobre nome, dado um endereço IP), especialmente preparada pode tornar o serviço inoperante ou permitir ao atacante a execução de código no servidor. Um servidor de DNS Bind aloca uma área de memória de no máximo 4096 bytes para uma consulta de DNS reverso. Este padrão de ataque constrói uma consulta que extrapola este limite causando um estouro de “*buffer*”. Um sistema de detecção de intrusão de redes pode, para detectar este tipo de ataque, monitorar consultas deste tipo que ultrapassem 4096 bytes em sua área de dados.

5.6 *Phf*

Phf é um exemplo de programa CGI – “*Common Gateway Interface*”, mecanismo de interação entre aplicações em um servidor *Web* e programas do sistema operacional, disponibilizado com todo servidor *Web* Apache. Enviando parâmetros especialmente criados para comprometer este programa um atacante pode, por exemplo, copiar o arquivo “*passwd*” com os usuários e senhas de um sistema Unix vulnerável.

5.7 *Sendmail*

O ataque *Sendmail* explora uma falha de estouro de “buffer” na versão 8.8.3 do software de correio eletrônico SMTP que permite a execução de código com privilegio de administrador/“*root*”. Através do envio de uma mensagem de e-mail para o servidor alvo, especialmente construído para causar o estouro de “*buffer*”, o atacante consegue forçar o serviço *sendmail* a executar código de sua escolha.

5.8 *Xlock*

Neste ataque, o atacante irá enganar o usuário de uma sessão gráfica X a digitar sua senha em uma versão alterada (cavalo de Tróia) do programa *Xlock*, utilizado para bloquear uma sessão gráfica X, depois de determinado período de inatividade. Uma versão modificada do programa *Xlock* foi desenvolvida pelo projeto MIT/DARPA especialmente para simular este ataque. Para este ataque ser bem sucedido o atacante tem que ter algum tipo de acesso à máquina do usuário para instalar a versão alterada do programa *Xlock*.

5.9 *Xsnoop*

Neste ataque, o atacante monitora todas as teclas digitadas e enviadas por usuários para um servidor X desprotegido. Este registro de teclas enviadas pode conter informações confidenciais que serão úteis ao atacante em novos ataques.

APÊNDICE F – PADRÕES DE ATAQUE DO TIPO USUÁRIO PARA SUPERUSUÁRIO

Sete padrões de ataques da classe "*Usuário para Superusuário*" ("*User to Root*") estão presentes na base de dados de treinamento e testes, desenvolvida pelo MIT/DARPA e utilizada neste trabalho. São eles:

6.1 *Eject*

Explora uma falha de estouro de "buffer" existente no utilitário "*eject*" disponibilizado no sistema operacional Solaris 2.5. *Eject* é um utilitário para manuseio de mídias removíveis. Um usuário com privilégios simples em um sistema vulnerável que explore esta falha passará a ter privilégios de administração e supervisão ("*root*").

6.3 Ffbconfig

Bastante similar ao ataque *Eject* descrito anteriormente, este padrão de ataque explora um estouro de "buffer" no programa *Ffbconfig* utilizado no sistema Solaris 2.5 para configurar placas gráficas instaladas no sistema. Devido à não verificação de tamanho de parâmetros recebidos pelo programa é possível que um atacante sobrescreva áreas de memória interna do programa obtendo acesso não autorizado.

6.4 Fdformat

Novamente uma falha de estouro de "buffer" é explorada para elevar o privilégio de usuário comum para usuário com poder de administração. O programa *fdformat*, vulnerável a este ataque, é utilizado para formatar disquetes e cartões de memória PCMCIA.

6.5 LoadModule

Ataque contra sistema operacional SunOS 4.1 que utilizem o sistema *xnews* de janelas. O programa *xnews* utiliza o módulo *loadmodule* para carregar alguns dispositivos em memória. Devido a um erro de programação no módulo *loadmodule* um atacante pode obter privilégios indevidos de administração.

6.6 Perl

Explora falha em algumas versões da linguagem de scripts *Perl*. Um módulo denominado *suidperl* apresenta erro de programação que permite que qualquer usuário com conta ativa no sistema alvo obtenha privilégios de administração.

6.7 OS

Ataque que explora uma falha no utilitário *PS* do sistema operacional Solaris 2.5. Se o usuário tiver permissão de escrita em diretórios temporários, ele pode explorar esta falha para executar códigos com privilégios de administração no sistema alvo.

6.8 Xterm

Explora uma falha que permite estouro de "*buffer*" na biblioteca *Xaw* do programa xTerm em versões de sistema operacional *RedHat Linux*. Permite ao atacante obter privilégios de administração supervisão a partir de uma conta de usuário comum.