UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UTILIZAÇÃO DE UM AMBIENTE DE HONEYNET NO TREINAMENTO DE REDES NEURAIS ARTIFICIAIS PARA DETECÇÃO DE INTRUSÃO

DANIEL LYRA ROCHA

ORIENTADOR: RAFAEL TIMÓTEO DE SOUSA JUNIOR

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA DE REDES DE COMUNICAÇÃO

PUBLICAÇÃO: PPGENE.DM – 259/06

BRASÍLIA / DF: MAIO/2006

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UTILIZAÇÃO DE UM AMBIENTE DE HONEYNET NO TREINAMENTO DE REDES NEURAIS ARTIFICIAIS PARA DETECÇÃO DE INTRUSÃO

DANIEL LYRA ROCHA

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

RAFAEL TIMÓTEO DE SOUSA JUNIOR, Doutor, UnB
(ORIENTADOR)

NOME DO MEMBRO DA BANCA, Título, Instituição
(EXAMINADOR INTERNO)

NOME DO MEMBRO DA BANCA, Título, Instituição
(EXAMINADOR EXTERNO)

NOME DO MEMBRO DA BANCA, Título, Instituição
(SUPLENTE)

DATA: BRASÍLIA/DF, DIA DE MÊS DE ANO.

FICHA CATALOGRÁFICA

ROCHA, DANIEL LYRA.
Utilização de um ambiente de honeynet no treinamento de redes neurais artificiais para detecção de intrusão [Distrito Federal] 2006.
xii, 193p., 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2006).

Dissertação de Mestrado – Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1.Detecção de Intrusão 2. Redes Neurais Artificiais
3. Honeynet

I. ENE/FT/UnB. II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

ROCHA, DANIEL. LYRA. (2006). Utilização de um ambiente de honeynet no treinamento de redes neurais artificiais para detecção de intrusão. Dissertação de Mestrado, Publicação 259/2006, Departamento de Engenharia Elétrica, Universidade de Brasília , Brasília , DF, 193p.

CESSÃO DE DIREITOS

AUTOR: Daniel Lyra Rocha
TÍTULO: Utilização de um ambiente de honeynet no treinamento de redes neurais artificiais para detecção de intrusão.
GRAU: Mestre ANO: 2006

Daniel Lyra Rocha
SRES Q.10 Bl O-1 Casa 15 – Cruzeiro Velho
CEP 70.645-515– Brasília – DF - Brasil

AGRADECIMENTOS

Ao meu orientador Prof. Dr. Rafael Timóteo de Sousa Júnior, pela total confiança e apoio.

Ao Prof. Ricardo Staciarini Puttini, co-orientador do trabalho.

Ao amigo Júnior, parceiro de projeto final de graduação. Entramos juntos no mestrado e escolhemos juntos o tema. A partir de uma primeira pesquisa mais apurada, cada um escolheu um foco para trabalhar, mas sempre se ajudando.

Aos meus colegas de trabalho Carlos Versati, Laerte Peotta, Marco Antônio, Nilson Borges e Fernando Higa que não mediram esforços na hora de me incentivar e ajudar no que fosse necessário na parte técnica.

Cada um deles teve grande contribuição neste trabalho. Sem o Versati, eu não saberia por onde começar a programar o *sniffer* em "C", chamado *dsni*. O Peotta me ensinou muito sobre *honeynet*, que fora seu objeto de trabalho anteriormente. O Marco ajudou no entendimento de IDS. O Nilson disponibilizou as máquinas e forneceu total apoio para que fosse montada uma *honeynet* no ambiente de trabalho. O Higa possibilitou de imediato minha mudança de turno no trabalho para que eu pudesse terminar a dissertação.

À minha família, que demonstrou preocupação e total incentivo durante o período de realização da dissertação.

E aos meus amigos de todos os finais de semana, sem esquecer da Liana, minha namorada que sempre me incentivou e entendeu as vezes que precisei ficar e fazer a dissertação.

Aos meus pais, irmãos, amigos e namorada.

RESUMO

UTILIZAÇÃO DE UM AMBIENTE DE HONEYNET NO TREINAMENTO DE REDES NEURAIS ARTIFICIAIS PARA DETECÇÃO DE INTRUSÃO

Autor: Daniel Lyra Rocha

Orientador: Rafael Timóteo de Sousa Júnior

Programa de Pós-graduação em Engenharia Elétrica

Brasília, mês de maio (2006)

O trabalho descrito nesta dissertação objetiva realizar uma contribuição acadêmica no avanço da utilização de redes neurais artificiais para detecção de intrusão. De uma maneira bem didática o leitor vai aprendendo, capítulo a capítulo, sobre os paradigmas envolvidos no problema. No último capítulo, ele pode acompanhar uma prova de conceito real sobre a proposta realizada. Trata-se da utilização de um rede real para a coleta de dados de intrusão que servirão para o treinamento de redes neurais artificiais voltadas para detecção de intrusão. A contribuição sobre os trabalhos já realizados no assunto anteriormente é a utilização de uma honeynet para coleta dessas intrusões.

ABSTRACT

USE OF A HONEYNET ENVIRONMENT IN THE TRAINING OF ARTIFICIAL NEURAL NETS FOR INTRUSION DETECTION

Author: Daniel Lyra Rocha

Supervisor: Rafael Timóteo de Sousa Júnior

Programa de Pós-graduação em Engenharia Elétrica

Brasília, month of may (2006)

The described work in this thesis objective to carry through an academic contribution in the advance of the use of artificial neural nets for intrusion detention. In a well didactic way the reader goes learning, chapter after chapter, on the involved paradigms in the problem. In the last chapter, it he can follow a test of real concept on the proposal carried through. One is about the use of a real net for the collection of intrusion data that will serve for the training of artificial neural nets directed toward intrusion detention. The contribution on the carried through works already in the subject previously is the one use honeynet for collection of these intrusions.

SUMÁRIO

Capítulo **Página**

LISTA DE TABELAS

Tabela **Página**

LISTA DE FIGURAS

Figura **Página**

1.INTRODUÇÃO

No começo da década de 60, diversos pesquisadores das principais Universidades americanas começaram a desenvolver trabalhos paralelos a respeito de interligação de computadores. A idéia entusiasmou o governo americano, que planejava uso militar da nova tecnologia e criou a ARPA (A*dvanced Research Projects Agency*), passando a incentivar pesquisas na área. Mais tarde, diversas Universidades passaram a se interligar através de uma rede batizada de ARPANET. Uma arquitetura de protocolos surgiu com o objetivo de interligar computadores independentemente de seus dispositivos físicos. Foi batizada de TCP/IP em alusão a seus dois principais protocolos de comunicação [1,3].

A ARPANET cresceu exponencialmente e diversas redes de computadores foram acrescentadas a ela. A arquitetura flexível e em camadas incentivou a criação de diversos protocolos interessantes de aplicação, que por sua vez incentivaram a criação de várias aplicações úteis e interessantes para o dia-a-dia não só de universitários ou militares, mas de qualquer indivíduo. Um novo mundo sem fronteiras, extremamente conectado e anônimo surgiu na vida de pessoas comuns, que não necessariamente sabem alguma coisa de informática. Os softwares passaram a lidar com a comunicação de forma cada vez mais transparente ao usuário final, permitindo que ele pudesse realizar tarefas complexas com um simples comando ou toque no *mouse*.

As mudanças ocorridas, entretanto, não eram imaginadas no começo e não foram planejadas, nem coordenadas. Os protocolos e aplicações foram surgindo de todas as maneiras, por todos os lados do mundo. Cada pessoa, anônima ou não contribuiu com o que podia para que a Internet funcionasse como hoje. Tudo era uma grande brincadeira, onde podia-se trocar mensagens, ler notícias, jogar virtualmente, conhecer pessoas, adquirir conhecimento e muitos outros passatempos. Entretanto, a partir de um certo momento, descobriu-se que a Internet poderia “trabalhar”, ou servir como instrumento de trabalho, muitas vezes o principal intrumento. Surgiu o *e-commerce* e as lojas virtuais, o mercado virtual, surgiram os contatos de emprego pela Internet, surgiram os bancos pela Internet: extremamente práticos e econômicos. De repente, boa parte das instituições estava representada na Internet. Diversos países, diversas empresas, diversos negócios. Tudo está na Internet. A informação que lá trafega é simplesmente todos os tipos de informação do mundo. Da mais banal à mais valiosa. Da informação de utilidade pública aos segredos mais

irreveláveis. De iniciativas solidárias à pedofilia. A Internet tornou-se a entidade mais democrática do mundo. A rede tornou-se um mundo paralelo, um novo Universo, com infinitas possibilidades.

Como em todas as esferas da vida, existem pessoas bem intencionadas e pessoas mal intecionadas conectadas à rede. O crescimento desregrado da Internet permitiu a criação de protocolos e softwares que são uma verdadeira “caixa preta” para alguns e uma verdadeira fonte de estudos para outros. A documentação do funcionamento da Internet é extensa na rede, a troca de informações é fácil e freqüente. Após adquirirem o conhecimento da arquitetura, dos protocolos, serviços e estruturas, as pessoas os utilizam de três maneiras: propondo melhorias nas aplicações e protocolos existentes ou criando novos de acordo com o interesse e necessidade; explorando falhas em protocolos e softwares existentes; apenas operando as soluções existentes. Neste último caso, há operadores benígnos: aqueles que apenas querem fazer o seu trabalho ou usufruir das vantagens da rede sem prejudicar ninguém; e operadores malígnos: aqueles que utilizam softwares e soluções propostas por estudiosos malignos visando prejudicar pessoas ou instituições e levar vantagem, seja ela financeira ou não.

Por causa dessa bipolarização de idéias e intensões, a Internet virou uma espécie de campo de batalha. Uma batalha invisível e interminável. Uma batalha irreversível e imprevisível[44]. De um lado pessoas anônimas, mal intensionadas, bem informadas, distribuídas, organizadas ou não: os atacantes. Serão chamados atacantes, pois o conceito de *hackers*, como são conhecidos, é um tanto quanto controverso na literatura. Alguns consideram que *hackers* podem ser tanto pessoas que estudam as falhas de segurança dos softwares para prejudicar e levar vantagens, quanto pessoas que estudam o mesmo tipo de falhas na intenção de melhorar a segurança, de divulgar as falhas para que possam ser corrigidas. Este é o outro lado da batalha. Profissionais e entusiastas da área de segurança da informação. Pessoas que estudam o assunto na intenção de combater os atacantes e principalmente proteger as instituições, os serviços da Internet e a si próprio. Eles são os defensores. É para eles que estudos como este são publicados. Qualquer serviço ou empresa que tenha alguma importância na Internet precisa deles. E eles precisam de qualquer informação relevante de como proteger sistemas e patrimônios virtuais.

Entretanto, para proteger é preciso conhecer o outro lado. Saber como o atacante pensa, como ele age, do que ele precisa. Nada melhor que conhecer o inimigo para enfrentá-lo

em iguais condições. É, portanto, uma batalha em que os dois lados utilizam as mesmas armas, o mesmo conhecimento. O atacante faz de tudo para permanecer anônimo, escondido. Não tem nada a perder, a não ser seu próprio tempo. Pode ganhar dinheiro ou status, satisfazendo seu próprio ego com invasões não permitidas, embora vez ou outra ele se entregue, por necessidade de se comunicar e trocar informações, ou por preciosismo ou exibicionismo. O defensor, como indivíduo, também é anônimo. Todavia, ele defende um patrimônio palpável, disponível virtualmente e muitas vezes precisa mostrar a cara, fornecer informações preciosas. A batalha é eterna e a ameaça cresce a cada dia. Um dia conseguirão parar a Internet? Torná-la obsoleta por falta de segurança? São possibilidades não nulas, como tudo na vida. Resta aos defensores a esperança. E seguir trabalhando, seguir defendendo. O presente trabalho pretende contribuir para a comunidade de segurança da informação no sentido de prover uma pesquisa acadêmica e uma prova de conceito de uma solução relativamente nova e promissora na detecção de padrões de ataques: as redes neurais artificiais.

As RNAs são modelos matemáticos não lineares que buscam atingir características de redes neuronais biológicas reais do cérebro humano. Características tais como aprendizagem, associação, generalização e abstração. Existem elementos somadores, chamados de neurônios, que estão conectados com outros do mesmo tipo através de conexões sinápticas. Os parâmetros matemáticos dessas conexões: pesos e *bias*, carregam o “conhecimento” da rede. Há um período de treinamento, onde estes parâmetros são ajustados conforme as entradas da rede vão sendo apresentadas. Neste período a rede “se acostuma” às entradas e se prepara para responder da mesma maneira quando uma entrada com o mesmo padrão for apresentada após o treinamento. É a capacidade de generalização das redes neurais aplicada no reconhecimento de padrões. As entradas são os parâmetros do pacote IP, que podem caracterizar um ataque ou um tráfego normal. Após o treinamento, a rede será capaz de reconhecer entre um pacote normal e um pacote de ataque[17,18,19,20,21,22,23,24,25,26].

A etapa mais crítica na construção de uma solução deste tipo é justamente o treinamento da rede neural. É neste período que se dá o aprendizado. É necessário escolher corretamente os parâmetros da rede, como número de neurônios, valores iniciais, critério de parada e, principalmente, dados de treinamento. É necessário que os dados representem, o mais fielmente possível, os dados que serão detectados posteriormente. A grande dificuldade é esta. Não há como prever que tipo de ataque a rede irá sofrer e simular esses ataques em

laboratório. O ideal é que se treine com dados reais de ataques sofridos em produção. Aí surge outro problema: classificar corretamente o pacote em ataque e não ataque. Existem mecanismos de segurança que conseguem realizar essa classificação. São chamados de IDS (*Intrusion Detection Systems*). A grande limitação destes mecanismos é que eles são baseados em assinatura, ou seja, para um ataque ser reconhecido, ele precisa estar na base de assinaturas do IDS, exatamente como está acontecendo na produção. Uma simples mudança ou fragmentação no pacote pode "enganar" o IDS convencional. A vantagem de um mecanismo baseado em redes neurais é a capacidade de generalização, podendo reconhecer um ataque apenas pelo seu padrão, mesmo que ele não tenha sido treinado anteriormente. Quer dizer, ataques novos e desconhecidos, que não podem ser detectados por um mecanismo baseado em assinatura, teoricamente podem ser reconhecidos por um algoritmo não linear, como o de redes neurais.

Permanece o desafio: como coletar dados de entrada que representem os possíveis ataques à rede de produção e fugir da dificuldade de capturar ataques não reconhecidos para treinamento? A solução proposta no trabalho é montar um dispositivo de segurança chamado *honeynet*. Uma rede real, rodando serviços e aplicativos reais, construída exclusivamente para ser invadida e coletar dados de ataque. Quanto mais próxima for esta rede da rede de produção, mais fiel será o treinamento e melhores resultados ele trará. Todo e qualquer tráfego coletado em uma *honeynet* será tráfego não autorizado, ou seja, ataque. Não existem serviços legítimos funcionando nela. Ataques conhecidos e desconhecidos serão coletados e treinados. A proposta do presente trabalho é, além de construir um protótipo da solução discutida, fornecer subsídios teóricos para que o leitor menos entendido do assunto possa compreender cada variável envolvida no problema e na proposta. Por este motivo, a dissertação foi dividida de uma maneira bem didática.

O capítulo 2 é uma contextualização básica sobre o funcionamento da internet e dos protocolos de comunicação. Este conhecimento será essencial para a compreensão do restante do trabalho. Após uma breve explanação histórica, o leitor poderá conhecer os principais protocolos da arquitetura TCP/IP, camada a camada, com ênfase nos protocolos IP e TCP. Os protocolos de aplicação, devido à quantidade e diversidade, não foram detalhados. O capítulo seguinte trata de questões relativas à segurança dos principais.

O capítulo 3 é destinado à segurança da informação. É importante adquirir a cultura de segurança e ter em mente a infinidade de ataques que os sistemas e protocolos estão sujeitos e

as possíveis proteções e precauções a serem tomadas. Após uma visão geral da segurança da informação, o capítulo discute uma série de vulnerabilidades a que os protocolos estão sujeitos. Destaque para os protocolos de aplicação que não foram citados anteriormente. Depois, são mostrados os principais tipos de ataques: engenharia social, roubo de senha, exploração de vulnerabilidades, negação de serviço e *man-in-the-middle*. Por último, algumas das principais proteções disponíveis atualmente para os sistemas e equipamentos exstentes: criptografia, AAA (autenticação, autorização e auditoria), segurança de *hosts*, *firewall* e IDS. Uma aprofundada maior nos sistemas de IDS, objetos diretos do estudo.

No capítulo 4, a parte teórica de redes neurais artificiais é apresentada. Após um histórico do surgimento e evolução das RNAs, através do *perceptron* e do ADALINE, o principal algoritmo de aprendizagem supervisionada é visto, o *backpropagation*. Os cálculos matemáticos não são óbvios, mas estão presentes apenas para ilustrar o funcionamento das redes. A importância maior para o trabalho está nas considerações práticas. As redes recorrentes e de aprendizagem competitiva, ou não supervisionada, também estão presentes no restante do capítulo.

O capítulo 5 tem importância vital no entendimento das motivações do trabalho. Diversas pesquisas foram realizadas no mundo todo recentemente a respeito de detecção de intrusão através de redes neurais artificiais e outros algoritmos não lineares. Algumas dessas pesquisas são apresentadas no capítulo: as configurações das redes, os tipos de dados de entrada e os resultados foram muito importantes para a maturidade da proposta. Apesar do presente trabalho tratar de um sistema de detecção em rede, alguns trabalhos sobre detecção em *hosts* também são apresentados no capítulo.

O capítulo 6 apresenta a *honeynet*, a maior inovação do trabalho em relação aos trabalhos anteriormente publicados e apresentados no capítulo anterior. Após uma apresentação do *honeynet project*, projeto precursor do mecanismo de *honeynet*, são discutidos tipos de *honeypots*, de baixa e de alta interação. Depois, os cuidados e mecanismos de uma *honeynet* de controle e captura de dados são mostrados. Por último, algumas possíveis evoluções dos *honeypots* existentes são debatidas: *honeypot farms* e *honeypot* dinâmicos.

Por fim, o capítulo 7 apresenta a proposta prática do trabalho. Mesclar todos os conceitos vistos anteriormente e criar um ambiente de treinamento para redes neurais artificiais aplicadas à detecção de intrusão utilizando uma *honeynet* como geradora de dados de entrada para treinamento. Uma *honeynet* real foi construída e alguns *softwares* foram

utilizados para coletar, tratar e treinar os pacotes de rede. Em resumo, a proposta é coletar os pacotes de rede capturados na *honeynet*, filtrá-los, normalizá-los e treinar uma rede neural artificial não supervisionada. Esta rede cria um mapa visual de representação do tráfego da *honeynet* que irá auxiliar na determinação de perfis diferentes de ataque. Com base nesses perfis, a rede neural supervisionada é treinada com o algoritmo *backpropagation*. Espera-se poder separar nesses perfis, e com ajuda de ferramentas auxiliares, tipos de ataques comuns e reconhecidamente classificados, como ataques de *scanning, denial of service* e exploração remota de vulnerabilidades. Desta maneira, por exemplo, é possível construir 3 diferentes treinamentos de redes neurais artificiais especialistas, um para cada tipo de ataque. Um sistema de detecção poderia ser construído para consultar cada uma dessas redes a cada pacote e determinar se se trata de um ataque e a qual tipo ele pertence. Ainda neste capítulo, os resultados são apresentados.

2. PROTOCOLOS DE COMUNICAÇÃO

Este capítulo apresenta os protocolos existentes na Internet que são mais pertinentes ao assunto a ser tratado e que serão vistos nos capítulos subseqüentes, com o objetivo de subsidiar teoricamente o leitor.

No auge da guerra fria, nos anos 60 e 70, o governo americano começou a investir em pesquisas para desenvolver uma rede de comutação de pacotes que conectasse os órgãos do governo de maneira mais segura do que as redes telefônicas existentes. Então a divisão científica do Pentágono, a ARPA (*Advanced Research Project Agency*) criou uma rede de pesquisas interligando Universidades e governo, a ARPANET. Pouco a pouco centenas de Universidades e repartições públicas foram sendo conectadas a ela através de linhas privadas. Depois foram criadas as redes de rádio e satélite e começaram a surgir problemas com os protocolos existentes. Foi criada, então, uma arquitetura de protocolos com o objetivo de conectar várias redes diferentes ao mesmo tempo. Essa arquitetura ficou conhecida como modelo de referência TCP/IP graças aos seus dois protocolos principais[3].

2.1 MODELO OSI E MODELO TCP/IP

O modelo OSI (*open systems interconection*) trata da interconexão de sistemas abertos a comunicação com outros sistemas, ou seja, seu modo de funcionamento é publicamente disponível para que qualquer fabricante possa desenvolver aplicações que sigam este padrão e se comuniquem com outras aplicações diferentes. A proposta do modelo OSI foi desenvolvida pela ISO (*International Standards Organization*) com o objetivo de padronizar internacionalmente os protocolos utilizados nas diversas camadas. O modelo OSI possui 7 camadas e foi concebido antes de os protocolos terem sido inventados[1,3].

Já o modelo TCP/IP, surgiu da prática, foi criado com base nos protocolos existentes da rede ARPANET. Possui 4 camadas que resumem as mesmas funcionalidades das camadas OSI.

Os dois modelos se baseiam no conceito de pilha de protocolos independentes. Abaixo da camada de transporte são definidos serviços de transporte de pacotes fim a fim,

independente do tipo de rede. Acima da camada de transporte, estão as camadas que dizem respeito aos usuários orientados à aplicação do serviço de transporte. O modelo OSI definiu claramente, seguindo a idéia de orientação a objetos, a diferença entre 3 conceitos fundamentais da relação entre as camadas: o **serviço** informa o que a camada faz; a **interface** informa como os processos acima dela podem acessá-la e especifica os parâmetros e resultados esperados; os **protocolos** são de responsabilidade de cada camada, ou seja, ela pode utilizar qualquer protocolo que realize o serviço corretamente. O modelo TCP/IP não distingue tão bem essas diferenças e foi tentando se adaptar ao modelo OSI. Isso faz o modelo OSI mais flexível e bem encapsulado, podendo ser substituído com relativa facilidade por novas tecnologias[3] . A figura 2.1 ilustra a diferença de camadas entre os modelos OSI e TCP/IP

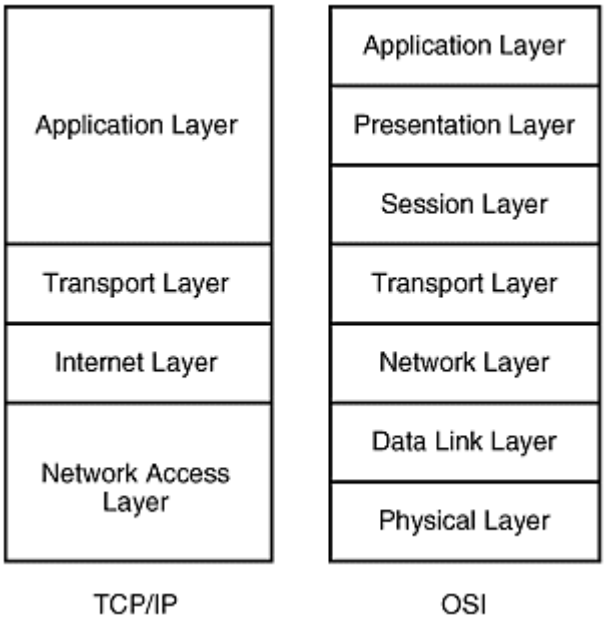


Figura 2.1 – Modelos de referência; TCP/IP x OSI

Apesar do modelo OSI ter surgido antes e ter algumas vantagens citadas, na prática ele foi deixado de lado. Isso porque o modelo TCP/IP se popularizou através da arquitetura utilizada na ARPANET, que mais tarde veio a ser chamada INTERNET.

As camadas implementadas pela arquitetura TCP/IP são (de baixo para cima): camada de interface com a rede, camada Internet, camada de transporte e camada de aplicação.

A arquitetura funciona mais ou menos da seguinte maneira: supõe-se uma comunicação entre dois *hosts* através de uma rede qualquer. As camadas do primeiro *host* irão

“conversar” com as camadas correspondentes do segundo *host.* Quer dizer, a camada de Internet do primeiro irá, por exemplo, endereçar o pacote para que a camada Internet do segundo receba, entenda e saiba pelo endereço quem mandou o pacote. A camada de transporte do primeiro *host* através de *flags* irá controlar a conexão em conjunto com a camada de transporte do segundo. No que diz respeito à interface entre as camadas dentro do próprio *host*, o fluxo funciona através do conceito de encapsulamento. Os dados em si são gerados em cima da pilha de protocolos e são encapsulados pela camada de aplicação, isto é, a camada de aplicação insere antes dos dados um *cabeçalho* que servirá justamente para comunicar-se com a camada de aplicação de outro *host* na rede. A camada de aplicação entrega, então, o pacote para a camada abaixo, de transporte. Esta vai encapsular os dados recebidos com seu próprio cabeçalho e entregar para a camada de baixo, que fará o mesmo. Até que, na camada de interface com a rede, após inserido o último cabeçalho o pacote vai para o meio de comunicação. Chegando no *host* de destino, o pacote será desencapsulado pela camada de interface com a rede (tirará o cabeçalho colocado pelo *host* de origem) e entregue à camada superior. Todas as camadas farão um processo semelhante até que na camada de aplicação o último cabeçalho será retirado e os dados estarão lá. Cada cabeçalho que foi retirado provê à camada que o retirou informações que ela precisa saber para responder no próximo pacote. Tudo isso ficará mais claro nos itens a seguir do capítulo.

2.2 CAMADA DE INTERFACE COM A REDE

Como o próprio nome sugere, é a camada que inclui os *drivers* dos dispositivos no sistema operacional e a sua correspondente placa de rede no computador. É responsável por prover todos os detalhes de conexão física e dos meios de comunicação com a rede. No modelo de referência TCP/IP nada é especificado sobre protocolos desta camada. Apenas se diz que o *host* tem que se conectar com a rede utilizando algum tipo de protocolo, para que seja possível enviar pacotes IP. Seus protocolos são os mais diversos: Ethernet, Token Ring, PPP, FDDI, X.25, L2TP, SLIP, *frame relay*, etc. No caso de redes locais LAN, o protocolo mais utilizado é o ethernet. Para conexão com a Internet, diversos tipos de combinação de protocolos podem ser utilizados.

2.3 CAMADA INTERNET

Está relacionada à transferência de pacotes da origem para o destino. Para que o pacote chegue ao seu destino, às vezes, é necessário que passe por diversos dispositivos no meio do caminho. Dispositivos que tem a função de rotear os pacotes, ou seja, mandá-los para o caminho mais curto ou menos demorado na direção do seu destino final. Cada dispositivo desses por onde passa o pacote é considerado um *hop* (passo em direção ao destino). Para que os roteadores e *hosts* saibam de onde vem e para onde vai o pacote, é necessário que ele tenha um endereço de origem e um de destino, chamados de endereçamento IP. Por fim, é uma camada que não se preocupa com a conexão, apenas entrega os datagramas sem se preocupar se estão chegando ou não, ou se chegam na ordem correta, deixando esta tarefa para a camada superior, de transporte.

Essa camada inclui protocolos de roteamento como RIP, OSPF, BGP que implementam complexos algoritmos de roteamento, controle de congestionamento e moldagem de tráfego e que estão fora do escopo do estudo. O que interessa saber é que esses roteadores recebem os pacotes, desencapsulam até a camada Internet e a partir de seus algoritmos e tabelas entregam o pacote ao próximo *hop* ou ao destino, se for o caso.

A camada inclui também protocolos que serão melhores detalhados como ARP, ICMP e IP.

2.3.1 ENDEREÇAMENTO IP

Um sistema de comunicação para ser um serviço universal, precisa permitir que qualquer computador possa se comunicar com qualquer outro em qualquer lugar. Portanto, o método para identificação de cada computador precisa ser globalmente aceito. Frequentemente identificadores de *hosts* são classificados como nomes, endereços ou rotas. O nome indica quem é o objeto. O endereço, onde ele está. E a rota, como chegar a ele [1].

Na arquitetura TCP/IP, os endereços são providos pelo protocolo IP. A partir deles é possível criar rotas e mapear nomes humanamente compreensíveis.

Os endereços IP escolhidos como padrão para a Internet são endereços de 32 *bits* que

identificam cada máquina conectada à rede. A figura 2.2 ilustra a divisão criada nestes 32 bits para atender a diferentes classes de endereçamento. A classe A é destinada normalmente a grandes empresas que possuem muitos *hosts*. A quantidade de redes que podem possuir este endereço é pequena: 128 (7 bits destinados à rede ou 2^7 endereços). Porém a quantidade de *hosts* dentro dessas redes é enorme: mais de 16 milhões de máquinas (24 bits). A representação decimal do endereço é feita de 8 em 8 bits separados através de “.”. Por exemplo, a classe A tem uma faixa de endereços que vai de 1.0.0.0 a 127.255.255.255. A classe B possui 14 bits destinados ao endereçamento de rede e 16 destinados ao endereçamento de *hosts*. Seu intervalo é de 128.0.0.0 a 191.255.255.255. A classe C possui 21 bits destinados ao endereçamento de rede e 8 bits destinados ao endereçamento de *hosts*. É o mais utilizado por provedores e empresas de pequeno e médio porte na Internet, pois permite endereçar mais de 2 milhões de redes diferentes, cada uma com 254 *hosts*. A classe D é inteiramente reservada a endereços do tipo *multicast*, onde um datagrama é destinado a vários *hosts* e vai de 224.0.0.0 a 239.255.255.255. A classe E, que não é mostrada na figura foi reservada para uso futuro e vai de 240.0.0.0 a 247.255.255.255.

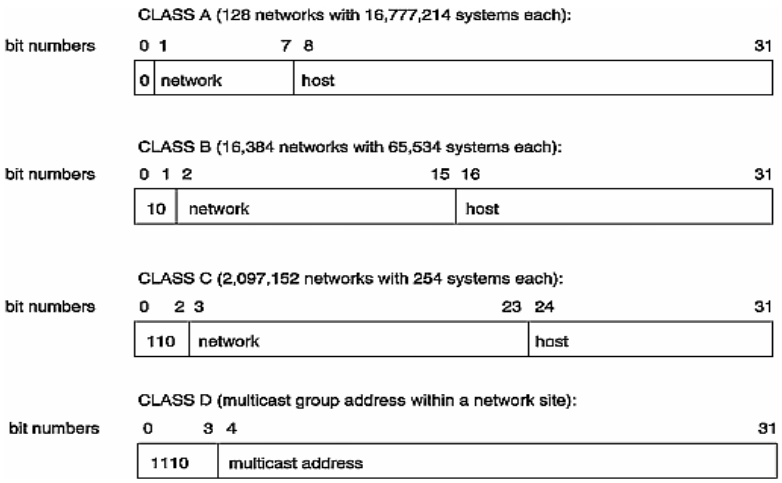


Figura 2.2 – Classes de endereçamento IP

Apesar do número de endereços possíveis com 32 bits (2^{32}) ser muito grande, com o aumento exponencial de usuários conectados a Internet, esse endereçamento passou a ser escasso. Para resolver este problema e outros como a falta de preocupação com segurança do protocolo IP vigente (versão 4), o protocolo IPV6 (versão 6) há muito está pronto para entrar em produção. Entretanto, a mudança de hardware e software para suportar o novo protocolo,

com endereçamento quase que ilimitado seria muito grande. Além disso uma solução muito boa foi encontrada para a solução de endereços escassos. O endereço classe A 127.x.x.x é reservado para teste de *loopback*. Além disso, os endereços classe A 10.0.0.0 a 10.255.255.255, classe B 172.16.0.0 a 172.31.255.255 e classe C 192.168.0.0 a 192.168.255.255 são reservados para NAT(*network address translation*, RFC 1631) e não são válidos na Internet. Isto quer dizer que só precisam ter um endereço válido na Internet, os servidores de aplicação(como servidores HTTP, SMTP) e *gateways* de rede(*firewall*, roteadores). Os *hosts* de usuários ficam atrás desses equipamentos, utilizando uma faixa de endereços reservados, ou falsos. Desta maneira, eles se "protegem" da Internet e acessam seus serviços por meio destes servidores, ou roteadores que realizam a tradução do endereço de muitos *hosts* para um ou poucos endereços válidos na Internet[2].

Além desses aspectos, há também um parâmetro essencial no endereçamento de redes que se chama máscara de rede. Por meio desta máscara, é possível dividir um endereçamento de qualquer classe, criando sub-redes diferentes e dividindo de fato as classes. Por exemplo, um endereçamento reservado para classe C 192.168.254.0 a 192.168.254.255 terá que ter a máscara de rede 255.255.255.0, ou seja, os primeiros 3 conjuntos de 8 bits estão com o valor 255, ou todos os bits com valor 1 e não há espaço para mudança no endereçamento 192.168.254. O último conjunto está com o valor 0, quer dizer que o endereçamento pode variar de 0 a 255 (na prática de 1 a 254). É possível dividir ainda mais esta rede, colocando, por exemplo, o valor 128 no último conjunto. Dessa maneira, o valor binário do último conjunto seria 10000000, os 7 últimos números podendo variar, ou seja, podem ser criadas duas sub-redes dentro da classe C, uma de 0 a 127 e outra de 128 a 254. Assim as sub-redes podem ir aumentando se o número de 1s for aumentando na máscara e quanto mais sub-redes menos *hosts* em cada uma delas, até que, no limite da divisão, a máscara 255.255.255.255 representa o próprio *host*.

2.3.2 PROTOCOLO ARP

O protocolo ARP (A*ddress Resolution Protocol*, RFC 826) é responsável pela interface entre camada Internet e camada de interface com a rede. Isso porque a camada física, ou o *hardware* de rede não compreende um endereçamento do tipo IP, definido anteriormente.

Existe um outro tipo de endereçamento de 48 bits nas placas dos fabricantes de *hardware*. Com isso tem que existir um protocolo que traduza de um endereço para o outro para que as camadas de interface com a rede possam se comunicar e as camadas superiores não tenham que se preocupar com endereço físico. Essa tradução é feita através do protocolo ARP. Quando uma máquina com um IP A precisa se comunicar com uma máquina com IP B ela solta um pacote de difusão na rede que pergunta: “A quem pertence o endereço IP B? Responda para IP A”. A máquina de destino responde “O IP B está no endereço físico X”. Com isso o ARP mantém nos *hosts* uma tabela com o mapeamento dos últimos endereços que ela precisou se comunicar ou que ela teve acesso através das inúmeras difusões na rede. Essa tabela tem um certo *timeout*, pois os IPs vez ou outra trocam de máquina e os endereços físicos são fixos.

O RARP (*Reverse Address Resolution Protocol*, RFC 903) permite que uma estação recém-inicializada transmita seu endereço físico e diga: “Meu endereço físico de 48 *bits* é X, alguém sabe meu IP?” O servidor RARP vê a solicitação, procura um endereço em seus arquivos e retorna com o IP correspondente[3]. Isso normalmente era utilizado quando uma máquina sem sistema operacional era colocada na rede para ser instalada. O RARP tem uma limitação, porque utiliza um endereço de destino composto somente de 1s para chegar no servidor RARP. Com isso é necessário que exista um servidor deste tipo em cada rede porque os roteadores não conseguem repassar a requisição. Para solucionar, foi criado o protocolo BOOTP(RFC 951) que utiliza o protocolo de transporte UDP para isso[3]. Como evolução para o BOOTP, foi criado o protocolo DHCP(RFC 2131), que em um único pacote UDP envia todas as informações necessárias para a inicialização da máquina.

2.3.3 PROTOCOLO IP

Internet Protocol (IP) é um dos pilares da arquitetura TCP/IP. Pode-se dizer que o IP, no fundo, é uma abstração da camada física, pois ele provê a mesma funcionalidade: aceitar e entregar pacotes (chamados de datagramas). É o protocolo que provê a base dos três serviços principais da arquitetura: serviço de aplicação; serviço de transporte; serviço de entrega de pacotes sem conexão[1]. O IP, portanto, é encarregado da entrega dos datagramas, sem garantir se estes irão chegar no destino, se irão se atrasar, sair da ordem, duplicar. Apenas usa

a lei do melhor esforço, tratando os datagramas de maneira independente.

De maneira geral, o datagrama IP é muito parecido com um pacote físico, pois possui um cabeçalho e um campo de dados. Neste cabeçalho, possui endereço de origem e endereço de destino. A figura 2.3 ilustra o datagrama IP. Abaixo cada campo será detalhado.

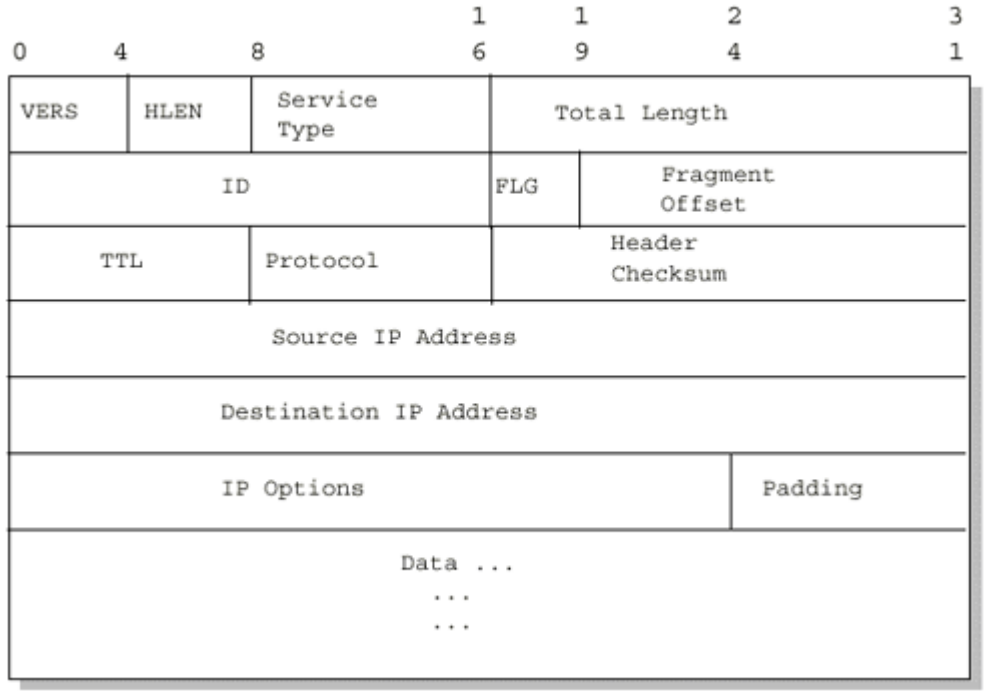


Figura 2.3 – Datagrama IP

a) *VERS* (4 bits): campo de versão do protocolo IP, atualmente na versão 4;

b) *HLEN* (4 bits): campo de tamanho do cabeçalho, medido de 32 em 32 bits. na verdade todos os campos tem tamanhos fixos, exceto o IP Options e Padding. Se o cabeçalho não conter opções a medida é de 20 octetos, ou HLEN=5;

c) *Service Type* (8 bits): campo chamado de TOS (*Type of Service)* especifica como o datagrama deve ser entregue. Os 3 primeiros bits são o campo PRECEDENCE , que tem uma prioridade variando de 0 (datagrama normal) a 7 (controle de rede). Depois vêm 3 bits de *flags* (0 – desligado; 1 – ligado): D (retardo), T (taxa de transferência) e R (confiabilidade) e depois 2 bits que não são utilizados. Na prática, os roteadores ignoram este campo TOS. Se todos implementassem as suas funcionalidades, poderia ser possível implementar um controle de congestionamento, os roteadores poderiam escolher a prioridade de tráfego de acordo com o tipo de aplicação: videoconferência, transferência de arquivo, tráfego de controle, etc.

d) *Total Lenght* (16 bits): Tamanho total do datagrama IP (cabeçalho e dados), medido em

octetos. A limitação de tamanho é 2^{16} octetos ou 64KB;

Os próximos itens dizem respeito à fragmentação de datagramas. Isso acontece porque o tamanho do pacote físico (quadro) pode variar dependendo de sua tecnologia. Um datagrama IP muito grande poderia não caber em quadro físico pequeno e um datagrama IP, se fosse muito pequeno, poderia não aproveitar tecnologias que possuem quadros grandes. Portanto o protocolo IP deixa a cargo do software escolher qual será o tamanho dos fragmentos. Apenas no destino final esses fragmentos são reconstruídos (duas desvantagens aqui: havendo a perda de um fragmento, todo o datagrama será descartado e se no meio do caminho houver tecnologias que possibilitem quadros maiores que a fragmentação original, não será aproveitado). Todos os fragmentos de um datagrama utilizam os mesmos campos do cabeçalho, com exceção dos campos FLAGS, Fragment Offset e Total length.

e) *ID* (16 bits): Número inteiro único que identifica o datagrama. Todos os fragmentos daquele datagrama terão o mesmo ID. Com isso, o destino poderá reconstruir o datagrama.

f) *FLAGS* (3 bits): O primeiro bit não é utilizado. O segundo bit é o *flag* DF que significa não fragmente. É uma ordem explícita para os roteadores não fragmentarem o datagrama, pois a máquina de destino é incapaz de juntar os pedaços. O terceiro é o MF que significa mais fragmentos. É necessário e utilizado para a máquina de destino saber que o datagrama ainda não chegou ao fim. Apenas o último fragmento do datagrama virá com o bit MF desabilitado. O campo Total Length de cada fragmento contém o tamanho total do fragmento e não de todo o datagrama.

g) *Fragment Offset* (13 bits): Utilizado para informar ao destino a que ponto do datagrama o fragmento pertence. Ele é medido em unidades de 8 octetos e contém a informação de onde começa o fragmento. Utilizando essa informação, o Total Length de cada fragmento e o *flag* MF para saber o último fragmento, o destino consegue computar o tamanho total do datagrama.

h) *TTL* (8 bits): o *time to live* representa o tempo que o pacote pode permanecer na Internet. Quando o pacote sai da origem o TTL é setado. A cada *hop* que ele passa, o roteador decrementa em uma unidade. Quando o valor chega em 0, o roteador descarta o pacote e manda um pacote de advertência para a origem. É um recurso que evita que os datagramas fiquem vagando indefinidamente.

i) *Protocol* (8 bits): É o campo que especifica o tipo de protocolo da camada superior (transporte) está sendo utilizado. A RFC 1700 especifica a numeração desses protocolos

j) *Header Checksum* (16 bits): É uma checagem feita nos dados do cabeçalho a cada 16 bits, através do algoritmo complemento-de-um, com o objetivo de determinar se algum dado foi alterado. A cada *hop* o *checksum* tem que ser calculado novamente pois o campo TTL está sempre mudando.

k) *Source IP Address* e *Destination IP Address* (32 bits + 32 bits): São os endereços IPs de origem e de destino vistos anteriormente. Determinam onde o pacote foi originado e seu destino final na rede.

l) *IP Options* (tamanho variável): São opções do datagrama IP. Não são obrigatórias e são utilizadas principalmente para teste. O tamanho varia de acordo com as opções selecionadas. O primeiro campo de opções é o campo Option Code, formado por um octeto dividido no bit Copy, dois bits de Option Class (classes) e 5 bits de Option Number (número). Existem 4 classes de opções, mas apenas 2 delas são utilizadas: 0 e 2. Na classe 0 existem 7 números, mas os mais interessantes na prática são 3, 7 e 9. O 7 é a opção *record route*, que serve para gravar, ao longo do caminho do datagrama todos os endereços IPs dos *hops*. Estes endereços vão sendo anexados ao campo de opções. A 3 e a 9 são a *loose source routing* e *strict source routing* e tem basicamente a mesma função: a origem grava no campo Option os IPs por onde o pacote deve passar antes de chegar no destino. A diferença é que no *strict* os IPs devem ser exatamente aqueles designados, na mesma ordem, sem nenhum outro no meio do caminho. O *loose* é mais flexível e permite que entre os IPs designados hajam outros no meio do caminho. Ainda assim tem que passar pelos designados. A classe 2 possui apenas um número, o 4, que é a opção *timestamp*, semelhante à opção *record route*, porém após a gravação do IP a cada *hop*, o roteador também grava o *timestamp*, que é o formato de data e hora. Algumas dessas opções necessitam que todos os fragmentos contenham o campo de opções (por exemplo as opções *source routing*, que escolhem explicitamente a rota para o datagrama*)*, neste caso o bit Copy será 1. Caso contrário, onde apenas um fragmento precisa ter o campo IP Options (caso do *record route*), o campo Copy será 0.

m) *Padding* (variável): o campo *padding* não contém nenhuma informação, apenas "enchimento" do datagrama IP. Ele é utilizado para completar o cabeçalho do datagrama quando houver opções, para que este volte a ser múltiplo de 32.

2.3.4 PROTOCOLO ICMP

O ICMP (*Internet Control Message Protocol*, RFC 792) é um protocolo utilizado para reportar falhas de entrega de datagrama, erros de conexão, *timeout* de protocolos, mensagens de controle, teste da Internet, etc. No caso de um erro no datagrama, o ICMP apenas é capaz de reportar esse erro de volta para a origem. Se o erro ocorrer em algum roteador intermediário não será possível reportá-lo do erro, pois não se sabe por onde esse pacote trafegou. A origem é responsável por tomar as providências a partir do erro reportado [1].

Como o datagrama IP, a mensagem ICMP possui um cabeçalho e um campo de dados. Porém, apesar de pertencer à mesma camada Internet, ele precisa ser encapsulado em um pacote IP (na área de dados) antes de passar à camada física e dali ser mandado à Internet. Quer dizer, ele utiliza as características de entrega de datagrama do IP para chegar ao seu destino.

O formato do ICMP é variável de acordo com o tipo. os três primeiros campos sempre são os mesmos: *type, code* e *checksum*. O *checksum* tem 16 bits e é calculado da mesma maneira do protocolo IP. O *type* tem 8 bits e identifica a mensagem. O *code* tem 8 bits e provê informações sobre o tipo de mensagem. Além disso, o ICMP que reporta erros sempre inclui o cabeçalho e os primeiros 64 bits do datagrama que causou o erro, assim a origem pode investigar melhor qual protocolo e qual aplicação causou o erro informado.

A tabela 2.1 mostra os tipos possíveis das mensagens ICMP.

Tipo	Mensagem
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect (change a route)
8	Echo Request
11	Time Exceeded for a Datagram
12	Parameter Problem on a Datagram
13	Timestamp Request
14	Timestamp Reply
15	Infomation Request (obsoleto)

Tipo	Mensagem
16	Information Reply (obsoleto)
17	Address Mask Request
18	Address Mask Reply

Tabela 2.1 – ICMP type (modificado – [1])

O recurso mais utilizado do protocolo ICMP são os tipos *echo request* e *echo reply*. São utilizados por um programa chamado *ping*. A máquina de origem transmite um *request* e espera até a máquina destino o receba e devolva um *reply*. Qualquer máquina destino que recebe um *request*, manda de volta um *reply*. Com isso, a conectividade entre as duas máquinas foi testada, ou seja, elas estão funcionando e roteando corretamente, os roteadores entre elas estão roteando corretamente e os protocolos ICMP e IP estão funcionando. A figura 2.4 mostra um exemplo de uma mensagem ICMP Echo.

Outro recurso importante é a mensagem destino inacessível (*unreachable destination*), utilizada quando a sub-rede ou o roteador não pode localizar o destino. Para este tipo de erro existem 13 códigos diferentes: *network unreachable, host unreachable, protocol unreachable, port unreachable, fragmentation needed and DF set, source route failed, destination network unknown, destination host unknown, source host isolated, communication with destination network administratively prohibited, communication with destination host administratively prohibited, network unreachable for type of service* e *host unreachable for type of service*. Rede inacessível normalmente implica em falhas de roteamento, *host* inacessível normalmente é falha de entrega. Os erros podem acontecer porque o destino não existe, ou porque o *hardware* está indisponível e muitos outros motivos. Pode ocorrer também do pacote IP estar com o FLAG DF setado e a rede só aceitar pacotes menores. O código *fragmentation needed* é setado. Resumindo a mensagem de destino inacessível ocorre quando o datagrama não pode ser entregue por algum motivo. A figura 2.4 mostra um exemplo desta mensagem ICMP.

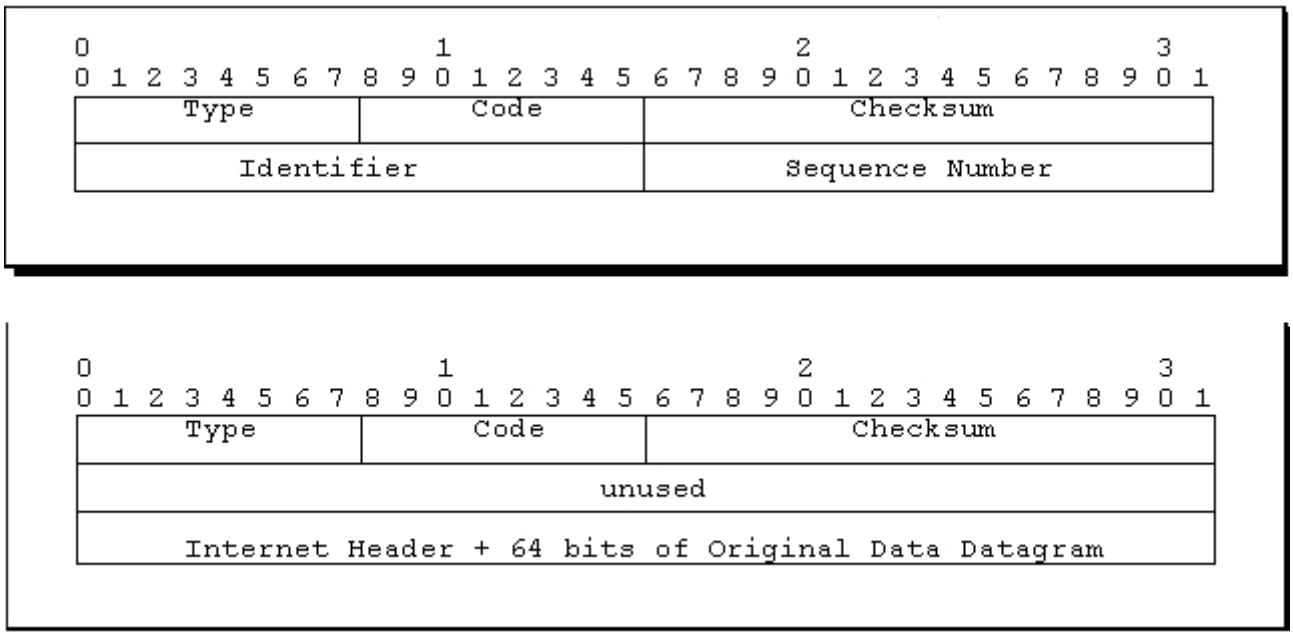


Figura 2.4 – Exemplos de mensagens ICMP; Echo e Unreachable Destination

O ICMP tipo *source quench* serve para os roteadores avisarem à origem que a taxa de transmissão de datagramas está muito alta e está causando congestionamento no roteador. A idéia é que quando um roteador descartasse um pacote por estar congestionado, ele mandasse essa mensagem ICMP à origem. Porém, raramente é utilizado, pois o tráfego já é muito alto. A camada de transporte fica encarregada deste controle de congestionamento.

O tipo *time exceeded* é enviado quando o pacote é descartado porque seu contador chegou a zero. Ele possui dois códigos: *time-to-live count exceded* quando o campo TTL do IP chega a 0 e *fragment reassembly time exceeded* quando o tempo de espera para receber determinado datagrama de um fragmento expira.

Os tipos *timestamp request* e *timestamp reply* realizam a mesma tarefa do *Echo*, porém registram o tempo de chegada da mensagem e o tempo de partida da resposta, podendo ser utilizados para medir o desempenho da rede.

Os tipos *address mask request* e *reply* servem para uma máquina conhecer a máscara de rede de outra máquina.

O tipo *redirect* é enviado ao transmissor quando um roteador muda sua tabela de roteamento e percebe que há uma rota melhor para um determinado pacote.

Finalmente, o tipo *parameter problem* é utilizado quando existem outros tipos de problemas no datagrama que os outros tipos de mensagem não cobrem, por exemplo um valor inválido em algum campo do cabeçalho IP.

2.4 CAMADA DE TRANSPORTE

A camada de transporte é o coração da arquitetura TCP/IP. Sua função é promover uma transferência de dados confiável e econômica entre a máquina de origem e a de destino, independente das redes físicas em uso no momento[3]. Ela aceita os dados da camada de aplicação, divide em unidades menores, se necessário, repassando para a camada Internet. Do outro lado, a camada de transporte do receptor remonta as mensagens. Além disso, garante que essas unidades chegarão corretamente ao destino. É o verdadeiro protocolo fim a fim, onde origem e destino trocam mensagens e controlam o fluxo de mensagens. Nas camadas inferiores, os protocolos são trocados entre máquinas vizinhas, não entre origem e destino. Dois protocolos são definidos nesta camada: o protocolo TCP, orientado à conexão, permitindo uma conexão segura, livre de erros e controlando o fluxo para impedir sobrecarga de um receptor lento; e o UDP, sem conexão, não confiável para aplicações que não necessitam de controle de fluxo nem de manutenção da seqüência das mensagens enviadas. A entrega imediata é mais importante que a entrega precisa. Os dois protocolos serão melhor detalhados adiante.

2.4.1 PROTOCOLO UDP

O UDP (*User Datagram Protocol*, RFC 768) provê um serviço de entrega de pacotes baseado em uma conexão não confiável, utilizando o protocolo IP para transportar as mensagens. Ele adiciona ao IP a habilidade de distinguir entre muitos destinos no mesmo *host* [1]. Cada pacote UDP contém em seu cabeçalho um número da porta de origem e número da porta de destino. Essas portas diferenciam os múltiplos programas ou processos de rede que rodam numa mesma máquina. O UDP não utiliza nenhum tipo de mecanismo de controle das mensagens que manda, podendo essas serem perdidas, duplicadas ou chegarem fora de ordem, assim como os datagramas IP.

Assim como os outros protocolos, UDP possui um cabeçalho e uma parte de dados. O cabeçalho é dividido em 4 campos de 16 bits. A figura 2.5 ilustra um datagrama UDP.

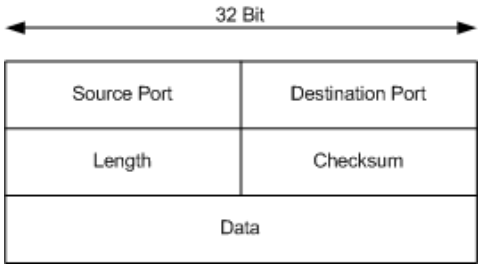


Figura 2.5 – Pacote UDP

Os campos *Source Port* e *Destination Port* contém os números de porta utilizados pelo UDP para entregar o datagrama ao processo receptor. O campo *source port*, ou porta de origem é opcional e especifica a porta na qual a resposta deve ser enviada.

O campo *length* corresponde ao tamanho em octetos de todo o datagrama UDP, incluindo cabeçalho e dados. O valor mínimo do campo é 8 (só o cabeçalho).

O campo *Checksum* corresponde à checagem semelhante ao IP. É opcional no UDP e quando não há, o valor do campo é 0. Quando o UDP utiliza o *checksum*, ele acrescenta um pseudo-cabeçalho de 12 octetos ao datagrama, contendo os valores de IP origem e destino, protocolo e tamanho, além de completar o restante com zeros para calcular o *checksum*. No entanto, ele não manda os zeros. No destino, a máquina retira os mesmos dados, remonta o pseudo-cabeçalho e verifica o *checksum*. Essa estratégia ajuda a detectar pacotes extraviados, no entanto viola a hierarquia do protocolo, pois as informações de endereçamento IP deveriam estar na camada IP e não na camada UDP.

O número da porta de cada aplicação segue uma regra híbrida na arquitetura TCP/IP. As portas de valores baixos (0 – 1023) são classificadas através de uma autoridade central (IANA), que entrega esses valores para as aplicações mais importantes existentes. As abaixo de 256 são as portas conhecidas e reservadas para os serviços-padrão. São definidas na RFC 1700. As aplicações menores, ou proprietárias podem utilizar a porta que for mais conveniente de valor alto. De 1024 a 49151 elas são registradas no IANA. De 49152 a 65535 são dinâmicas.

2.4.2 PROTOCOLO TCP

O TCP (*Transmission Control Protocol*, RFC 793, 1122, 1323) é o protocolo mais

completo e complexo da família TCP/IP. Ele foi projetado para oferecer um fluxo de bytes fim a fim confiável em uma rede não-confiável. O TCP existe como protocolo de comunicação e como processo, ou parte do kernel da máquina do usuário. Ele recebe informações da camada superior, ou seja, as aplicações, divide essas informações em pedaços de até 64KB e os repassa para a camada IP. Na prática, costuma dividir em segmentos de 1,5KB. Ele também gerencia temporizadores para retransmissão no caso de perda de pacotes e faz a reorganização dos segmentos. Em outras palavras, ele oferece confiabilidade à conexão[1,2,3,4].

As conexões estabelecidas pelo protocolo TCP são todas *full-duplex*, o tráfego acontece em duas direções e ao mesmo tempo. E são conexões ponto-a-ponto, isto é, existe um ponto terminal, ou *socket* (identificado por endereço IP mais porta) de um lado e um *socket* do outro lado estabelecendo a conexão.

O TCP é formado por um cabeçalho fixo de 20 octetos e uma parte opcional. A seguir vem a parte de dados que pode ser nula, ou não. O tamanho máximo de um segmento TCP é de 64KB, que constitui a parte de dados do protocolo IP. Dependendo do software que o implementa e do tamanho do quadro físico que transporta os pacotes na rede, o TCP pode encher seu *buffer* até que a quantidade de dados chegue a um certo tamanho e transmitir ou segmentar uma quantidade que considera grande e transmitir em múltiplos segmentos (o que aumenta o tamanho total da mensagem, pois assim como no IP, o cabeçalho será repetido em cada segmento).

A figura 2.6 mostra o segmento TCP. À medida que os campos forem sendo explicados, alguns conceitos ficarão claros.

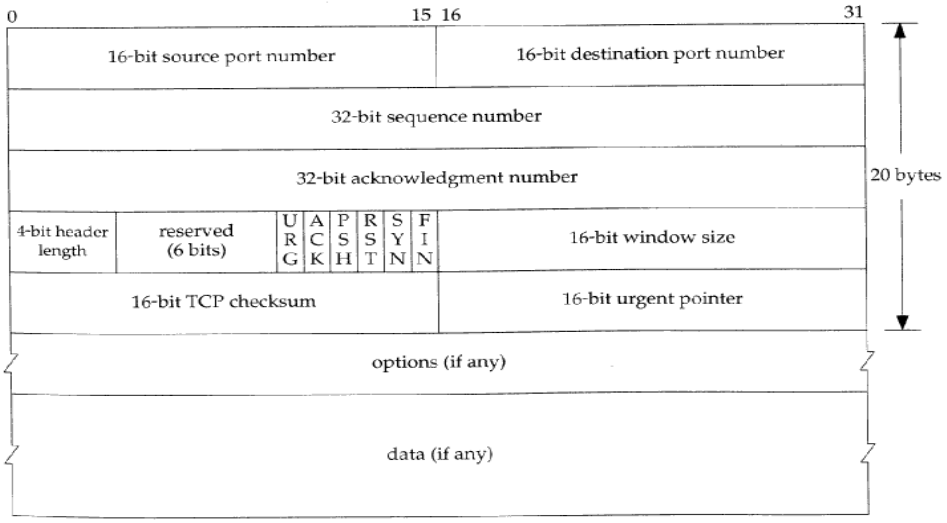


Figura 2.6 – Segmento TCP (Stevens, 2000)

O cabeçalho TCP é formado pelos seguintes campos:

a) *Source Port Number* e *Destination Port Number* (16 bits + 16 bits): Assim como no protocolo UDP, são números que identificam um processo dentro da máquina. Juntamente com o endereço IP, formam os *sockets* que caracterizam a comunicação TCP fim a fim. Cada máquina decide como alocar suas portas, respeitando a numeração do IANA (ver UDP).

b) *Sequence Number* (32 bits): É o campo que identifica o byte que representa a posição do início dos dados do segmento. Cada byte do segmento tem seu próprio número de seqüência de 32 bits.

c) *Acknowledgement number* (32 bits): Representa a posição do próximo octeto aguardado pelo transmissor, ou seja, o número de seqüência do último octeto recebido com sucesso mais um. Só é ativado quando o valor do *flag ACK* é 1.

Para entender melhor os campos anteriores, seria interessante mostrar como uma conexão é estabelecida. A figura 2.7 esclarece o estabelecimento do *three way handshake*. O primeiro segmento TCP é mandado com o *flag SYN* setado (ver adiante) para o estabelecimento da conexão. A máquina escolhe um ISN (*Initial Sequence Number*) para o segmento de acordo com o seu contador e popula o campo S*equence Number*. O campo *Acknowledgement Number* fica desativado. *O* destino recebe o segmento e, caso esteja no estado *LISTEN* (adiante será vista a máquina de estados), devolve um segmento com o *flag SYN* para completar o estabelecimento da conexão. Aproveitando a "carona", o *flag ACK* de confirmação (ver adiante) do segmento anterior é ativado e o campo *Acknowledgement number* é preenchido com o valor do *Sequence Number* recebido mais 1. Isso porque, apesar

de não ser um dado do segmento, o flag *SYN* "gasta" um *Sequence Number*. O transmissor original ao receber a confirmação e a solicitação de conexão do outro lado, ainda precisa enviar um último segmento com o *flag ACK* ativado e o *Acknowledgement number* esperado pelo receptor.

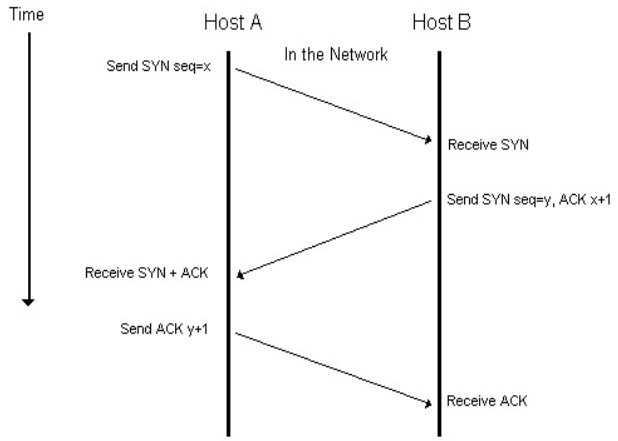


Figura 2.7 – Estabelecimento da conexão

d) *header length* (4 bits): Assim como no IP, é medido a cada 32 bits e informa o tamanho do cabeçalho TCP. Isso porque o campo *Options* é variável. Em seguida há um campo de 6 bits não utilizado.

e) *Flags* (6 bits): Os flags são parte essencial do cabeçalho TCP. Existem 6 *flags*:

-URG (Urgent Pointer) – flag que avisa ao TCP para parar de acumular dados no buffer e procurar os segmentos urgentes, que têm prioridade sobre quaisquer outros. É utilizado por exemplo quando um usuário utiliza o CTRL-C para interromper um processo remoto já iniciado;

-*ACK – flag* que indica que o campo *Acknowledgement number* é válido. utilizado para confirmar os segmentos recebidos (o TCP espera por uma confirmação de todos os segmentos que manda);

-*PSH – flag* que avisa ao receptor que ele deve entregar os dados que tem no *buffer* o mais rapidamente possível à aplicação, ou seja, a aplicação precisa que os dados sejam enviados imediatamente, não pode esperar;

-R*ST – flag* utilizado para reiniciar uma conexão. Também utilizado para rejeitar um segmento inválido ou recusar uma tentativa de conexão;

-*SYN – flag* utilizado para estabelecer conexão. Quando utilizado com *ACK=0* quer dizer conexão requerida e quando utilizado com o *ACK=1* quer dizer conexão aceita;

-*FIN – flag* utilizado para encerrar uma conexão. Assim como o *SYN*, ele também utiliza um *Sequence number.*

f) window size (16 bits): O controle de fluxo do TCP é gerenciado por meio do algoritmo de sliding window, ou janela deslizante, de tamanho variável. Normalmente o TCP divide os dados que recebe em seqüência de octetos chamados segmentos e os transmite cada um em um datagrama IP. O mecanismo de janela deslizante permite que diversos desses segmentos sejam mandados antes que chegue uma confirmação (ACK) dos anteriores. Isso mantém a rede ocupada e permite que o receptor receba pacotes até que o seu buffer esteja cheio. O mecanismo opera em nível de bytes e não de segmentos. A figura 2.8 ilustra um exemplo de janela deslizante. O tamanho da janela é de 4 bytes. Em (a), os quatro primeiros foram mandados e nenhum ACK de confirmação foi recebido. Em (b), dois segmentos ACK foram recebidos e dois continuam sendo aguardados. Porém a janela deslizou dois bytes e mandou mais dois. Agora aguarda os dois anteriores de confirmação e mais dois. Em (c), as quatro confirmações chegaram e a janela deslizou em 4 bytes.

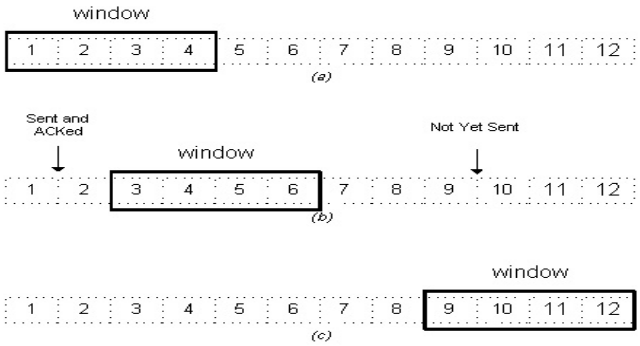


Figura 2.8 – Exemplo de *sliding window* ou janela deslizante

A grande diferença do exemplo para o TCP é que no último o mecanismo de janela deslizante tem tamanho variável, ou seja, um algoritmo é utilizado para controlar o tamanho. Na verdade, o tamanho da janela vai de acordo com o *buffer* do receptor e o receptor anuncia esse *buffer* através do campo *window size*. Se, por exemplo, o *buffer* do receptor for de 5K e o transmissor envia um segmento de 3K. O receptor irá receber o segmento e, enquanto a aplicação não retirar aqueles 3K de lá, o *buffer* terá apenas 2K livres. O receptor envia então o segmento de confirmação, com o valor de 2K no campo *window size*. Supondo que o transmissor tenha mais de 2K para enviar, ele só enviará 2K e encherá o *buffer* de transmissão dele à espera de resposta do receptor. Se o receptor receber aqueles 2K e a aplicação ainda não tiver retirado os 3K anteriores, no pacote de confirmação enviado ao transmissor, o campo

window size terá valor 0. Sendo assim o transmissor terá que esperar até que o receptor mande outro pacote semelhante, com o valor de *window* alterado. Há duas situações que o transmissor é habilitado a mandar um segmento, mesmo que o valor de *window size* seja 0: um segmento com o *flag* URG permitindo que o usuário aborte o processo; um segmento de 1 byte, para avisar ao receptor reanunciar o pacote com o tamanho do *buffer*, pois há a possibilidade dele ter se perdido na rede.

g) *Checksum* (16 bits): O campo *checksum* no TCP é obrigatório e é calculado exatamente da mesma maneira que o UDP, utilizando o pseudo-cabeçalho.

h) *Urgent Pointer* (16 bits): Quando o *flag URG* é setado, significa que este campo está habilitado. Ele é usado para indicar um deslocamento de bit no *Sequence Number* no qual os dados urgentes deverão estar, substituindo os segmentos interrompidos (que terão outro número de seqüência).

i) *Options* (variável): A opção mais importante do TCP permite que cada máquina especifique o MMS (*maximum segment size*), ou tamanho máximo de segmento que está disposto a receber. O menor deles será escolhido como padrão da conexão.

É impossível entender o protocolo TCP sem passar pela sua máquina de estados. A tabela 2.2 ilustra os possíveis estados do TCP.

Estado	Descrição
CLOSED	Nenhuma conexão está ativa ou pendente
LISTEN	O servidor está esperando pela chegada de uma chamada
SYN RCVD	Uma solicitação de conexão chegou; espera por ACK
SYN SENT	A aplicação começou a abrir conexão
ESTABLISHED	O estado normal para a transferência de dados
FIN WAIT 1	A aplicação disse que acabou de transmitir
FIN WAIT 2	O outro lado concordou em encerrar
TIMED WAIT	Aguarda a entrega de todos os pacotes
CLOSING	Ambos os lados tentaram encerrar a transmissão simultaneamente
CLOSE WAIT	O outro lado deu inicio a um encerramento
LAST ACK	Aguarda a entrega de todos os pacotes

Tabela 2.2 – Máquina de Estados TCP [3]

As conexões iniciam no estado *CLOSED*. Dependendo da situação, ela sai desse

estado para executar uma abertura passiva (*LISTEN*), ou ativa (*SYN SENT*). Quando a conexão estiver estabelecida, o estado será o *ESTABLISHED*. A iniciativa de encerramento poderá ocorrer de ambos os lados. Após o encerramento, o estado volta para *CLOSED*. A figura 2.9 ilustra uma máquina de estados finita TCP.

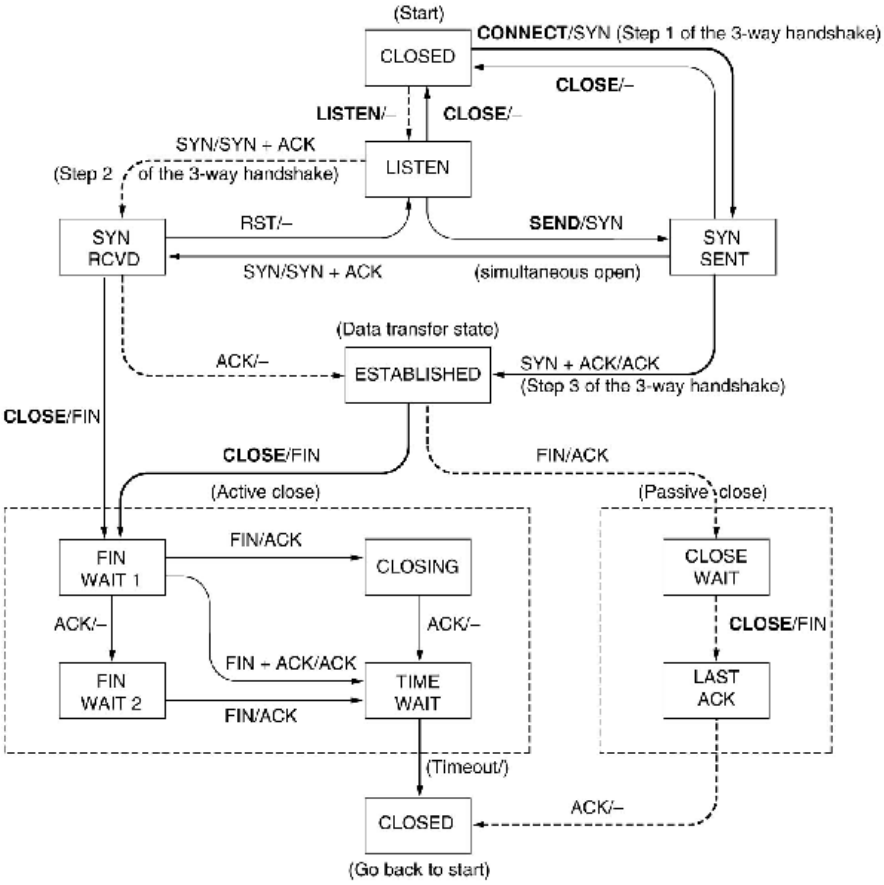


Figura 2.9 – Máquina de Estados TCP

Normalmente, na Internet, a conexão entre dois *hosts* utiliza a arquitetura cliente-servidor, ou seja, uma máquina cliente necessita de um serviço da máquina servidor e pede o estabelecimento de uma conexão para acessar o serviço. É a abertura ativa de conexão. A máquina servidor está com o serviço habilitado, no estado L*ISTEN* e aceita a conexão. É a abertura passiva. Caso o servidor não esteja no estado *LISTEN*, ele devolverá um segmento com o *flag RST* para o cliente, terminando a tentativa de conexão. A figura 2.9 mostra como seria a mudança de estado normal para os dois lados durante uma conexão. Na linha escura, a situação do cliente e na linha pontilhada escura, a situação do servidor.

O cliente, primeiramente no estado *CLOSED*, envia um *SYN* para iniciar o estabelecimento da conexão passando para o estado *SYN SENT*. Assim que ele receber o *SYN* e o *ACK* e mandar o seu *ACK*, estabelecendo a conexão, o estado muda para *ESTABLISHED*. A partir daí começa toda a transferência de dados necessária na conexão. Supondo que o cliente inicie o fechamento ativo da conexão, ele manda um segmento com o *flag* FI*N* ativado e fica aguardando no estado *FIN WAIT 1.* Assim que ele recebe o *ACK* de resposta, ele vai para o estado *FIN WAIT 2*. Caso esta resposta demore duas vezes o tempo de vida máximo do pacote, o cliente encerra a conexão. Eventualmente o servidor receberá um *timeout* e também encerrará a conexão. Quando o cliente receber a confirmação do seu *FIN* e, em conseqüência o *FIN* do servidor, ele manda o último *ACK* de confirmação e vai para o estado *TIMED WAIT*. O tempo expira e o cliente volta para o estado *CLOSED*.

O servidor está no estado *LISTEN*. Ao receber um *SYN* do cliente, vai para o estado *SYN RCVD* e responde com um *SYN+ACK*. Quando receber o *ACK* de confirmação do cliente, irá para o estado *ESTABLISHED*. Supondo ainda que o cliente inicie o fechamento da conexão, o servidor, ao receber o *FIN*, manda o *ACK* de confirmação e passa ao estado de fechamento passivo *CLOSE WAIT*. Então ele manda o seu *FIN*, passando ao estado *LAST ACK*. Ao receber o último *ACK*, ele vai para o estado *CLOSED*.

Outro aspecto importante do TCP é o congestionamento que ele tem que enfrentar. O algoritmo das janelas deslizantes com tamanho variável, juntamente com o campo *window size* evitam o congestionamento entre as extremidades. Porém, o congestionamento pode ocorrer também no meio da rede, entre os roteadores. Um congestionamento é detectado pelo TCP quando há um *timeout* dos pacotes. A solução encontrada pelo TCP para controlar o problema do congestionamento foi criar uma outra janela a *congestion window* para medir o congestionamento na rede, separando esse problema do problema do congestionamento no receptor, medido por outra janela. Cada uma delas mostra o número de bytes que o transmissor pode enviar e ele escolhe o menor valor. E como o transmissor escolhe o tamanho da janela de congestionamento? Ele ajusta a janela para o tamanho do segmento máximo em uso na conexão. Então manda uma rajada com esse segmento. Se não houver *timeout*, ele duplica o tamanho e envia uma rajada com esse segmento de tamanho duplicado, não havendo *timeout*, ele duplica novamente e manda. E assim vai, até que ocorra um *timeout* ou a janela do receptor seja alcançada. Se ocorrer um *timeout* e o segmento que foi enviado for de 2X, um terceiro parâmetro é utilizado, o limitante (*threshold*). O limitante será mantido em X e a

janela será reiniciada com o tamanho do limite. A partir daí o crescimento é linear, ou seja, uma vez o segmento máximo a cada rajada. Portanto, o *timeout* define o tamanho do limite, onde terá o início o crescimento linear. Se o crescimento linear provocar um *timeout* novamente, o limite irá baixar até metade desse novo *timeout* e o processo começará novamente. Se for recebido um pacote do tipo I*CMP source quench,* o TCP o tratará como um *timeout* [3].

O último aspecto a ser considerado no protocolo TCP é a questão do gerenciamento de temporizadores. O temporizador mais importante o do TCP é o temporizador de retransmissão. Quando um segmento é enviado, este temporizador é disparado. Assim que ele expira, o segmento é retransmitido. O grande desafio do TCP é saber qual deve ser o tempo de *timeout*. Se o intervalo for curto demais, ocorrerão retransmissões desnecessárias. Se for muito longo, prejudicará o desempenho. O algoritmo utilizado tem que ser altamente dinâmico. Ele foi criado por Jacobson (1988) e diz o seguinte: a variável *RTT* é a melhor estimativa no momento para a viagem de ida e volta. Uma vez enviado um segmento, um temporizador é iniciado para contar o tempo de viagem ou para retransmitir, se for o caso. O tempo de viagem medido é o parâmetro *M*. o RTT é atualizado de acordo com a fórmula:

$$RTT = \alpha\, RTT + (1-\alpha) M \quad (2.1)$$

onde α é um fator de suavização, normalmente com o valor de 7/8. A escolha do *timeout* nas primeiras implementações era feita multiplicando-se o valor de *RTT* por 2. A experiência não foi boa e Jacobson propôs uma fórmula que era próxima ao desvio padrão da função densidade de probabilidade dos tempos de chegada da confirmação (na verdade o desvio médio). A fórmula é a seguinte:

$$D = \alpha\, D + (1-\alpha)(RTT - M) \quad (2.2)$$

onde α é o fator de suavização que pode ser igual ou não ao anterior. A variável *D* possui uma grande vantagem. Ela pode ser calculada utilizando apenas soma, subtração e deslocamento de inteiro. A maioria das implementações calculam o *timeout* da seguinte maneira:

$$TIMEOUT = RTT + 4\mathrm{D} \quad (2.3)$$

No caso de um segmento retransmitido, pode haver confusão quanto ao cálculo de RTT. Então utiliza-se o algoritmo de Karn: o RTT não atualiza e o temporizador é dobrado a cada falha até que os segmentos cheguem de primeira [3].

O outro temporizador existente no TCP já foi citado e se refere ao caso em que o buffer do receptor está cheio e ele manda um segmento com o campo window size = 0. Então

quando o buffer libera, ele manda outro segmento com o window size disponível. Se, de repente, esse segmento se perder, cada um dos lados estará esperando resposta do outro. Então existe o temporizador de persistência no transmissor que é ativado quando ele recebe um segmento com window size = 0. Não recebendo resposta depois do timeout ele manda um segmento de 1 byte e obriga o receptor a retransmitir o segmento com o window size disponível.

2.5 CAMADA DE APLICAÇÃO

É a camada que contém os protocolos de alto nível da rede e utiliza as camadas abaixo para prover o transporte e entrega dos pacotes ao seu destino. Existem muitos protocolos nesta camada e estão sempre surgindo novos. Por esse motivo não serão detalhados neste capítulo. Optou-se por explicar um simples exemplo de funcionamento de um protocolo essencial na Internet, o DNS. Adicionalmente à explicação, será dado o alerta de algumas falhas de segurança inerentes ao protocolo DNS. Esse alerta preparará o leitor para o próximo capítulo, o de segurança da informação, onde mais protocolos da camada de aplicação e suas respectivas falhas de segurança serão apresentados.

Esta camada provê as mais conhecidas aplicações de redes e Internet para o usuário final. Dentre seus protocolos tradicionais estão protocolos de login remoto como o TELNET; de transferência de arquivos como o FTP e o TFTP; de resolução de nomes, o DNS; de gerenciamento de redes, o SNMP; de correio eletrônico, como SMTP, POP e de navegação *web*, o HTTP. Todos os protocolos citados surgiram relativamente no início da Internet e das aplicações e existem até hoje. Entretanto, como a Internet no início não tinha muita preocupação com a parte de segurança da informação e dos protocolos, estes protocolos foram mal projetados para atender a esse requisito. Com o advento da criptografia (tornar os dados indecifráveis a não ser para quem possua a chave para decifrá-los) aplicada aos protocolos, esses antigos protocolos inseguros estão sendo aos poucos substituídos: o TELNET e o FTP pelo SSH; o HTTP pelo HTTPS; o SNMP ganhou uma versão com criptografia; o correio eletrônico ganhou criptografia de mensagem com o PGP e até o protocolo IP ganhou uma versão criptografada de baixo nível, o IPSEC. A grande desvantagem dos protocolos que utilizam criptografia para os “velhos” protocolos é que eles adicionam alguns bits à

mensagem e ficam mais pesados. Entretanto, com a rápida evolução dos meios de comunicação permitindo taxas de transmissão cada vez maiores isso não chega a ser um problema.

Apesar dessa nova “camada” de segurança relativamente recente, na Internet há de tudo. Muitos sistemas antigos e legados ainda utilizam os antigos protocolos ou mesmo versões dos novos que contém falhas de segurança. Durante a dissertação esses protocolos serão explorados, pois eles são a “isca“ para os mal intencionados na Internet.

2.5.1 PROTOCOLO DNS

O protocolo DNS (*Domain Name System,* RFC 1034 e 1035) utiliza o protocolo de transporte UDP na porta 53 e surgiu simplesmente para mapear os endereços binários de rede, ou o endereçamento IP para nomes com caracteres ASCII. Ele converte os caracteres ASCII em endereços IP. Desta maneira, o usuário final que deseja acessar uma página *web*, ou mandar um correio eletrônico não precisa saber o endereço IP do destino, apenas o nome. Na antiga ARPANET, existia apenas um arquivo, chamado de *hosts.txt* que fazia esse mapeamento. No entanto com o crescimento exponencial da Internet, viu-se que era impossível manter um mapeamento centralizado apenas em um arquivo.

Criou-se, então, uma estrutura de atribuição de nomes hierárquico, baseado em domínio. Em resumo, o DNS funciona da seguinte maneira: para mapear um nome em um endereço IP, o programa aplicativo chama um procedimento de biblioteca denominado resolvedor (*resolver*) e passa seu nome para ele como um parâmetro. O resolvedor envia um pacote UDP para um servidor DNS local, que procura o nome e retorna o endereço IP para o resolvedor. Este então retorna o IP para o aplicativo, que a partir daí estabelece uma conexão TCP ou envia pacotes UDP ao IP de destino [3].

A hierarquia de domínios da Internet é como mostra a figura 2.10. Sua leitura funciona da direita para a esquerda. Na verdade existe a entidade raiz e diversas zonas abaixo dela que são administradas por uma entidade central da Internet (*NIC – Network Information Center*). No caso da empresa Novell, ela solicitou junto à NIC o registro de um subdomínio abaixo do domínio *com*. A NIC delegou a ela autoridade sobre o subdomínio novell. A partir daí, a Novell passa a ter o controle sobre *novell.com*. Caso ela queira colocar um servidor

diretamente abaixo deste subdomínio, por exemplo um servidor chamado *server*, ele terá o nome de *server.novell.com*. Caso ela queira dividir o subdomínio em outros subdomínios, como mostra a figura, também pode. Supondo que exista uma filial que queira registrar o subdomínio *provo*. Ela solicitará à matriz da empresa que detém o subdomínio *novell.com*. A matriz delegará autoridade do subdomínio *provo* e a filial passa a ter a responsabilidade sobre ele podendo, por exemplo, colocar um equipamento *host1* com o nome *host1.provo.novell.com*. Ou poderá delegar um subdomínio abaixo dele a uma subdivisão. Portanto é uma estrutura hierárquica que não faz distinção se é outro subdomínio ou se é um equipamento que está debaixo de certo subdomínio e que, teoricamente não tem fim.

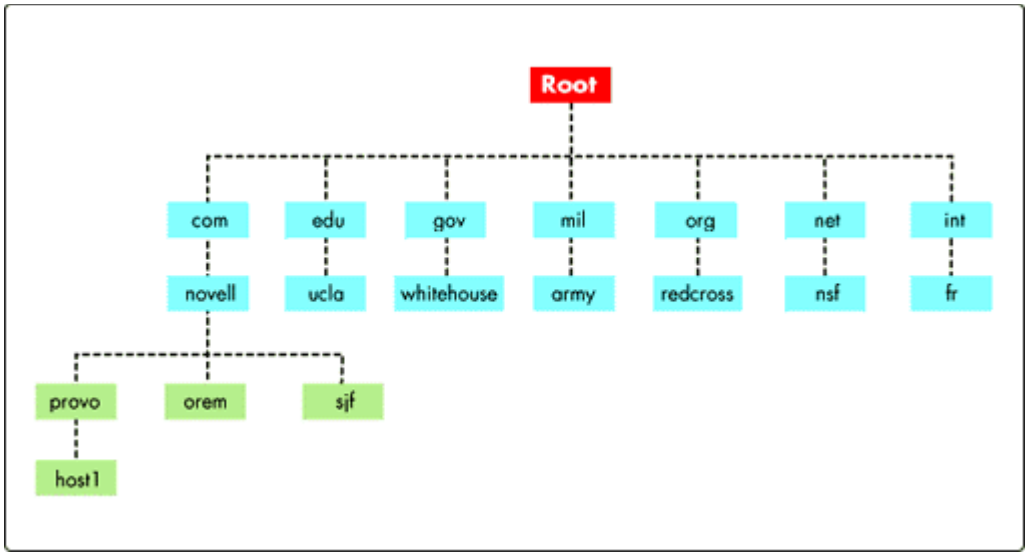


Figura 2.10 – Hierarquia DNS

O mecanismo de mapeamento de nome funciona através dos servidores de nome. Supondo que cada subdomínio possua o seu servidor de nomes, quando o resolvedor de uma máquina que esteja dentro da filial *provo* solicitar o nome *server.redcross.org*, ela enviará a solicitação para o servidor de nomes da novell, que por sua vez enviará para o servidor de nomes *com*, que por sua vez enviará para o servidor raiz, que enviará para o *org*, que enviará para o *redcross* que devolverá o IP por toda esse caminho de volta. Portanto, quando um subdomínio se registra, ele tem que fornecer o endereço de seu servidor de domínio para a entidade superior. Assim, cada servidor de nomes só precisa conhecer o endereço de seus subdomínios imediatamente abaixo e acima. Na prática, o funcionamento é mais simples, pois não há um servidor de nomes para cada subdomínio. As organizações normalmente reúnem as informações de todos os seus subdomínios em um único servidor. Normalmente os servidores locais também conhecem o endereço de outros servidores, como o raiz, para agilizar o

processo quando uma solicitação é feita passando por ele. Assim, a consulta não terá que passar por toda a hierarquia. Outro parâmetro importante a ser fornecido no momento do registro é o *TTL (time to live)* que serve para determinar quanto tempo o endereço daquele domínio poderá ficar na memória *cache* dos servidores de domínio, ou seja, quando um usuário pede para acessar um nome, o servidor verifica se ele foi processado recentemente e repassa ao usuário mais rapidamente, pois está em sua memória *cache* local. Entretanto, eventualmente, esse mapeamento poderá mudar. Por isso o TTL indica de quanto em quanto tempo o servidor terá que ir buscar novamente a informação do mapeamento.

Para as consultas inversas, ou seja, para mapear endereços IPs em nomes de hosts também há uma árvore hierárquica parecida com a das consultas normais. Uma árvore não está relacionada com a outra. A árvore inversa raramente recebe manutenção ou atualização como recebe a árvore mais comum.

Esta separação entre consulta e consulta inversa de nomes pode trazer um problema de segurança para o protocolo DNS. Uma pessoa mal intencionada que consiga controlar uma parte da árvore inversa pode fazê-la mentir. Isto é, o registro inverso pode falsamente conter o nome de uma máquina que é confiável para a vítima. O “atacante” então faz uma chamada remota (por exemplo um *ssh*, que será visto mais tarde) ao computador da vítima, que irá aceitar inocentemente achando ser a máquina confiável.

A maioria dos novos sistemas é imune a esse ataque pois implementam uma checagem cruzada. O sistema, quando quer saber o nome através do IP consulta o DNS e quando chega o nome resultante, ele faz a consulta inversa para saber se o IP é realmente o que ele havia mandado na primeira consulta. O inverso também funciona.

Ainda há outra variante deste ataque, onde o atacante consegue envenenar a memória *cache* da resposta de DNS da vítima anterior ao início da chamada de checagem cruzada. Então, quando a checagem acontece, tudo parece estar bem, mas na verdade o intruso obteve sucesso e ganhou acesso.

No capítulo seguinte, serão apresentados outros protocolos da camada de aplicação que possuem falhas semelhantes em seus mecanismos, inclusive os principais protocolos como IP, TCP ou ICMP. Essas falhas deixam a máquina conectada à Internet vulnerável a uma gama de ataques para obter privilégios, executar comandos, ou mesmo tomar o controle da máquina. A única maneira de evitá-los é adotar uma série de medidas de segurança que são mandatórias para a navegação na Internet nos dias de hoje.

3 SEGURANÇA DA INFORMAÇÃO

3.1 INTRODUÇÃO À SEGURANÇA DA INFORMAÇÃO

A segurança da informação na Internet não é diferente de nenhum outro tipo de segurança. As variáveis envolvidas e os detalhes técnicos são bastante diferentes dos outros tipos, mas existem premissas e experiências acumuladas ao longo do tempo que não devem ser esquecidas.

Em primeiro lugar, não existe nada com segurança absoluta. Cada técnica, cada protocolo, cada aplicação, cada sistema tem sua falha de segurança. Seja por mau uso, por concepção falha, implementação falha, ou outro motivo. Existe sempre uma "porta" aberta ao atacante. Seja ela conhecida ou não. O atacante estará sempre rondando à procura dessa porta. Existe um conceito que muitas vezes ajuda na defesa de um sistema, mas pode ser muito enganoso e traiçoeiro: é a segurança por obscuridade. Às vezes um atacante não consegue invadir um sistema pois ele não conhece nada sobre aquele sistema. Não conhecendo seu funcionamento, não há como achar uma falha de segurança. O defensor, porém, deve sempre assumir que o inimigo conhece bem o sistema. Conhece como se fosse ele próprio, ou melhor do que ele. Essa é a premissa mais segura a se assumir. Estar sempre preparado para qualquer tipo de ataque é o objetivo dela e a segurança por obscuridade nunca é um bom caminho.

A proteção de um sistema é sempre construída em camadas, assim como uma cebola. Um atacante consegue passar por uma barreira de proteção, então há outra para barrá-lo e o sistema está salvo. As camadas são constituídas por dispositivos físicos, softwares ou políticas de segurança. Quanto mais diversificada for a proteção, mais segurança existirá. Todas as camadas tendem a ter a mesma importância. Não adianta investir todo o tempo e dinheiro disponível em um tipo de proteção, deixando as outras de lado, pois um furo em qualquer uma delas terá a mesma conseqüência. Para uma empresa, investir em segurança é sempre uma economia financeira. Os danos de ter um sistema invadido, informações roubadas e credibilidade no mercado abalada são financeiramente muito altos e incalculáveis. O preço de *hardware*, *software* e mão-de-obra especializada de acordo com a necessidade é calculável e

quase sempre menor. Há que se calcular qual é o risco a que o negócio está sujeito na Internet, o valor do patrimônio a defender e a partir daí investir na segurança.

Ao planejar um sistema, ou uma solução, a segurança deve sempre ser um dos pilares do projeto. Não é a partir do problema que se conserta o sistema, sempre que possível, como está no ditado popular: “é melhor prevenir do que remediar”. Mudar um sistema depois que está feito por questões de segurança é um problema. A programação original não é fácil, mas se for bem feita poderá evitar muitos problemas futuros. A manutenção também é muito importante. Nenhum *software* é concebido sem *bugs* ou falhas. A utilização é o seu maior teste. Se um *software* não é necessário para o bom funcionamento do sistema, é melhor não utilizá-lo. Um programa ou protocolo deve sempre ser tratado como inseguro, até que se prove o contrário, ou seja, devem-se negar todos os serviços e processos de um sistema, a não ser que eles tenham provado ser essencial para o bom funcionamento do mesmo. Permitindo estes, assume-se o risco do que venha a acontecer.

O inimigo é invisível. Não se sabe quem é ou que quer. Pode estar em qualquer lugar fisicamente e pode utilizar todo o tipo de técnica ou informação para montar sua estratégia de ataque. Não se pode subestimar qualquer tipo de informação. Um papel jogado no lixo, uma conversa informal, um e-mail qualquer. Tudo é fonte de informação. Não é possível avaliar qual o valor que ela terá para o inimigo, porém é possível evitá-la o máximo possível. O mais prudente é não confiar em ninguém e tentar imaginar quem teria interesse no ataque. Um funcionário de uma empresa não deve ter acesso a mais programas e sistemas do que o necessário para o trabalho. Pesquisas mostram que um ataque interno a uma empresa, isto é, feito por seus próprios funcionários obtém sucesso muito mais facilmente e é muito mais freqüente do que se imagina. É necessário prevenir ataques internos e externos isolando os sistemas mais importantes. A segurança física é essencial. É absurdamente mais fácil invadir uma máquina a que se tem acesso físico do que uma máquina remota, que não se sabe ao certo como está funcionando. A proposta do presente trabalho é enfocar a proteção contra invasões remotas. Entretanto proteções físicas não são menos importantes, são provavelmente até mais importantes.

Supondo que uma nova rede esteja nascendo e que seja necessário definir os mecanismos de segurança que serão utilizados. Primeiramente, é necessário dividir o problema em três partes: a segurança dos *hosts*, a segurança da rede e a segurança de *software*.

A segurança de *software* envolve boas práticas de programação e preocupação constante com segurança. Precisa ser amplamente divulgada aos programadores e apoiada pela alta gestão. O principal recurso de segurança disponível ao programador é a criptografia. Consiste na aplicação de algoritmos matemáticos para cifrar a mensagem, isto é, embaralhar os bits de tal maneira que qualquer pessoa que consiga capturar os dados não consiga entender do que se trata. Apenas o usuário que possui a chave para decifrar conseguirá remontar a mensagem. Esta chave pode ser a mesma chave utilizada para cifrar a mensagem, ou pode ser diferente, dependendo do algoritmo de cifragem utilizado. Existem protocolos e algoritmos para gerenciar a troca dessas chaves. Além disso, existe uma arquitetura denominada PKI (*Public Key Infrastructure*) criada para fornecer essas chaves em grande escala, através dos chamados "certificados digitais". Este assunto será visto, ainda que superficialmente, mais adiante no capítulo.

Para planejar segurança de *hosts* é essencial que se saiba qual é a finalidade daquele *host*, ele será um servidor, ele será uma máquina de usuário final, um dispositivo de rede, de segurança? Após isso, é necessário saber que sistema operacional será instalado nele. Depois, quais serviços funcionarão nele e que tipo de usuário irá operá-lo. Levantados esses requisitos, é hora de instalar o sistema operacional. Com o sistema operacional instalado, vem uma parte muito importante na segurança de *hosts*, o *hardening*. Significa retirar todos os serviços dispensáveis do sistema operacional, fechar portas abertas, mudar permissões de arquivos. Ou seja, significa "blindar" o sistema operacional. A dificuldade está nas especificidades de cada sistema operacional. Há um *hardening* específico para cada sistema e para cada função que esse sistema irá desempenhar. Por isso, às vezes, só é possível fazer o *hardening* nas máquinas mais importantes. Entretanto, o ideal de segurança é que se faça um *hardening* mínimo em todas as máquinas. Em seguida, é necessário equipá-lo com todos os *softwares* de segurança possíveis e necessários, principalmente em máquinas Windows de usuário final, pois elas são as mais atacadas e estarão na *net* visitando lugares desconhecidos. Esses *softwares* incluem: *anti-vírus*, *anti-trojan*, *anti-spyware*, *firewall* pessoal, *IDS* de *host* e outros. O *host* está pronto para ser plugado na rede. Porém, a segurança dele não pára por aí Após essa fase, se for necessário instalar um novo *software* na máquina, tem que se avaliar se ele está de acordo com a política de segurança da empresa. Além disso, tem que haver uma política de *advisory* e de *patch management*. Diariamente na Internet são divulgados diversos *bugs* e "vulnerabilidades" (falhas de segurança) nos diversos sistemas operacionais e

softwares nas diversas áreas existentes. A política de *advisory* consiste em acompanhar essas notícias e, diante de um cadastro dos sistemas e *softwares* existentes dentro da empresa, avisar dos *bugs* e vulnerabilidades divulgados. A política de *patch management* se encarrega de automaticamente, quando for possível, aplicar os *patchs* de segurança e atualizações nas máquinas vulneráveis. Os *patchs* são códigos que "consertam" uma falha que foi descoberta em um sistema. Há um problema muito grave nesse ponto, que muitas vezes um *patch* de segurança muda permissões de arquivos ou mexe em partes do sistema que não poderia mexer, pois prejudica o funcionamento de outros sistemas existentes, ou do próprio sistema. Por isso, muitas vezes na prática, há um receio em se aplicar os *patchs* de segurança. Em um nível de "paranóia" mais alto, para computadores que possuem informações estratégicas, a solução é o EFS (*encrypt file system*). Consiste em criptografar de alguma maneira os dados contidos no *HD* de tal forma que apenas o usuário que tiver a chave para decifrar os dados consiga fazê-lo.

A segurança de rede, como era de se esperar, diz respeito aos dispositivos de rede existentes: *switche*s (equipamentos de rede que entregam pacotes através do endereço físico MAC), roteadores, *firewalls*, *IDS*, etc. E também à arquitetura construída com ajuda deles. Em primeiro lugar, a segurança de *hosts* se aplica a todos esses dispositivos. Os roteadores, *firewalls* e *switches* devem estar com o sistema operacional atualizado, os *patchs* aplicados e o *hardening* realizado. O ponto chave na segurança das redes diz respeito ao AAA (*Autentication, Autorization* e *Accounting*), ou autenticação, autorização e auditoria. Existem dispositivos como *RADIUS* e *TACACS* que conseguem realizar essas tarefas centralizadamente. A autenticação diz respeito a quem está acessando o sistema. Normalmente o usuário possui um *login* e uma senha que fornece acesso a um determinado sistema. Com a criação da arquitetura PKI, essa política está sendo substituída pela política de certificação digital. O usuário que possui um certificado digital, a princípio, consegue provar a sua autenticidade. Adicionalmente pede-se o *login* e senha para garantir que o certificado digital não tenha sido roubado. A autorização diz respeito a quais recursos o usuário pode acessar quando está dentro de um sistema. Um exemplo de implementação de autenticação e autorização está nos roteadores. Normalmente os roteadores são dispositivos que ficam muito expostos, nas bordas da Internet. Existe uma ACL (*Access List*), ou lista de acesso, que lista quais usuários estão permitidos de acessar aquele roteador e qual é o nível de permissão de determinado usuário. Pode-se realizar uma lista de acesso com a ajuda de um RADIUS ou TACACS, utilizando certificados digitais. Além disto eles permitem um *accounting*, ou

auditoria. Desta maneira todos os registros de quem acessou, a que horas, para fazer o que estarão guardados em um arquivo, ou em um banco de dados. O dispositivo ou elemento central de segurança de uma rede chama-se *firewall*. Ele implementa um muro de proteção e segmenta redes, podendo funcionar como um *gateway* e fazendo o papel dos roteadores. A característica de segmentação de redes realizada pelo *firewall* é muito importante, pois ele pode criar uma rede particular de especial importância do ponto de vista de segurança chamada de DMZ, ou zona desmilitarizada. A DMZ separa os servidores do restante da rede, protegendo-os da rede interna e protegendo a rede interna da rede externa. Desta maneira, uma configuração muito comum deixa a rede interna de um lado, a rede externa do outro e os servidores do outro, sendo o *firewall* o ponto central que através de regras configuráveis, rotas e traduções de endereço (NAT, PAT) controla o acesso a cada serviço e a cada máquina dentro dessas redes. Além disso ele pode inspecionar pacotes, filtrar conteúdo, gerenciar VPNs (túneis seguros), etc. A VPN (*Virtual Private Network*) é uma outra camada da segurança de redes que utiliza a criptografia. De um lado da comunicação, os pacotes IPs são criptografados e jogados na Internet. O outro lado (muitas vezes o *firewall*) descriptografa e reconstrói o tráfego. Existem dispositivos específicos para isso também, os concentradores de VPN. A criptografia dos pacotes IPs é feita por um protocolo especialmente com essa função que atua acima da camada IP, o IPSEC. A VPN também pode ser construída em camadas acima, por exemplo através do protocolo SSL. Por último, na área de segurança das redes, destaca-se o objeto principal deste estudo, o IDS (*Intrusion Detection System*) ou sistema de detecção de intrusão. Um sistema que inspeciona todos os pacotes que entram em determinada rede, ou determinado segmento à procura de pacotes que possam ser considerado uma intrusão, ou um ataque. Cada pacote que chega, ele compara a uma base de determinados padrões de intrusão que possui e determina se pode tratar-se de uma intrusão ou não. Pode funcionar em apenas um dispositivo, ou distribuído pela rede. Uma desvantagem é que a intrusão precisa estar na base de padrões de IDS, qualquer ataque que consiga fugir deste padrão passará desapercebido. Este tipo de IDS é chamado de IDS baseado em assinatura. O IDS que está sendo proposto no trabalho é um IDS mais inteligente que consigue, através da generalização inerente das redes neurais artificiais, determinar um ataque, mesmo que esse não tenha sido visto antes pelo IDS. Entretanto este assunto será muito debatido posteriormente. Outra grande desvantagem do IDS é a quantidade de falsos positivos que ele oferece. Isso quer dizer gerar um alarme de ataque, sendo que o tráfego é normal. Outra

desvantagem é a incapacidade de detectar ataques contidos dentro de túneis, ou seja, no tráfego criptografado. Uma última desvantagem é que ele consegue detectar a maioria dos ataques, mas não consegue preveni-los, agindo antes que eles aconteçam. Para esta tarefa, estão surgindo os IPSs (*Intrusion Prevention System*) baseados em assinatura em conjunto com inteligência artificial.

O capítulo está organizado da seguinte maneira: primeiro serão exploradas algumas vulnerabilidades, ou falhas de segurança existentes em alguns dos principais protocolos da Internet. Em seguida, serão vistos alguns tipos de ataques possíveis e depois discutido os mecanismos de segurança existentes para contê-los. Os atacantes normalmente utilizam combinações desses ataques afim de comprometer um sistema. A defesa também deve ser construída, como discutido, através de combinações de técnicas e dispositivos.

3.2 VULNERABILIDADES EM PROTOCOLOS

As falhas nos protocolos e aplicações são a principal arma de todo atacante. É em conseqüência delas que todos os ataques que serão vistos mais tarde acontecerão. Portanto, é essencial entender o funcionamento do protocolo e cada vulnerabilidade que ele possa proporcionar. O que será apresentado adiante é apenas "a ponta do *iceberg*", isto é, exemplos do funcionamento e exploração de alguns protocolos importantes. Para aprofundamento no assunto é sugerida a leitura da bibliografia apresentada e de outras intermináveis fontes disponíveis na Internet ou nas livrarias.

3.2.1 CAMADA INTERNET

3.2.1.1 IP

O protocolo IP, visto com detalhes no capítulo anterior está sujeito a dois tipos de ataques muito comuns. Eles são construídos a partir de características do funcionamento normal do IP.

O primeiro ataque é chamado de IP *spoofing* e consiste em mascarar a origem do

pacote, ou seja, fabricar um IP de origem falso para enganar o destino quanto à origem do pacote. Qualquer *host* pode fabricar um datagrama IP com qualquer endereço de origem. Cabe ao sistema receptor fazer uma verificação reversa. Não se pode confiar no campo *IP Source Address* do protocolo IP, a não ser em situações extremamente controladas [5].

O segundo ataque é mais complicado de se verificar. Trata-se de utilizar a fragmentação dos datagramas IPs para realizar, por exemplo, um ataque de negação de serviço, que será mais detalhado posteriormente. O datagrama IP possui um tamanho máximo de 64KB, porém o tamanho dos quadros físicos normalmente são menores. O quadro Ethernet, por exemplo, é de 1500 bytes. Alguns podem ser menores ainda. A medida da unidade máxima de transmissão de um quadro físico é a MTU (*Maximum transfer unit*). Quando um datagrama precisa atravessar de uma MTU maior para uma MTU menor, o datagrama precisa ser fragmentado e o IP utiliza o campo *fragment offset* para indicar a qual parte do datagrama o fragmento pertence e o *flag* MF para indicar que há mais fragmentos a chegar. Na chegada, o destino remonta o datagrama com base nesses campos. O cabeçalho IP é mantido em todos os fragmentos. Entretanto, o cabeçalho dos protocolos acima como TCP, ou ICMP normalmente estão apenas no primeiro fragmento. Com isso, o dispositivo receptor (roteador, *firewall* ou IDS), para analisar o datagrama precisa manter um estado com o ID do datagrama e esperar a chegada de todos os fragmentos. O que acontece muitas vezes é que esse dispositivo não mantém o estado e o primeiro fragmento, que contém o cabeçalho TCP ou ICMP é barrado, ou detectado e os demais passam e simplesmente vão se acumulando esperando o complemento e acabam derrubando o dispositivo, provocando a negação de serviço. O *flag* DF (*don't fragment*) também pode ser utilizado pelo atacante. Ele induz o dispositivo de rede a não fragmentar, descartando o pacote e mandando um pacote ICMP informando o tamanho da MTU. A informação pode ser útil para o atacante mandar datagramas maiores que a MTU, obrigando a fragmentação.

3.2.1.2 ARP

O protocolo ARP foi visto no capítulo anterior e consiste em mensagens automáticas de requisição e resposta de tradução de endereços IP para endereços MAC (físicos). Um atacante facilmente pode fabricar essas mensagens, ou modificá-las. Uma técnica chamada de ARP *spoofing*. Por exemplo, uma maquina lança a pergunta: “quem é o IP A?" Um atacante

intercepta essa mensagem e intercepta a mensagem legítima de resposta, se houver. Então ele fabrica uma resposta dizendo: “O IP A sou eu, MAC X”. A partir daí a máquina de origem passará a se comunicar com ele, confiando que ele é realmente o IP A. Em alguns sistemas seguros, o ARP é inibido e a tradução de endereços é feita de forma estática para evitar esse tipo de ataque. Em sistemas que utilizam tráfego criptografado com VPN IPSec não é possível realizar esse ataque.

3.2.1.3 ICMP

O ICMP pode ser utilizado de diversas maneiras como ferramenta de ataque. Existe um ataque chamado *smurf* do tipo DoS (*denial of service*) ou negação de serviço. Consiste em mandar pacotes ICMP *echo request* para um endereço de *broadcast* utilizando o *spoofing* para direcionar as respostas para o alvo. Desta maneira diversos *hosts* dessa rede irão responder com o *echo reply* para o alvo, até que a memória dele seja insuficiente e ele não consiga mais responder nem conexões legítimas, criando uma condição de negação de serviço.

Outro ataque é o ICMP *fingerprint* que consiste em mandar um ICMP *echo request* e analisar o ICMP *echo reply*. A pilha TCP/IP é implementada de maneira diferente nos sistemas operacionais e a resposta ao ICMP às vezes é diferente. Compara-se a resposta a uma base de assinaturas existente e sabe-se qual tipo de sistema operacional está rodando no alvo.

O ICMP *echo request* também pode ser utilizado para descobrir quais máquinas estão ativas e quais não estão numa rede. É uma ferramenta importante para o administrador de rede e também para o atacante.

Outro tipo de ataque com o ICMP é o tunelamento. O atacante esconde informações ilícitas dentro do payload do ICMP *echo request*, que normalmente é vazio e permitido nos *firewalls* por ser uma ferramenta de administração. Dentro do sistema já existe um programa que foi instalado pelo atacante anteriormente (*backdoor*, visto mais adiante) que espera aquelas informações que passaram. A maneira de evitar é bloquear os pacotes ICMP *echo*.

O ICMP ainda pode ser fabricado com um tamanho maior do que o normal. Muitos sistemas não estão preparados para lidar com isso e quebram.

3.2.2 CAMADA DE TRANSPORTE

3.2.21 TCP

O TCP, ao estabelecer uma conexão, utiliza o *three way handshake.* É necessário que os dois lados da conexão troquem segmentos de sincronização, contendo os números seqüenciais e confirmações. Esta característica pode ser explorada pelo atacante. O primeiro segmento da conexão utiliza o *flag* de sincronização SYN. O atacante envia um segmento SYN com um IP de origem falso (IP *spoofing*). O receptor irá responder com um SYN+ACK para o endereço falso que nunca fechará o terceiro passo (ACK). Entretanto os recursos de memória do receptor permanecem esperando por uma confirmação. O atacante envia centenas desses segmentos SYN com endereço falso, até que o receptor estoure o seu *buffer* e não consiga aceitar mais nenhuma conexão, legítima ou não. O resultado é uma negação de serviço e muitas vezes a máquina pára, necessitando ser reiniciada. Este tipo de ataque é chamado de SYN *flood* e é um tipo de negação de serviço (DoS). Normalmente este ataque é evitado aumentando-se o *buffer*, diminuindo o *time out* das conexões, e prevenindo o IP *spoofing*.

Outro ataque ao TCP pertence à classe dos ataques do tipo *scan*. Trata-se do TCP *scan* ou SYN *scan*. O atacante manda o mesmo segmento SYN para iniciar a conexão, porém com o endereço de origem correto. Determinada porta é especificada. Caso a resposta seja um SYN+ACK, a porta está aberta à espera de conexão. Caso contrário, ela está fechada. O atacante testa todas as portas do alvo.

O TCP, por fim, poderá ter a sua sessão “seqüestrada” num ataque do tipo *man-in-the-middle*. Um atacante se coloca no meio de uma conexão TCP, derrubando a máquina de destino com um ataque de negação de serviço, respondendo à máquina de origem como se fosse o destino, através do IP *spoofing* e redirecionando o tráfego de origem para si próprio. Também há a possibilidade do atacante prever ou “adivinhar” o *sequence number* inicial de uma sessão e introduzir pacotes esperando que o alvo confie e comece a direcionar pacotes a ele.

3.2.2.2 UDP

O UDP é um protocolo sem confiabilidade, que não provê o mesmo controle do TCP. Realizar um IP *spoofing* em um datagrama UDP é muito mais fácil do que no TCP. Ele é enviado uma vez, sem nenhum tipo de confirmação, ou conexão. O UDP pode ser utilizado também em um ataque de *scan*. Apesar de nem todos os sistemas operacionais implementarem, a RFC 1122 prevê que quando uma mensagem UDP é entregue a uma porta fechada, o sistema retorne uma mensagem ICMP *Destination Unreachable*. Portanto o atacante testa as portas do alvo. As que retornarem a mensagem estão fechadas, caso contrário, estarão abertas.

3.2.3 CAMADA DE APLICAÇÃO

Além do ataque em cima do DNS discutido no capítulo anterior, diversos protocolos de aplicação possuem graves falhas de segurança e permitem exploração. Esta seção procura tratar os principais de maneira resumida.

3.2.3.1 SMTP e MIME

O SMTP (*Simple Mail Transfer Protocol*) é um dos protocolos mais populares da Internet. É utilizado para mover e-mails na rede. Um SMTP tradicional transfere caracteres de texto ASCII de 7 bits. Dentre os comandos do SMTP, existe o *MAIL FROM*. Este comando informa quem está mandando o e-mail. Novamente como acontece com vários protocolos, ele pode sofrer *spoofing*. Não há como saber ao certo quem mandou o e-mail. Esse controle tem que ser feito por algum mecanismo superior. Um servidor de e-mail, também, sempre está sujeito a um ataque de DoS, recebendo uma quantidade de e-mail acima de sua capacidade. Outro ataque conhecido de SMTP é o *spam*, que consiste em mandar um e-mail de propaganda, ou contendo código malicioso para uma quantidade de pessoas muito grande. Normalmente o atacante invade servidores SMTP, aproveitando-se de alguma falha de segurança e o faz mandar um e-mail para milhares de endereços simultaneamente, enchendo a caixa de mensagens dos usuários.

O MIME (*Multipurpose Internet Mail Extensions*) é um protocolo utilizado para

executar automaticamente e-mails recebidos e pode ser muito perigoso. Ele é sujeito a ataques de fragmentação semelhante ao IP. Permite a quebra de um e-mail em diversos pedaços que pode ser utilizado para enganar anti-virus e filtros de conteúdo. Alguns clientes de e-mail, como o Outlook Express permitem a refragmentação. Outro perigo é a execução automática de programas e o envio de *scripts* que podem ser maliciosos.

3.2.3.2 RPC e NIS

O protocolo RPC (*Remote Procedure Call*) é muito importante e provê serviços importantes na rede. É, também, um perigo em potencial. É um protocolo que oferece aos programadores de serviços de rede uma linguagem para especificar os nomes e parâmetros dos pontos de entrada externos, traduzindo-os em rotinas internas. Assim a programação fica facilitada e os programadores podem fazer chamadas facilmente para um servidor remoto utilizando TCP ou UDP. O perigo está na hora da autenticação. Dos serviços que exigem autenticação, exigem o número do usuário e grupo que está chamando e nome da máquina que está chamando. Não há como garantir a autenticidade dessas informações, a não ser que se utilize criptografia. Um importante protocolo que utiliza RPC, o NFS (*Network File System*) possuía essa falha nas primeiras versões, mas já foi corrigido[5].

A maioria dos servidores baseados em RPCs não utilizam uma porta fixa para se comunicar (a exceção do NFS) e aceitam a porta que o sistema operacional designar. Para isso, utilizam o *rpcbind*, ou *portmapper*. Ele próprio é um RPC e intermedia a negociação entre clientes e servidores. Para contatar um servidor, o cliente primeiro pergunta ao *rpcbind* do servidor qual é o número da porta e o protocolo (TCP ou UDP). Além dessa função, o *rpcbind* possui algumas facilidades para o atacante. Pode, por exemplo fazer uma chamada para tirar o registro de um determinado serviço, ou informar a todos na rede quais serviços RPC estão rodando (*rpcinfo).* Entretanto, o maior perigo do *rpcbind* é a capacidade de realizar chamadas indiretas. Para evitar cabeçalho extra na determinação de um número de porta, o programa pode pedir que o *rpcbind* repasse a chamada de RPC diretamente para o servidor. O problema é que a mensagem encaminhada deve conter o endereço de retorno do próprio *rpcbind* e não há como o servidor distinguir entre requisições legitimas e não legítimas. Algumas versões do *rpcbind* contém seu próprio filtro para evitar esse problema [5].

Outra aplicação perigosa de RPC é o NIS (*Network Information Service*). Ele é usado

para distribuir importantes informações de um servidor central para os clientes. Informações como arquivos de senha, tabelas de endereços, chaves de criptografia usados para RPC seguro. Caso o *host* seja corrompido, essas informações podem cair em mãos erradas. Outro perigo que ele oferece é a possibilidade de redirecionar clientes remotamente para servidores NIS secundários, caso o primário esteja fora do ar.

3.2.3.3 FTP, TELNET e SSH

O FTP é um protocolo muito antigo e útil para transferência de arquivos. Ele ainda é muito utilizado na *net* para disponibilizar documentos, *papers*, imagens e outras coisas de utilidade pública. Também é utilizado para distribuir programas maliciosos e ilícitos. A autenticação é feita através de usuário e senha e ele oferece uma console de linha de comando. Nas antigas versões do protocolo havia um comando de conexão que apresentava alguns problemas: o *PORT*. O cliente escutava em uma porta aleatória e mandava o comando *port* para o servidor avisando em qual porta ele deveria se conectar. Então ele se conectava. Qualquer *firewall* que aceitasse o protocolo FTP tinha que prever esse comando. Então o atacante podia, por exemplo, distribuir um código malicioso que quando executado fizesse um *port* para a sua máquina remota habilitando sua conexão ao alvo, passando pelo *firewall*. Com o comando *PASV*, o cliente passou a fazer a conexão com o servidor. Apesar disso, o *ftp* ainda continua sendo um protocolo inseguro, que não utiliza criptografia e há que se tomar diversas precauções ao se utilizar. Nunca dar permissão de escrita na área de *ftp*, usar o arquivo *.rhosts* para determinar os *hosts* confiáveis, não deixar cliente remoto algum mudar permissões de arquivos, evitar deixar o arquivo de senhas na área de *ftp*, ficar atento aos *bugs* de servidor.

O TELNET também é um protocolo antigo, popular e inseguro. É um terminal de acesso para uma máquina que aceita conexões de várias máquinas não confiáveis e pode ter a senha facilmente capturada por *sniffers*. O programa possui possibilidades de ser compromissado para logar sessões ou entregar usuários e senhas. Portanto, é um protocolo bastante inseguro que raramente é utilizado nos dias de hoje, principalmente depois que surgiu o *ssh*. Outros protocolos semelhantes como o *rlogin*, o *rsh* e o *rcp* (*remote login, remote shell* e *remote copy*) da arquitetura BSD possuem falhas semelhantes e foram substituídos pelo *ssh*.

O SSH é um protocolo criado para substituir os programas citados (inclusive *ftp*). Ele

possui exatamente as mesmas funcionalidades e adicionalmente fornece criptografia assimétrica (discutida mais tarde) aos dados evitando a maioria das falhas. Provê terminal remoto, cópia de arquivo (*scp*) e até tunelamento de protocolo X11 (gráfico) e outros protocolos. É, atualmente, uma ferramenta essencial para a manutenção e operação das redes e será visto com mais detalhes posteriormente. Apesar disso, possui também uma pequena falha. para que possam ser construídos *scripts* para utilizar o SSH de maneira automática, ele possui um arquivo chamado *autorized_keys* que cadastra chaves que não precisam de autenticação usuário-senha. Se, de alguma maneira, o atacante conseguir substituir esse arquivo, ele será autorizado a se logar no sistema via *ssh*.

3.2.3.4 X11

O X11 é o protocolo dominante para ambientes gráficos em sistemas UNIX e utiliza a rede para comunicar entre aplicações e dispositivos de entrada e saída, permitindo que estes estejam localizados em máquinas diferentes. É um poder e um perigo muito grande. O servidor controla todos os dispositivos de interação. As aplicações fazem chamadas ao servidor quando querem acionar o usuário. Caso os parâmetros estejam corretos, ela conecta-se ao usuário sem nenhuma intervenção do servidor. Portanto todos os dispositivos passam a ser controlados pela aplicação, o que constitui um perigo enorme. Se o servidor estiver desprotegido, qualquer atacante de qualquer lugar na *net* será capaz de sondar o servidor X11. A porta do servidor normalmente é a 6000 ou 6000 e pouco. Para que não aconteça a invasão o protocolo provê uma série de proteções. A primeira é a autenticação baseada no endereço do *host*, guardado em um arquivo. Obviamente, facilmente burlada pelo *spoofing* ou seqüestro de sessão. A segunda é o *magic cookie*, um byte secreto compartilhado por aplicação e servidor. Apenas se a aplicação souber esse byte estará apta a operar. Todavia, ele pode ser roubado na rede. O terceiro mecanismo é um desafio/resposta utilizando a criptografia. Entretanto a chave também pode ser roubada na rede, caso não haja um método forte de autenticação. Por isso, a melhor maneira de utilização do protocolo X11 é mesmo mantê-lo restrito apenas à máquina local, ou se necessário utilizar o tunelamento através do IPsec ou *ssh* para um host remoto [5].

3.2.3.5 WWW, HTTP e HTTPS

A Internet, muitas vezes, é confundida com seu conjunto de aplicações e protocolos mais populares, o WWW (*World Wide Web*). Seu protocolo mais utilizado é o HTTP (*Hipertext Transfer Protocol*). A função do HTTP é de transferir um bloco de informações e uma descrição de tipo de dados para o cliente. O *browser* é o *software* cliente responsável por interpretar a informação e apresentar ao usuário da maneira correta. O grande problema é que códigos executáveis são tipos válidos e alguns *browsers* são configurados para interpretá-los e processá-los automaticamente. Em conseqüência, o atacante pode utilizar códigos maliciosos para atacar a vítima[9]. Normalmente o cliente contata um servidor e manda uma requisição. A resposta pode ser um arquivo ou um apontamento para outro servidor. Quando a resposta é um arquivo, além do perigo do *script* malicioso discutido, ele pode conter *tag*s que especifica o programa a ser utilizado para processar o documento. Isso pode ser utilizado também pelo atacante de forma maliciosa. O HTTP possui a característica de não possuir estados. Quer dizer que cada requisição envolve uma conexão separada com o servidor. Depois que o documento foi transmitido, a conexão cai. Para estabelecer uma sessão o HTTP precisa de outros métodos de saber o estado da sessão. Ele pode utilizar um *GET*, que passa o parâmetro de estado através da barra de endereço do *browser* ou um *POST*, que passa o parâmetro de maneira escondida. Ainda pode utilizar os *cookies*, arquivos que são criados na máquina do cliente para guardar o estado. De qualquer maneira, é perigoso que o servidor acredite que esses estados que estão na máquina do cliente representem o verdadeiro estado da sessão, pois eles podem ser facilmente modificados. Outro cuidado que deve-se ter no servidor é deixar as informações importantes em um *host* diferente do servidor. Muitas vezes o cliente, ao fazer uma requisição ao servidor passa informações a um programa que roda no próprio servidor e que processa a informação. Algumas vezes esses programas não são confiáveis e possuem furos de segurança.

A maneira mais segura de realizar uma sessão é aplicando a criptografia. Para isso, existe um protocolo desenvolvido pela *Netscape Communications* chamado de SSL (*Secure Socket Layer*). Ele opera na camada de transporte e é independente de protocolo de aplicação. O uso mais comum é o HTTPS, utilizado para criptografar o tráfego HTTP. Assim como o SSH, utiliza algoritmos de chave pública, ou assimétricos em conjunto com a criptografia simétrica, que serão mais discutidos adiante.

Assim como todos os protocolos apresentados nessa seção, existem diversos outros protocolos de aplicação e todos eles possuem alguma falha de segurança ou alguma

possibilidade de serem utilizado maliciosamente. Resta ao administrador conhecer minimamente essas possibilidades para manter seu serviço funcionando de forma segura.

3.3 ATAQUES

3.3.1 ENGENHARIA SOCIAL

O ataque de engenharia social é realizado em cima de pessoas, normalmente funcionários despreparados de uma empresa que pressionados por alguém que aparenta ser o que não é entregam informações importantes ou tomam ações induzidas por esse alguém, o atacante. Pode acontecer também de forma passiva através da espionagem, ou por erros ou omissões dos funcionários.

Na verdade o ataque procura controlar o elemento humano da segurança e tem como grande mestre o famoso *hacker* Kevin Mitnick, que conseguiu resultados impressionantes com sua engenharia social. As técnicas vão desde remexer o lixo de uma empresa, à procura de manuais, documentos, memorandos, desenhos até a persuasão de pessoas por meio de telefonemas, encontros, mentiras, etc. Sempre no intuito de agregar informações ao seu objetivo: invadir o sistema.

3.3.2 ROUBO DE SENHA

Descobrir a senha é a maneira mais fácil de entrar no computador pela “porta da frente”[5]. Normalmente os sistemas são baseados em usuário e senha para acesso. O atacante tentará usar todas as senhas que forem possíveis para determinado usuário até conseguir “adivinhar” a senha correta de acesso. Esses ataques são conhecidos como ataques de força bruta e existem *softwares* especializados nesse tipo de ataque que utilizam um dicionário de possíveis senhas para realizá-lo. Uma maneira de evitá-lo é construir senhas fortes, evitando partes do nome, data de nascimento, nome de pessoas queridas e outras informações que

podem ser descobertas através de engenharia social. Recomenda-se também que se construa senhas grandes utilizando caracteres numéricos, alfanuméricos e especiais alternadamente. Outra maneira é desabilitar *logins* conhecidos como: *guest, admin, root,* primeiro nome do usuário. Com isso o atacante não saberá qual usuário atacar. E configurar o sistema para que bloqueie o usuário após determinado número de tentativas falhas de *login*. Um número ilimitado de tentativas oferece um número ilimitado de chances ao atacante. O sistema também deverá registrar o número de tentativas de *login* com insucesso, para que o usuário saiba que foi atacado. Normalmente, quando há essa proteção de bloqueio após tentativas falhas, a estratégia do atacante é capturar de alguma maneira o arquivo de senha da vítima e realizar o ataque de força bruta *offline*.

A partir do momento em que o atacante consegue acesso a um sistema a partir de uma senha descoberta, fica muito mais fácil de acessar outras máquinas e outros sistemas. Normalmente as pessoas reutilizam as senhas em várias máquinas e sistemas. Outra estratégia do atacante após invadir uma máquina é instalar um terminal de sessão legítimo utilizando qualquer senha que queira para o *login.*

3.3.3 LEVANTAMENTO DOS DADOS

Ao decidir atacar uma rede, ou um *host,* o primeiro passo do atacante é o levantamento dos dados. A engenharia social, apresentada anteriormente, auxilia no levantamento de dados subjetivos que poderão ser úteis na reconstrução da estrutura da vítima. Um levantamento objetivo pode ser feito através do *scanner* da rede. Consiste em utilizar técnicas variadas visando descobrir que tipo de máquinas há na rede, que tipo de sistemas operacionais estão rodando, que tipo de serviços estão habilitados, etc.

3.3.3.1 *footprinting*

O *footprinting* é o método utilizado para identificar os alvos de um ataque de rede. É o processo de levantamento das informações de uma rede através de fontes disponíveis publicamente. O objetivo é criar um mapa identificando sistemas e aplicações que poderão ser vitimas de um ataque.

Para realizar o *footprinting,* o atacante visita o *site* da empresa na busca de informações úteis. Utiliza mecanismos de pesquisa na Internet para encontrar informações valiosas. Utiliza o *whois* para encontrar informações sobre o domínio da empresa e os endereços IPs. Utiliza a ferramenta *dig* ou *nslookup* para fazer consultas de DNS sobre a empresa. Utiliza o *ping* (ICMP) para determinar a presença de *hosts* no range de endereçamento válido da empresa. Utiliza a ferramenta *tracert* para determinar os *hops* antes de chegar no endereço, tentando mapear os roteadores no caminho. Utiliza o *nmap* para fazer o *port scanner* e o *fingerprint* dos endereços[10].

3.3.3.2 *fingerprinting*

O *fingerprinting* é a técnica utilizada para determinar informações sobre o produto e a versão de sistemas operacionais e aplicações. Cada plataforma e versão possuem um jeito próprio e único de responder a determinadas requisições. Sendo assim, a resposta pode ser considerada como uma impressão digital do sistema. O atacante, com base na informação sobre produto e versão procura numa base de dados se há alguma vulnerabilidade conhecida naquele *software*. No *fingerprint* ativo, o atacante envia diferentes tipos de pacotes para o alvo e observa a resposta. no passivo, ele analisa o tráfego normal e consegue determinar qual é o sistema.

O atacante pode realizar o *fingerprinting* mandando pacotes de requisições normais para portas comuns e observando a resposta. Funciona com HTTP, FTP, SMTP, telnet. Ou então mandando dados inválidos para portas comuns e observando a mensagem de erro, que normalmente entregam o produto ou a versão. Esses dados inválidos podem ser caracteres especiais: “*”, “~”, ou pacotes IP, TCP ou ICMP inválidos[10].

3.3.3.3 *port scanning*

O *scanner* de portas consiste em uma série de técnicas utilizadas para saber quais serviços e portas estão disponíveis em um *host* esperando uma conexão. Por exemplo, se o atacante descobre que a porta 80 está aberta, provavelmente este *host* tem um servidor *Web* rodando, pois a porta 80 é o padrão TCP para escutar requisições do protocolo HTTP. Uma vez que o atacante sabe quais serviços estão disponíveis, ele procurará saber dos produtos e

versões utilizados e das contas de usuário do sistema em busca de alguma vulnerabilidade conhecida. Existem diversos tipos de *scanners* diferentes. As ferramenta mais conhecidas para realizá-los são *nmap, Nessus, SATAN*.

Como já foi discutido anteriormente, o protocolo TCP permite um tipo de *scanner* chamado de SYN *scan* ou TCP *scan*. Consiste em mandar segmentos TCP SYN em todas as portas (0 a 65.535) de um *host* e esperar pela resposta. Se vier um SYN+ACK, a porta está aberta. Caso contrário, está fechada.

Já foi discutido também o UDP *scan*, que consiste em mandar datagramas UDP vazios para todas as portas e esperar a mensagem ICMP *Destination Unreachable*.

Outra possibilidade é utilizar o *flag* FIN do TCP. No FIN *scan*, o atacante manda para todo o range de portas do *host* um segmento TCP com o *flag* FIN, para encerrar a conexão. Se a porta já estiver fechada, o atacante receberá um segmento TCP com o *flag* RST como resposta. Se a porta estiver aberta, o segmento FIN será descartado.

O *scan sweep* é realizado quando o atacante varre um range de endereços IPs na busca de uma porta específica aberta, por exemplo a porta 23 do *telnet.*

O *spoofed portcanning* é fazer um *scan* utilizando endereço IP de origem falso e capturar o tráfego de resposta. Entretanto ele só funciona se o alvo, o atacante e o endereço falso forem do mesmo segmento de rede. Ele pode ser útil contra IDSs (*Intrusion Detection Systems*) que permitem que uma determinada máquina possa fazer um *scan* sem gerar alerta[7].

Existe ainda uma possibilidade de utilizar o FTP remoto para realizar o *scanner*, mascarando assim a origem do *scan*. Essa técnica é chamada FTP *bounce*.

Há ainda um tipo de *scan* especial, que consiste em capturar todo o tráfego da rede e analisá-lo para determinar quais portas e *hosts* estão disponíveis. Esse *scan* exige que o atacante esteja no mesmo segmento do alvo e a placa de rede esteja em modo promíscuo. Uma ferramenta muito conhecida e especializada nessa captura é o *tcpdump*, que será explorada mais tarde.

3.3.4 EXPLORAÇÃO REMOTA DE VULNERABILIDADES

3.3.4.1 *exploits*

As falhas de segurança nos protocolos vistas anteriormente e falhas de segurança em aplicações podem permitir que o atacante consiga invadir a máquina alvo. Ele poderá conseguir um acesso privilegiado, ou a execução de algum código malicioso. Essas falhas são comumente chamadas de vulnerabilidades do sistema. Todos os dias vulnerabilidades são descobertas e divulgadas na Internet nos mais diversos *softwares*. Assim que surge a notícia da vulnerabilidade, pessoas mal intencionadas escrevem um código especialmente para explorar essas vulnerabilidades. Normalmente esse código é escrito em linguagem "C" e é chamado de *exploit*. Para não serem afetados por esses *exploits*, os usuários e administradores devem estar atentos às vulnerabilidades divulgadas e devem aplicar os *patchs* ou as atualizações que são recomendados e consertam as vulnerabilidades. Entretanto, existem casos em que as pessoas mal intencionadas são as primeiras a descobrir uma vulnerabilidade. Elas, então, criam o *exploit* antes da vulnerabilidade ser divulgada. São os chamados "*zero day exploits"*.

Além dos códigos, a exploração de vulnerabilidades também pode ser feita através de ferramentas. Por exemplo, existem ferramentas especializadas em DoS, para tentar derrubar o servidor ou ferramentas de força bruta para tentar quebrar a senha de administrador.

Uma vez que o atacante comprometeu o sistema alvo e conseguiu um acesso privilegiado ao computador, através da exploração de vulnerabilidades, seu próximo passo é instalar um *rootkit*. Uma série de ferramentas e *scripts* para automatizar algumas ações. Dentre elas, a instalação de *backdoors*, ou "porta dos fundos", uma porta aberta que permita que o atacante volte à máquina sem que o usuário perceba. Além disso, esses *rootkits* instalam capturadores de teclado, ferramentas de administração remota, *sniffers* para capturar tráfego e senhas, substituem arquivos do sistema que permitiriam que os *rootkits* fossem descobertos e apagam registros de *log* do sistema. Existem *softwares* especializados em descobrir *rootkits* na máquina, inclusive *kernel rootkits*, que se escondem no kernel do sistema operacional.

3.3.4.2 *backdoors*

O mecanismo de *backdoor* foi originalmente criado por programadores para permiti-los um acesso especial a seus programas para que eles pudessem consertar o código quando houvesse um *bug*. Muitas vezes era colocado de maneira maliciosa para conseguir

informações privilegiadas do usuário. Mais tarde, os "*hackers*" incorporaram o termo *backdoor* definindo como qualquer mecanismo que permita um acesso posterior de maneira transparente a um sistema comprometido, sem que ele precise explorar novamente a vulnerabilidade para obter acesso como da primeira vez. Se o dono do sistema descobrir a vulnerabilidade ou a intrusão, mas não descobrir o *backdoor*, o atacante ainda terá o acesso à máquina. Uma ferramenta bastante utilizada como *backdoor* é o *netcat*[10].

3.3.4.3 *vírus*

Os *vírus* são programas que conseguem se replicar, atacar outros programas e realizar ações maliciosas e não solicitadas como falha de sistema ou perda de dados. O vírus pode entrar no computador através de um disquete infectado, ou um anexo de e-mail, ou um arquivo baixado da Internet.

Os vírus podem ser classificados em vírus de setor de *boot*, de arquivo, de macro, múltiplo e polimórfico. Os vírus de setor de *boot* infectam a MBR (*master boot record*) do HD. Eles são carregados na memória a cada vez que o sistema é iniciado. Os vírus de arquivo se anexam a programas executáveis e são carregados toda vez que esses programas rodam. Os vírus de macro são escritos em linguagem macro, afetando aplicações como *Microsoft Word*. Os vírus múltiplos são uma combinação dos de setor de *boot* e os de arquivo. Os vírus polimórficos podem ter a forma de qualquer um dos anteriores, porém eles têm a capacidade de realizar uma mutação em seus próprios códigos para tornar mais difícil a detecção.

A maneira que se tem de evitar os vírus é ter um *software* anti-virus baseado em assinatura instalado, evitar abrir anexos de e-mail quando a origem não for confiável (e foi visto o quão fácil é forjar uma origem) e fazer *backups* do sistema continuamente. Quando um vírus é descoberto em um computador, é necessário isolá-lo da rede, reinstalá-lo e procurar pelo vírus em outras máquinas.

3.3.4.4 *worms*

Worms podem danificar uma máquina como se fosse um vírus, porém eles são mais poderosos. Eles têm a capacidade de se propagar para outros sistemas e máquinas na rede. Alguns simplesmente se replicam gastando a memória e a banda do usuário. Outros contém

código de vírus e podem corromper arquivos, roubar documentos, enviar e-mails ou deixar o sistema inoperante. Eles são como *exploits* automáticos viajando na rede, prontos a achar uma máquina vulnerável.

Para se replicar, eles utilizam diversas técnicas como: mandar cópia de si mesmo através de e-mail para todas as pessoas cadastradas no livro de endereço do usuário; execução remota, utilizando exploração de vulnerabilidades em protocolos ou aplicações. Um exemplo muito comum é a exploração do *buffer overflow,* um *bug* de estouro de memória que pode derrubar a aplicação e fornecer um acesso privilegiado ou execução de código; *login* remoto utilizando as tentativas de "adivinhar" a senha, já discutidas.

3.3.4.5 *trojans*

Um *trojan*, ou cavalo de tróia, como é conhecido, também é um programa escrito para causar danos ao usuário, como um vírus ou um *worm.* A diferença é que geralmente ele é transparente ao usuário e, disfarçando-se como um programa legítimo, não tem efeitos óbvios como corrupção de arquivos inoperância do sistema. Além disso, normalmente ele não se replica.

Os *trojans* normalmente são arquivos executáveis que fornecem controle do sistema ou roubam informações. Podem chegar por e-mail, ou serem implantados por um atacante ao conseguir invadir um sistema, ou mesmo vir junto com um arquivo baixado na Internet. Normalmente, ele vem camuflado. O usuário executa um jogo, ou assiste uma apresentação de fotos, vê um vídeo enquanto que o *trojan* é instalado em sua máquina. Quando ele for executar determinada aplicação, o *trojan* entra em ação e cumpre o seu propósito.

Existem diferentes tipos de *trojan*: os ladrões de senha procuram por senhas armazenadas no sistema e as mandam por e-mail ao atacante. Podem também chamar uma tela de *login* falsa quando determinado programa é executado para que o usuário poste sua senha e ela seja capturada e mandada ao atacante. É o caso, por exemplo, da maioria dos *trojans* que roubam senhas de banco; os *keystroke loggers,* ou capturadores de teclado capturam tudo que o usuário digita e manda por e-mail ao atacante; as ferramentas de administração remota permitem que os atacantes tenham controle total sobre o sistema e façam o que quiserem remotamente; finalmente os *zombies* ou zumbis, permitem que o atacante realize um ataque de DDoS, que será detalhado adiante[10].

Não é fácil detectar a presença de um *trojan*. Pode ser que monitorando as portas do sistema, com uma ferramenta como o *netstat*, ele possa ser detectado. Ou fazendo um inventário de todos os executáveis e comparando com uma lista feita previamente, quando o sistema estava "limpo". O melhor a se fazer ao descobrir um cavalo de tróia é reinstalar o sistema e recuperar o *backup*.

3.3.4.6 *adwares* e *spywares*

Vírus, *worms* e *trojans* são tipos de programas com o nome genérico de *malware.* Há ainda dois tipos de *malware* que causam menos danos que os anteriores: os *adwares e spywares*. Na verdade os *spywares* são um tipo de *adware*.

O *adware* é um *software* que se instala de forma transparente e sem o conhecimento do usuário, normalmente quando ele baixa e instala *softwares* da Internet. Depois de instalado, ele mostra propagandas enquanto o usuário navega na Internet.

Além de mostrar propagandas, o *adware* pode monitorar os hábitos de navegação do usuário mandando essas informações para um terceiro, por exemplo uma empresa de *marketing* que a partir dessas informações constrói as propagandas. Nesta situação, ele é chamado de *spyware*. Essas informações normalmente não identificam o usuário, porém esses *softwares* são vistos por muitos como invasores de privacidade, além de "gastar" a banda de Internet do usuário para mandar as informações.

Existem *softwares* especializados em achar e remover esses *malwares* do computador. Normalmente os *softwares* de anti-virus não se preocupam com eles.

3.3.5 NEGAÇÃO DE SERVIÇO

Até agora, diversas vezes no capítulo foi citado o ataque de DoS (*denial of service*) ou negação de serviço. Nessa seção esse ataque será mais detalhado e alguns exemplos apresentados.

O DoS explora uma vulnerabilidade do *software* ou do protocolo com o objetivo de parar ou degradar a qualidade de um serviço na Internet. Normalmente o endereço de origem de um ataque desse tipo é falso (*spoofing).* As estratégias de DoS são diversas, entre elas:

inundar a rede com tanto tráfego quanto for possível para que o alvo pare de funcionar; inundar a rede com um número absurdo de requisições de tal maneira que o servidor não consiga mais atender as requisições legítimas; interromper a comunicação entre o alvo e os clientes legítimos de alguma maneira.

3.3.5.1 derrubando um *link*

Realizar um ataque de DoS em um *link* de rede é muito simples. O atacante só precisa gerar mais pacotes do que o *link* pode suportar. Apenas o destino do pacote precisa estar correto. Os outros campos podem ser aleatórios. O ataque pode partir de apenas um *host* desde que o *link* dele seja maior do que o do alvo, ou de vários *hosts*, de maneira coordenada no que é chamado DDoS (*distributed denial of service*).

3.3.5.2 DoS de força bruta

O ataque *smurf*, discutido na seção 3.2.1.3 é um ataque do tipo DoS de força bruta. Ele manda um ICMP *echo request* para um endereço de *broadcast* e faz o *spoofing* da origem para o endereço do alvo. Todas as máquinas que responderem, responderão para o alvo e se a banda disponível for grande, afetará os serviços da máquina alvo.

O ataque Echo-Chargen utiliza duas portas UDP: 7 (echo) e 19 (chargen). O echo, quando recebe um pacote, ecoa de volta um *payload*. O *chargen* quando recebe um pacote devolve com uma *string* de caracteres pseudo-aleatórios. A estratégia é enviar datagramas da porta *echo* para a porta *chargen* de vários *hosts* na esperança que eles devolvam os datagramas para a porta *echo* e comece um ping-pong resultando em consumo de banda e *cpu*.

3.3.5.3 DoS sofisticados

O primeiro ataque de DoS que se tem notícia é o ataque TCP SYN apresentado na seção 3.3.3.2. Ele se baseia nas características de estabelecimento de conexão do protocolo TCP.

O ataque *teardrop* tira proveito da fragmentação do protocolo IP. Por exemplo, chega um fragmento de 20 octetos com *ID* 2 e *fragment offset* 0. Depois chega outro fragmento de 4

octetos com o mesmo *ID* 2 e *fragment offset 1*0. O primeiro fragmento gastou 20 octetos do datagrama, então o *fragment offset* do segundo fragmento teria que começar no 21. O sistema operacional então tenta retroceder de 20 para 10, calculando como número negativo. Os números negativos no sistema operacional podem ser traduzidos para números positivos muito grandes invadindo a área de memória de algum outro programa. Feito isso algumas vezes, o sistema será desativado. A seção 3.3.3.1 mostra também um ataque de fragmentação para negação de serviço.

O ataque *ping of death* ou ping da morte consiste em mandar um pacote ICMP grande demais, ou seja, maior do que 64KB. Esse datagrama será fragmentado em muitas partes. O truque é utilizar essa fragmentação de modo em que o último fragmento ainda contenha um *fragment offset* válido (menor que 65.535), mas um tamanho que faça o datagrama ultrapassar os 65.535. Como os sistemas normalmente não processam o pacote até que tenham recebido todos os fragmentos e tentado remontá-los, pode haver um estouro das variáveis internas de 16 bits causando congelamento do sistema, *dump* de kernel, reinicializações, etc[15].

3.3.5.4 negação de serviço distribuída

O ataque de DoS clássico é um ataque de *host* para *host*. Normalmente um *host* mais poderoso gera um tráfego grande para derrubar o outro *host*. O DDoS (*distributed denial of service*) realiza esse ataque de maneira ampliada. Muitos *hosts* coordenadamente atacam o alvo, chegando-se muito mais facilmente à condição de negação de serviço da vítima. O atacante às vezes não possui tantas máquinas à sua disposição. Então, ele pega “emprestado” máquinas de usuários comuns, transformando-as em “zumbis” e as utiliza no ataque.

A implementação de um DDoS é simples. Primeiramente o atacante roda ferramentas automáticas para encontrar *hosts* vulneráveis na rede. Uma vez encontrado, o *host* é comprometido e o atacante instala, por exemplo um *trojan* DDoS transformando o *host* em um zumbi que pode ser controlado remotamente por uma estação mestre. Algumas ferramentas populares de DDoS são a TFN, TFN2K, Trino e Stacheldraht, disponíveis na Internet. Uma vez que o atacante esteja com uma quantidade razoável de zumbis, ele manda um sinal da estação mestre para começar o ataque contra o alvo. O ataque é, normalmente um SYN *flood*, ou uma inundação de ICMP, ou outro tipo simples de DoS, mas o fato de centenas ou milhares de zumbis serem usados no ataque cria uma massa muito grande de tráfego que pode rapidamente consumir todos os recursos da máquina e a tornar indisponível na Internet[10].

Os zumbis utilizados no ataque de DDoS também podem servir para outros propósitos. Muitos atacantes criam verdadeiras redes de robôs, ou zumbis. Um dos objetivos é o ataque de DDoS, mas ela pode servir também para um *scanning* distribuído, ou qualquer outra atividade ilícita na qual o atacante não queira utilizar sua própria máquina. Os *trojans* e especialmente os *worms* são utilizados na criação desses robôs e os comandos da comunicação entre mestre e robôs são normalmente encriptados.

3.3.6 *MAN-IN-THE-MIDDLE*

Alguns livros de criptografia dividem os ataques em dois tipos. O ataque passivo e o ataque ativo. O ataque passivo é o ataque no qual captura-se o tráfego para análise. O ataque ativo engloba a maioria dos tipos que foram discutidos, como *spoofing*, *denial of service*, exploração.

Supondo que haja um tráfego normal entre dois *hosts* e um atacante consiga se posicionar exatamente no meio desses dois *hosts*. Teoricamente, ele terá a possibilidade de interromper esse tráfego, de interceptar passivamente esse tráfego, de modificar os dados passantes de uma máquina para outra, ou fabricar um tráfego falso para uma das máquinas. O ataque *man-in-the-middle* utiliza todas essas premissas e exige que o atacante esteja no mesmo canal de comunicação de ambas as partes e consiga capturar todo o tráfego.

Como foi comentado na seção 3.3.3.2, o atacante tentará seqüestrar a sessão TCP, derrubando a máquina de destino através de um DoS e respondendo à origem como se fosse o

destino. Também existe o caso do atacante estar no meio e manter a sessão respondendo a ambas as partes da conexão, o que é muito mais difícil de se realizar, pois a quantidade de pacotes duplicados e de confirmação será muito grande e os alvos poderão detectar o ataque. Existem ferramentas no mercado especializadas no *man-in-the-middle* para seqüestro do TCP: *Hunt, Juggernaut, IP-Watcher*.

O *man-in-the-middle* também pode ser utilizado contra o protocolo ARP. O atacante direcionará as mensagens para o seu endereço físico MAC, como se esse fosse o endereço legítimo de destino.

Outro protocolo sujeito ao ataque é o protocolo de criptografia *Diffie-Hellman* (DH). Nesse tipo de ataque, a origem pede uma chave pública ao destino para poder cifrar a mensagem. O atacante intercepta essa requisição e manda uma chave pública própria se fazendo de destino. A origem, então, troca a chave de sessão com o atacante e todas as mensagens cifradas poderão ser lidas pelo atacante. Ao mesmo tempo, o atacante se mascara como origem e requisita uma chave pública para o destino. Muda a chave de sessão e se comunica de maneira criptografada também com o destino.

3.4 PROTEÇÃO

Os mecanismos de proteção para uma máquina ou para uma rede são muitos e procuram cobrir os mais diversos tipos de vulnerabilidades possíveis, se possível de maneira preventiva, evitando o ataque antes que aconteça. Essa seção objetiva dar uma “passada” pela maioria desses dispositivos. Isso porque é impossível esgotar cada um dos assuntos a serem tratados a seguir devido à complexidade dos mesmos.

Os assuntos que serão discutidos, a princípio são: criptografia; autenticação, autorização e auditoria; segurança de *host;* ferramentas especializadas (anti-virus, anti-spam, anti-trojan, etc); *penetration test; firewall* e, por fim, sistemas de detecção de intrusão. O último mecanismo é mais diretamente ligado ao assunto da dissertação e será mais cuidadosamente detalhado com o objetivo de subsidiar o leitor para os próximos capítulos da dissertação.

3.4.1 CRIPTOGRAFIA

A criptografia é uma ciência matemática que tem como objetivo descobrir caminhos de assegurar a privacidade da comunicação entre partes. Sua história é longa: em 500 aC Hebreu utilizou substituição reversa de letras do alfabeto para escrever no "*Book of Jeremiah*". O uso militar sempre foi muito comum, Julio César utilizava um algoritmo de deslocamento de caracteres para criptografar mensagens militares. Na segunda guerra, a Alemanha utilizava uma máquina de criptografia, Enigma. Os aliados conseguiram quebrar a criptografia de Enigma contribuindo com sua vitória sobre o nazismo[10].

Justamente com o fim da guerra e o surgimento dos computadores, a criptografia avançou muito e surgiram os sistemas de criptografia modernos, os cripto-sistemas. São algoritmos passo a passo que utilizam matemática complexa e objetivam transformar texto claro em texto cifrado, ou seja, humanamente incompreensível.

Um cripto-sistema obedece a uma metodologia, que inclui um ou mais algoritmos matemáticos de criptografia, chaves de criptografia, um sistema de gerenciamento de chaves, o texto original e o texto cifrado. A metodologia é encarregada de cifrar o texto original utilizando uma chave e do outro lado da comunicação decifrá-lo utilizando uma chave que pode ou não ser a mesma.

A metodologia baseada em chaves combina um algoritmo de criptografia com uma chave para criar o texto cifrado. A segurança maior reside no segredo da chave de criptografia mais do que no segredo do algoritmo. O grande problema é como criar e mover as chaves em um canal de comunicação inseguro. Outro problema é a autenticação. Como saber que quem criou a chave e cifrou o texto é realmente quem está dizendo ser. Existem dois diferentes tipos de metodologia que implementam uma metodologia baseada em chave: simétrica ou chave privada; assimétrica ou chave pública.

3.4.1.1 Criptografia simétrica

A criptografia simétrica é a metodologia na qual origem e destino utilizam a mesma chave criptográfica. Adicionalmente, é utilizada a função *hash* para garantir a integridade da mensagem.

A função *hash* ao ser aplicada transforma, matematicamente, uma mensagem de qualquer tamanho em uma pequena mensagem de tamanho definido. A mensagem resultante é chamada de assinatura digital da mensagem original, pois apenas aquela mensagem original irá resultar na pequena mensagem resultante (embora tenha sido provado a possibilidade de fabricar a mensagem *hash* resultante através de outras mensagens originais no famoso algoritmo de *hash MD5*).

A origem aplica a função *hash* na mensagem, pega o resultado e anexa na mensagem original. Cifra tudo junto com a chave criptográfica e manda a mensagem cifrada. O destino recebe a mensagem, decifra com a mesma chave criptográfica que já era conhecida e pode ver o texto original e o *hash*. Então ele aplica o mesmo algoritmo de *hash* ao texto original e compara com o *hash* recebido. Sendo igual, ele garante que a mensagem não foi modificada.

Em nenhum momento houve troca da chave pelo canal inseguro, nem tão pouco o destino questionou se realmente a origem é confiável.

Alguns dos algoritmos mais utilizados de criptografia simétrica são: DES, 3DES, IDEA, RC5, *blowfish*. De *hash* são: MD5, SHA.

3.4.1.2 Criptografia assimétrica

A criptografia assimétrica utiliza um par de chaves para realizar o processo cifragem-decifragem. Esse par é criado em conjunto, sendo que uma chave é utilizada para cifrar enquanto que a outra é utilizada para decifrar. A chave privada só é conhecida por seu próprio dono. A chave pública é de domínio público e é distribuída por uma entidade confiável, a AC (Autoridade Certificadora). As chaves são criadas e distribuídas em um formato padrão, certificado digital X.509. A arquitetura de funcionamento da AC é conhecida como PKI (*Public Key Infrastructure*).

Esse par de chaves é criado na AC e distribuído aos destinatários. Normalmente, o tamanho dessas chaves precisa ser muito maior do que a chave simétrica, pois elas serão alvo de ataques de força bruta tentando quebrá-las. A chave simétrica é pequena pois muda constantemente, já o par de chaves assimétricas não necessita mudar. A metodologia assimétrica utiliza, na verdade, ambas as criptogafias: simétrica e assimétrica. Sendo a chave simétrica menor, ela é utilizada para cifrar a mensagem. O par assimétrico é utilizado apenas para cifrar a chave simétrica do momento.

O primeiro passo da metodologia assimétrica é criar e distribuir as chaves assimétricas de forma segura. A chave privada é entregue ao seu dono pela AC. A chave pública é armazenada em um banco de dados na AC. A AC tem que ser confiável. Se ela estiver comprometida, toda a estrutura estará também. O transmissor cria uma assinatura digital do texto, aplicando uma função *hash* e criptografa essa assinatura utilizando a sua chave privada, anexando o resultado ao texto original. O transmissor cria uma chave simétrica, chamada de chave de sessão, que será utilizada apenas para essa transmissão e cifra o texto resultante da operação anterior. Então, ele pede à AC a chave pública do receptor (esse passo é vulnerável ao ataque *man-in-the-middle*). A chave pública do receptor é assinada digitalmente pela AC, ou seja, a AC utiliza sua chave privada para criptografar a chave pública do receptor, garantindo que foi ela mesma quem cifrou. Os dois lados precisam previamente ter a chave pública da AC. O transmissor decifra a chave pública do receptor com a chave pública da AC e utiliza a chave pública do receptor para cifrar a chave de sessão. O transmissor anexa a chave de sessão, cifrada com a chave pública do receptor, ao texto resultante e envia. O receptor recebe a mensagem, desanexa a chave de sessão e, utilizando sua chave privada, decifra a chave de sessão. Com a chave de sessão ele decifra o texto resultante. Após isso, ele desanexa o *hash* do texto original. Decifra o *hash* cifrado utilizando a chave pública do transmissor (antes ele pede a chave pública do transmissor à AC). Agora que ele tem o texto original e o *hash* resultante, ele aplica a função *hash* ao texto original e compara as duas funções. Sendo iguais, a integridade da mensagem está garantida.

A grande desvantagem desse sistema está na confiança cega se precisa ter na AC. Se um atacante conseguir dominá-la, tudo está perdido. Outra grande preocupação que permanece é como distribuir o par de chaves assimétrico. Ele é gerado na AC e precisa ser entregue ao dono. A maneira mais segura que há é a entrega pessoal, cara a cara. Porém, muitas vezes pela facilidade, ela é feita pela rede interna da empresa, assumindo-se que ela é segura.

Alguns dos algoritmos mais utilizados de criptografia assimétrica são o RSA, o ECC e o ElGammal.

3.4.2 AAA (AUTENTICAÇÃO, AUTORIZAÇÃO E AUDITORIA)

Como já foi citado na introdução do capítulo, o AAA constitui um ponto chave na segurança de uma rede. A autenticação é a primeira linha de defesa de qualquer rede e previne a entrada de usuários não autorizados ao sistema. Ela possui diversos desafios como coletar os dados de autenticação, transmitir os dados seguramente e garantir que a pessoa que está utilizando o sistema ainda é a pessoa que originalmente se autenticou. A autenticação pode ser baseada em três princípios que podem ser utilizados sozinhos ou em conjunto: em algo que o usuário sabe; em algo que o usuário possui; em algo que o usuário é.

Na autorização baseada em algo que o usuário sabe, normalmente utiliza-se a identificação do usuário e uma senha para autenticação. Os problemas relacionados à senha já foram discutidos neste capítulo. Outra possibilidade é a utilização de uma chave criptográfica, ou de um certificado digital atestado por uma AC confiável assegurando a autenticidade do usuário.

A autorização baseada em algo que o usuário possui usualmente é utilizada junto com a autenticação baseada no que o usuário sabe e provê uma segurança a mais para o usuário. Nesse método, é utilizado um dispositivo físico para autenticação. Esse dispositivo é chamado de *token*. Há os *tokens* de memória e os *smart tokens*. Os *tokens* de memória são dispositivos que guardam, mas não processam informação. Necessitam de dispositivos especiais de leitura e escrita para acessarem e escreverem a informação. Essa informação normalmente é a identificação do usuário. Aliado ao token de memória normalmente a autenticação exige uma senha para evitar que apenas com o roubo do dispositivo físico o atacante possa ganhar acesso ao sistema. Esse tipo de dispositivo é muito barato e permite um nível de segurança muito bom, aliado à senha. O ideal é unificar esse dispositivo com autenticação física de entrada e saída no ambiente, obrigando o usuário a desconectar da máquina quando for sair. A geração de *logs* também é facilitada com a posse desse dispositivo. Porém, ele possui algumas desvantagens como a necessidade de outro dispositivo para leitura e processamento das informações. A possível perda do dispositivo acarretará “dor de cabeça” ao usuário, que poderá ficar descontente com mais essa “dificuldade” na hora de acessar uma máquina. O *smart token* funciona exatamente da mesma maneira, mas possui circuitos integrados a ele que permitem que ele processe, escreva e leia as informações. Permitem também o armazenamento de certificados digitais.Além disso, permitem a criação de protocolos de autenticação mais sofisticados como geração dinâmica de senha ou protocolo criptográfico de desafio-resposta com geração de números aleatórios. Esses protocolos criam uma senha por

acesso, aumentando a segurança. Exemplos dos *tokens* de memória são os cartões de banco e de crédito, dos *smart tokens* são os *smart cards*.

Por fim, a autenticação baseada no que o usuário é, é o mais seguro e mais caro dos métodos. São as autenticações biométricas, que utilizam uma característica única da pessoa para garantir a sua autenticidade. Por exemplo a impressão digital, ou o padrão de voz, ou a letra, a íris, etc.

Em qualquer um desses métodos o grande desafio da autenticação é sua administração. Por exemplo, as senhas têm que possuir número mínimo, data de expiração, política de revogação. Tem que se designar uma pessoa de confiança responsável por administrar e acompanhar as políticas de autenticação de cada sistema. Existe um conceito de autenticação chamado de *single-sign-on* que significa autenticar-se apenas uma vez e ter acesso a todos os sistemas necessários. Do ponto de vista administrativo é maravilhoso. Administrar apenas uma autenticação e possuir os *logs* centralizados de cada operação. Do ponto de vista de segurança, pode ser perigoso. O atacante apenas terá o trabalho de entrar uma vez e conseguirá acesso a todo o sistema. É desejável que o *single-sign-on* seja feito apenas se o método de autenticação for o mais seguro possível (autenticação biométrica por exemplo).

A autorização é o processo de garantir direito de acesso a determinado recurso. O processo de autorização para acesso a recursos em uma rede geralmente é dividido em dois tipos: no baseado em função, os usuários são divididos em funções lógicas e os membros de cada função compartilham os mesmos privilégios; no baseado em recursos, cada recurso é assegurado utilizando-se uma lista de controle de acessos (ACL) que determina qual usuário pode acessar ao recurso e quais ações ele pode tomar.

A auditoria é o processo de análise de *logs* de autenticação e autorização. Caso ocorra algum problema, tem que estar devidamente registrado quem acessou o sistema, que dia e que horas acessaram o sistema, quais ações ele tomou, etc. Muitas vezes em sistemas muito complexos, onde o usuário precisa se autenticar em diversos subsistemas diferentes, a auditoria se torna uma tarefa extremamente complexa. Para auxiliar nessa análise, existem as ferramentas de correlação de *logs*, que conseguem buscar os registros de diversos sistemas diferentes e correlacionar, rastreando o caminho realizado pelo usuário.

3.4.3 SEGURANÇA DE *HOST*

Algo já foi dito desse assunto na introdução do capítulo. A segurança de *hosts* engloba assuntos importantes de segurança como o *hardening*, as ferramentas especializadas (*anti-virus, anti-spyware, anti-spam*, etc) e a política de *advisory* e *patch management*.

3.4.3.1 *hardening*

O *hardening,* ou blindagem do sistema operacional se refere a uma combinação de técnicas com o propósito de aumentar a segurança dos *hosts* contra ataques. As máquinas, que normalmente necessitam de *hardening* são aquelas que ficarão expostas diretamente à Internet, ou seja, os *bastion hosts*. São servidores web, servidores de e-mail, *firewalls*, servidores de DNS, etc. O processo de blindagem do sistema depende da plataforma utilizada.

De maneira geral, a blindagem consiste em desabilitar ou remover qualquer componente que tenha sido instalado por padrão, mas que não faça parte da função do *host*; desabilitar serviços de rede desnecessários para simplificar a configuração do servidor e prover apenas os serviços necessários para os clientes; acrescentar controle de acesso em componentes críticos do sistema como *dlls,* arquivos de configuração, registros do sistema e outros alvos potenciais de ataque; habilitar criptografia de senhas e arquivos importantes que não venha habilitada por padrão; configurar políticas de segurança para restringir acesso a funções críticas do sistema; utilizar checagem de arquivos de sistema e trilha de processos para registrar em *log* qualquer atividade suspeita[10].

A blindagem normalmente é realizada por meio de *script* específico para cada sistema. Faz-se uma instalação limpa com o servidor desconectado da rede e aplica-se o *script* construído. O *script* não é uma “receita de bolo” claramente definida e precisa sempre estar sendo testado e melhorado do ponto de vista de segurança e de gerenciamento.

3.4.3.2 ferramentas especializadas

São ferramentas que têm o objetivo específico de achar e acabar com tipos de *softwares* ou tipos de ataques específicos. A mais utilizada e tradicional é a ferramenta de

anti-virus. Mas também existe o anti-trojan, o anti-spyware, o anti-spam. De maneira mais geral, pode-se afirmar que essas ferramentas são tipos específicos de sistemas de detecção de intrusão especialistas e funcionam de maneira idêntica aos IDS (*Intrusion Detection Systems*) que serão vistos mais tarde.

Em todos os casos, basicamente dois tipos de análise podem ser feitas isoladamente, ou em conjunto: detecção baseada em assinatura ou detecção baseada em comportamento. A detecção baseada em assinatura é a tradicional e procura por "assinaturas" ou partes de código conhecidas que foram detectadas e catalogadas como maliciosas. É uma análise rápida e certeira quando se trata de uma ameaça conhecida, porém falha na hora de detectar novas ameaças, até que a base de assinaturas seja atualizada e a nova ameaça seja nela cadastrada. A detecção baseada em comportamento utiliza a política para indicar qual tipo de comportamento pode indicar uma ameaça. Se for detectada uma ação que viola uma política, como por exemplo um código tentando acessar o livro de endereços para se replicar, o *software* previne e pode até isolar o código numa caixa separada até que o administrador decida o que fazer com ele. A vantagem desse sistema é que ele pode detectar novas ameaças mesmo que a assinatura delas não seja cadastrada. A grande desvantagem é a quantidade de falso-positivos que podem ocorrer, ou seja, a detecção de códigos legítimos que se comportavam como se fossem uma ameaça.

3.4.3.3 *advisory* e *patch management*

Os *advisories* são avisos de seguranças para alertar os usuários sobre vulnerabilidades que foram descobertas em sistemas operacionais e aplicações. Os *advisories* podem ser publicados por diferentes fontes incluindo agências governamentais, organizações públicas ou privadas de segurança, ou empresas vendedoras de *software.* Os avisos tipicamente são postados em *sites* ou em listas de e-mail para que os administradores estejam sempre visitando os *sites* ou participando da lista para que possam manter seus sistemas atualizados da melhor maneira possível. Uma fonte interessante de *advisory* que mantém avisos e consertos para várias plataformas diferentes é o CERT/CC (*CERT Coordination Center*) localizado no Instituto de Engenharia de Software na Universidade de Carnegie Mellon (*www.cert.org*).

O *patch* é um código de "conserto" de *software* liberado pelos fabricantes ou criadores do *software* para corrigir falhas em seus produtos que podem acarretar em perigo ou perda de

dados ou falta de confiança no produto. No caso de *patchs* de segurança, as falhas permitem que atacantes comprometam o sistema e os *patchs* são lançados para consertar essas vulnerabilidades. A aplicação de *patchs* é uma prática necessária e prioritária a todos que estão conectados à Internet e para auxiliar as companhias a manter seus sistemas atualizados, o SANS Institute e o FBI trabalham juntos para manter um top-20 das principais vulnerabilidades de segurança, apontando o caminho para um detalhamento maior e os possíveis *patchs*. O sítio é www.sans.org/top20.

3.4.3.4 Teste de penetração

O *penetration testing* é o processo de emulação de atacantes quando acessam a segurança de determinadas máquinas ou redes alvos. Normalmente é realizado por empresas grandes, bem consolidadas e confiáveis. Elas realizam uma análise de segurança de redes através de poderosos *scanners* e posterior *exploit* de vulnerabilidades. O propósito é seguir exatamente os mesmos passos que o atacante faria e classificar a o alvo de acordo com o risco que ele está correndo no mundo real. De posse dessa classificação internacionalmente reconhecida e com um detalhado relatório das vulnerabilidades e riscos em mãos, a empresa alvo poderá planejar de maneira mais precisa seus investimentos em segurança da informação. Por exemplo na compra de uma solução de *firewall* ou *IDS.*

3.4.4 FIREWALL

O *firewall* é a figura central da segurança de uma rede. O mais comum é que se pense no *firewall* como uma caixa central designada para filtrar o tráfego da Internet. Não deixa de ser verdade. Entretanto, o *firewall* existe em diversas outras formas. Um roteador que faz filtro de pacotes, um *switch IP,* um filtro incorporado ao *kernel* do UNIX ou do *Windows.* Quer dizer, um *firewall* é qualquer dispositivo, *software,* arranjo ou equipamento que consiga filtrar determinado tráfego. Dependendo do tipo de aplicação e utilização, uma rede vai precisar de vários tipos de *firewalls* diferentes em vários pontos estratégicos.

Os *firewalls* conseguem filtrar em diferentes níveis na pilha TCP/IP e são classificados por essa característica. Existem quatro principais categorias: filtro de pacotes; *gateway* de

circuito; *gateway* de aplicação; e *stateful inspection*. Cada um desses é caracterizado pelo nível do protocolo que ele controla. Porém, na prática, todos eles necessitam de informações dos outros protocolos. Na verdade, eles costumam funcionar em conjunto e não de forma isolada. *Firewalls* comerciais, como por exemplo o *Check Point*, utilizam a combinação dos quatro tipos em conjunto e adicionalmente utilizam tecnologias proprietárias como *malicious code protector*, provendo um altíssimo nível de defesa.

É desejável que toda rede possua um *firewall* no qual todo o tráfego da entrada para a saída, ou vice-versa, precise necessariamente passar pelo *firewall*, ou seja, o acesso à rede local seja apenas pelo *firewall*. Além disso, apenas tráfego autorizado e definido pela política de segurança seja permitido passar. E o *firewall*, em si, precisa ser totalmente imune aos ataques, precisa possuir um sistema confiável e seguro.

A principal função de um *firewall* é manter o controle centralizado da segurança de uma rede, mantendo usuários não autorizados fora da rede protegida, proibindo serviços vulneráveis de entrar ou sair da rede e protegendo a rede contra vários tipos de IP *spoofing* e ataques de roteamento . Esse controle ocorre em diversos níveis: o controle de serviço determina o tipo de serviço que pode ser acessado por dentro ou por fora. O *firewall* pode filtrar o tráfego com base no endereço IP e porta TCP, pode prover um *proxy* que interpreta cada requisição de serviço antes de deixar passar ou pode fazer ele mesmo o papel de servidor, dependendo do tipo de *firewall*; o controle de direção determina a direção na qual uma requisição de serviço é inicializada e permitida; o controle de usuário controla o acesso ao serviço de acordo com o usuário (interno ou externo). O *firewall* pode ser utilizado como ponto final de um túnel criptorgrafado VPN (*Virtual Private Network*), utilizando, por exemplo, o protocolo IPSEC. Dessa maneira, o usuário externo que vier pela VPN pode ser tratado como um usuário local; por fim, o controle de comportamento controla como um serviço é usado. Por exemplo, um *firewall* pode filtrar e-mails para eliminar *spams* [14].

Outra função importante do *firewall* é a segmentação de redes em subredes. Ele é utilizado para segmentar, por exemplo, departamentos diferentes. Dessa maneira, não há como um usuário de um departamento acessar recursos de outro departamento a não ser que tenha uma regra específica no *firewall* permitindo. Essa solução protege uma subrede da outra e mantém o controle central da rede. Ele segmenta também, como já foi mencionado, a rede interna da rede DMZ, onde se localizam os *bastion hosts*, ou seja, os servidores Internet.

Além disso, o *firewall* é capaz de fazer tradução de endereços (NAT); prover uma

administração centralizada e uma monitoração de eventos relacionados a segurança, sendo ponto de auditoria; implementar alarmes de segurança; servir como plataforma para implementação de túneis IPSec; prover autenticação baseada em identificação e senha, senhas únicas para cada acesso, ou baseada em certificados digitais; prover roteamento ou encaminhamento de pacotes, dependendo do caso; implementar um DNS ; e outras funções que vão sendo agregadas ao longo do tempo.

O que um *firewall* não é capaz de fazer é prevenir que usuários locais tenham acesso externo não autorizado através de modems, nem a transferência de programas legítimos infectados por códigos maliciosos (apesar de não estar muito longe de acontecer).

Há muito mais detalhes a respeito de *firewalls* que não poderão ser explorados. A prática na configuração e manutenção dos mesmos ajuda muito a compreender o quanto uma rede depende do *firewall* atualmente. A seguir será explicado brevemente a respeito de cada tipo de *firewall*.

3.4.4.1 Filtro de pacotes

Este é o tipo mais simples de *firewall* e pode ser implementado em um roteador. Ele é configurado com uma série de regras para permitir, rejeitar ou descartar pacotes entrantes ou saintes da rede baseados no endereço IP ou no número da porta. Trabalham na camada de rede (ou Internet), porém precisam saber o número de porta. Combinados com a tradução de endereços, NAT, permitem uma primeira linha de defesa. São extremamente rápidos e funcionam na velocidade da linha de dados.

O primeiro parâmetro a se configurar neste tipo de *firewall* é a política padrão. Pode-se escolher permitir todos os pacotes a não ser que seja um expressamente proibido. Essa prática, porém, é muito perigosa e o comum é que se faça o contrário. A regra padrão é rejeitar ou descartar todos os pacotes, a não ser que ele esteja expressamente permitido. Desta maneira, à medida que for surgindo a necessidade, os serviços vão sendo liberados.

A tabela 3.1 mostra um exemplo simples de configuração de firewall do tipo filtro de pacotes. O *firewall* no caso está habilitado a enviar e-mails apenas, bloqueando qualquer outro tipo de serviço. Entretanto é necessário uma segunda regra, para permitir a entrada dos *flags* de confirmação ACK para que a conexão SMTP seja completa.

ação	*origem*	*porta*	*destino*	*porta*	*flags*	*comentário*
permitir	interno	*	*	25		Pacotes da rede interna para a porta SMTP
permitir	*	25	*	*	ACK	Confirmações ACK para SMTP
descartar	*	*	*	*	*	regra padrão

Tabela 3.1 – filtro de pacotes

3.4.4.2 *Gateway* de circuito

Este tipo de *firewall* escuta uma requisição de estabelecimento de conexão TCP e decide aceitar ou rejeitar baseado no número de porta. Quando o *firewall* aceita a conexão, a sessão é estabelecida entre o *firewall* e o *host* remoto. Então o *firewall* estabelece uma sessão separada com o *host* interno e conduz a comunicação entre as duas sessões utilizando um circuito de conexão interna que ele estabelece consigo mesmo. Desta maneira, os pontos finais não realizam a comunicação diretamente e sim através do *firewall.*

3.4.4.3 *Gateway* de aplicação

Um *gateway* de aplicação é também chamado de servidor *proxy* de aplicação. Ele é similar ao *gateway* de circuito, porém ele também pode filtrar o tráfego baseado no protocolo de camada de aplicação, como por exemplo um HTTP. Enquanto o *gateway* de circuito permite qualquer protocolo estabelecer uma conexão *proxy* na porta 80, o *gateway* de aplicação somente permitirá o tráfego HTTP legítimo, bloqueando qualquer outra aplicação, como por exemplo P2P, ou seja, programas de compartilhamento de arquivos que tentam utilizar a mesma porta 80.

O *gateway* de aplicação também registra o tráfego, faz autenticação, converte protocolos e algumas outras ações úteis. Entretanto, são mais complexos que os outros tipos de *firewall* para configurar. É necessário configurar cada protocolo de aplicação para cada tráfego que será permitido passar. Normalmente, um *software* ajuda nessa tarefa.

3.4.4.3 *Stateful Inspection*

É uma tecnologia desenvolvida inicilamente pela empresa CheckPoint, que criou um

produto chamado Firewall-1. Logo após, transformou-se em padrão de mercado tanto para firewalls comerciais, quanto para firewalls de código aberto. A diferença básica para os demais *firewalls* é que ele mantém uma memória das conexões TCP e cria pseudo-conexões UDP. Quer dizer, quando efetua-se uma conexão, o cliente envia um SYN e, se for aceita, recebe um SYN-ACK e o *firewall* guarda numa tabela temporária a origem, a porta e o destino da conexão. Quando o cliente for acessar novamente o serviço, o *firewall* apenas consulta a tabela temporária, possibilitando maior performance e segurança. A tabela também é criada e consultada quando chegam pacotes UDP.

Outra técnica utilizada é na autenticação. O cliente só precisa autenticar uma vez e essa autenticação será utilizada novamente para sessões deste cliente através do *firewall*.

O stateful inspection mantém informações de todos os 7 níveis do modelo OSI e oferece um alto grau de segurança e desempenho. É imprescindível para os *firewalls* modernos.

3.4.5 SISTEMAS DE DETECÇÃO DE INTRUSÃO

Os sistemas de detecção de intrusão são mais uma camada de proteção necessária às redes e *hosts* e são basicamente ferramentas (*hardware* ou *software*) de monitoração do comportamento do sistema. Havendo uma tentativa de invasão ou outra atividade maliciosa, o sistema tem o papel de relatar ao dono através de alarmes, ou mesmo agir de forma a deter o ataque, caso tenha sido configurado para isso.

Os IDSs (*Intrusion Detection Systems*), como são comumente conhecidos, podem monitorar um *host*, uma rede ou várias redes de acordo com seu tipo e sua arquitetura. O IDS de *host* normalmente monitora os programas e arquivos de uma máquina, observando se não estão sendo alterados. O IDS de rede normalmente observa e captura todo o tráfego da rede, à procura de pacotes que indiquem um ataque. Para que possam enxergar todo o tráfego, os "sensores" do IDS geralmente são posicionados em uma porta do *switch* que é configurada para receber todo o tráfego, ou mesmo na saída do roteador.

O IDS pode ser baseado em assinatura, isto é, ter uma base de tipos de ataques e como eles ocorrem cadastrada. Toda vez que um pacote chegar, ou que um arquivo ou programa for alterado compara-se com a base para saber se pode ser um ataque cadastrado. Ou pode ser baseado em comportamento, ou anomalia, isto é, o sistema "se acostuma" com o tráfego

normal ou a utilização normal dos programas e arquivos e quando sai do padrão, ele acusa que pode ser um ataque. Nos capítulos seguintes será visto que esse tipo de detecção por comportamento, ou anomalia é justamente a proposta do trabalho e que a inteligência artificial, através das redes neurais, pode entrar em ação para julgar sobre ser ataque ou não o que foi analisado.

Quando um alerta é gerado e na verdade não ocorreu nenhuma atividade maliciosa, ou seja, a atividade era normal, diz-se que ocorreu um falso positivo nesse alerta. Por outro lado, quando uma atividade maliciosa é bem sucedida e nenhum alerta é gerado, diz-se que ocorreu um falso negativo. Os falso positivos são um tormento para os analistas, pois tomam tempo e recursos. Pior ainda são os falso negativos, que permitem a entrada do inimigo sem que o IDS tenha cumprido sua missão.

3.4.5.1 Detecção em *host*

O IDS baseado em *host* (HIDS) monitora por ataques no sistema operacional, aplicações ou no *kernel*. Para fazer o seu trabalho, os HIDS precisam ter acesso privilegiado à máquina. Precisam acessar *logs* de auditoria, mensagens de erro, serviços, aplicações e qualquer outro recurso disponível para monitorar a máquina. Eles precisam saber como são dados normais e dados anormais de aplicação e por isso conseguem monitorar os dados de aplicação quando estão sendo decodificados e manipulados pelas aplicações.

Os HIDS são bastante precisos em determinar detecção de intrusão. A quantidade de falsos positivos gerados por eles é bem menor do que os NIDS (N*etwork Intrusion Detection Systems*). Os HIDS têm um conhecimento profundo do *host* que estão protegendo, sabem exatamente que tipo de atividade é normal ou não, um tráfego de rede pode passar por um NIDS desapercebido e ao chegar ao *host* ser detectado pelo HIDS. Em outras palavras, o HIDS pode descobrir ataques que o NIDS não consegue[6].

A desvantagem dos HIDS está no fato de terem uma visão muito limitada da topologia da rede e não conseguir detectar um ataque em algum outro ponto da rede. Isto é, se o alvo for uma máquina vizinha, que não tem um HIDS instalado, ele não conseguirá detectar. Mesmo que a máquina comprometida acesse a máquina protegida de maneira legítima, o HIDS não detectará. O HIDS tem que estar instalado em todas as máquinas, ou no maior número possível em uma rede. O que o torna uma solução cara financeiramente. Além disso, tem que

possuir versões para todos os tipos de sistema operacional existentes na rede, o que também é complicado. Existe ainda a possibilidade do HIDS, em um ataque bem sucedido ser desabilitado antes que ele possa detectar o ataque, ou seja, antes que ele leia os arquivos de *log* ou de erro.

3.4.5.2 Detecção na rede

O IDS de rede (NIDS) é um, ou vários dispositivos localizados em áreas estratégicas da rede e que conseguem enxergar o tráfego ao mesmo tempo em que ele está passando. São mais populares que os HIDS pois conseguem, com poucos dispositivos, proteger toda a infra-estrutura de rede. Os NIDS fornecem uma visão ampla sobre o que está acontecendo com a rede, sendo mais fácil identificar onde está o problema.

Em relação aos HIDS, os NIDS são mais resistentes à parada, ou interrupção. não dependem de *host* e podem estar em um *hardware* especialmente construído para ele, com o *hardening* em dia, muito mais difícil de serem destruídos por um atacante do que um HIDS. Além do mais, podem guardar as informações importantes em outra máquina na rede, dificultando o atacante de remover as evidências de um ataque.

Entretanto, o NIDS tem algumas desvantagens inerentes às suas características. Além do número mais elevado de falso positivos já citado, o NIDS exige estar em uma máquina parruda o bastante para agüentar todo o tráfego da rede, processando e interpretando em tempo real. Por isso, ele deve ser posicionado com cuidado na rede para evitar situações de perda de pacotes. De preferência receber os pacotes “duplicados” de algum outro dispositivo para analisar. Outro problema são as técnicas de evasão utilizadas pelos atacantes para burlar os NIDS. Por exemplo, a fragmentação de pacotes IP é um problema para os NIDS por causa da ordem de refragmentação diferente de cada sistema operacional. Quando um fragmento se sobrepõe a outro, dependendo da ordem de refragmentação, o resultado vai ser diferente e o NIDS freqüentemente não consegue lidar com as possibilidades. Outro exemplo é o tráfego criptografado, que não consegue, por razões óbvias, ser lido pelo NIDS. O atacante então criptografa seu código malicioso e manda já com garantias que será descriptografado do outro lado.

3.4.5.3 Detecção por assinatura

A detecção por assinatura identifica ataques que se parecem com a representação de intrusão que o IDS guarda em sua base de dados. Em um NIDS, o tráfego chega, ele compara o tráfego a uma base de assinaturas previamente cadastrada e se as características forem as mesmas, um alarme de intrusão é gerado. Por exemplo, um ataque de DoS utilizando pacotes ICMP maior do que o normal pode ser facilmente cadastrado na base de assinaturas. Todo ICMP maior do que 10000 bytes será considerado um ataque. Como dificilmente chegará um ICMP acima de 10000 bytes, a não ser que seja um ataque de DoS, o número de falso positivos dessa assinatura será baixo.

A detecção por assinatura é a técnica mais precisa para detectar ataques conhecidos. Quase todo o tipo de tráfego malicioso pode ser detectado com uma única assinatura. Poucos são os tipos de ataque comprovadamente elusivos para esse tipo de detecção (Koziol, 2003).

Entretanto, como não poderia deixar de ser, limitações existem neste tipo de detecção. Ela não tem conhecimento sobre a intenção do tráfego que coincide com a assinatura. Portanto, mesmo que o tráfego seja normal, ela emitirá o alarme, gerando um falso positivo. Além do mais, ela requer um conhecimento prévio dos ataques para poder ser construída a assinatura. Isso significa que ela não tem a capacidade de monitorar ataques desconhecidos ou ataques sem uma assinatura precisa. Em alguns casos, muda-se apenas um *bit* de um ataque conhecido e a detecção por assinatura irá falhar. À medida que surgem novos ataques, novas assinaturas são criadas e a base vai crescendo a cada dia. Como já foi visto, a quantidade de vulnerabilidades é enorme e só tende a crescer. Além disso, a largura de banda também cresce muito a cada dia. Como o IDS baseado em assinaturas precisa comparar cada pacote com sua base, vai chegar o dia em que computacionalmente será inviável para a maioria dos casos, pois o IDS começará a perder pacotes e gerar uma grande quantidade de falso negativos.

3.4.5.4 Detecção por anomalia

Na detecção por anomalia ou comportamento, existe a fase de treinamento dos dados e a fase de detecção propriamente dita. Durante a fase de treinamento, o uso “normal” do sistema é realizado. A partir daí, qualquer atividade que esteja fora do “normal” será relatada como uma possível intrusão, gerando um alerta. Por exemplo, em um HIDS, normalmente um usuário costuma se autenticar no período comercial (8:00-18:00). Certo dia, ele se autentica as

4:00 da manhã. O sistema gerará um alerta de intrusão, pois o comportamento diferiu do normal.

Este tipo de detecção resolve alguns problemas da detecção por assinatura, como por exemplo a necessidade de conhecer o ataque antes que ele ocorra e a necessidade de ter uma grande base de dados para comparação dos ataques. Além disso, é mais difícil para o atacante predeterminar qual ataque será bem sucedido contra o IDS, ou seja, qual ataque que com certeza o IDS não gerará alerta (fragmentação ou novos ataques).

Entretanto existem alguns problemas em potencial neste tipo de detecção. A fase de treinamento é muito importante e precisa ser feita com cuidado. Se, durante o treinamento, todos os dias houver um ataque as 4:00 da manhã com o usuário se autenticando, o IDS tomará isso como uma atividade normal e não gerará nenhum tipo de alerta na hora da detecção, provocando um falso negativo. Outro grande problema é grande quantidade de falso positivos que esse mecanismo gera. O treinamento, por melhor que seja, não contemplará eventos normais que acontecem aleatoriamente de tempos em tempos e que quando ocorrem geram alertas de falso positivos.

3.4.5.5 Posição do IDS

Um HIDS logicamente reside dentro do computador a ser protegido. Um NIDS possibilita alguns diferentes posicionamentos dentro de uma rede. O objetivo é “escutar” todo o tráfego passante numa rede. Não adianta posicionar um NIDS em um local que ele tenha acesso a apenas um segmento na rede, sendo que o ataque pode estar acontecendo em outro segmento.

Normalmente, como mostra a figura 3.1, o NIDS é conectado a uma porta do *switch* principal da rede a ser monitorada e essa porta recebe uma configuração especial chamada de *port spannig* que direciona todo tráfego que circular pelo *switch* para essa porta.

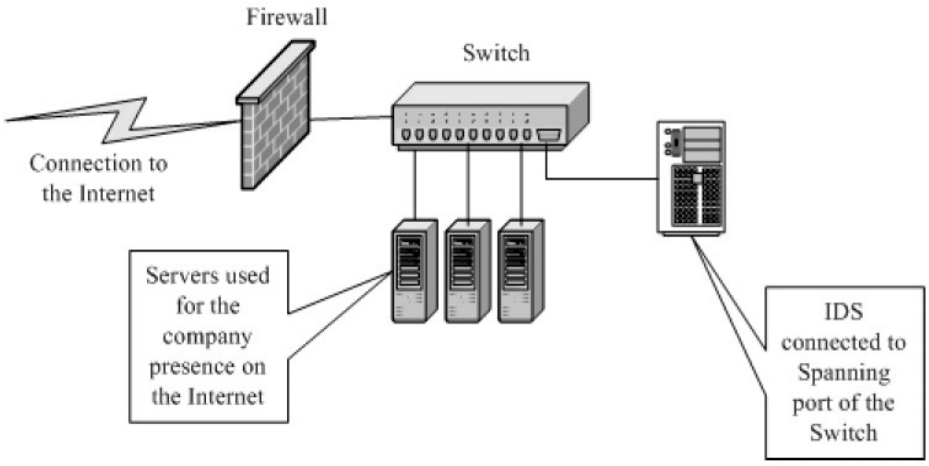


Figura 3.1 – NIDS no *switch*

Se o *switch* não possuir a configuração do *port spanning*, ou se ele já estiver sendo utilizado em outra tarefa, ou por qualquer outro motivo ele não for utilizado, o IDS poderá ser posicionado logo após o *firewall*, utilizando um *HUB* para espelhar o tráfego, como mostra a figura 3.2. Neste caso, o tráfego que o NIDS enxerga é apenas o tráfego entrante e sainte da rede. Ele não consegue enxergar o tráfego entre as máquinas da própria rede, como no caso anterior.

Para redes muito grandes e segmentadas, existe a possibilidade de existirem vários sensores inspecionando o tráfego em cada *switch* principal das subredes e trazendo os alertas para um elemento central estrategicamente localizado (de preferência numa rede exclusiva), o servidor NIDS.

4. REDES NEURAIS ARTIFICIAIS

Uma rede neural artificial consiste em uma coleção de elementos processadores que estão extremamente conectados. Ela transforma uma série de entradas em saídas desejadas. O resultado da transformação é determinado pela característica dos elementos e pesos associados com as interconexões ao longo deles[33].

A visão acima é a visão prática e recente do que vem a ser uma RNA, ou rede neural artificial. Na verdade tudo começou décadas antes, em 1943, quando Warren McCulloch e Walter Pitts apresentaram um trabalho intitulado “Neurônio Booleano”. O neurônio de McCulloch nada mais era do que uma tradução matemática do neurônio biológico, existente nos seres humanos.

4.1 A BIOLOGIA

O exato funcionamento do cérebro humano permanece um mistério. Porém, alguns aspectos deste impressionante processador são conhecidos. Em particular, o elemento mais básico do cérebro humano é um tipo específico de célula que, ao contrário do resto do corpo não parece regenerar. Este tipo de célula é a única parte do corpo que não é substituída lentamente. Por causa disso, supõe-se que essas células nos fornecem as habilidades de recordar, pensar e nos aplicam experiências anteriores em todas as nossas ações. Essas células, por volta de 100 bilhões delas, são conhecidas como neurônios [21].

Com o avanço da medicina e das pesquisas nessa área, alguns aspectos comuns da morfologia dos neurônios foram sendo conhecidos. O neurônio é o elemento básico do sistema nervoso central. Esta célula recebe, processa informação e comunica com as várias partes do corpo humano. Uma visão esquemática de um neurônio é mostrada na figura. 4.1.

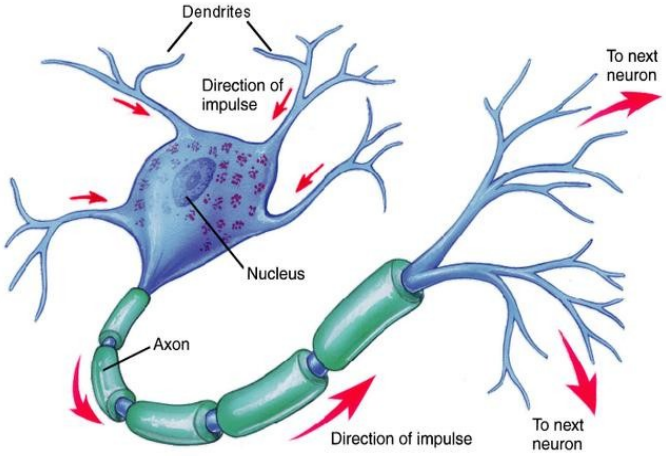


Figura 4.1 – Neurônio humano

O neurônio é composto basicamente de 3 elementos: o núcleo, os dendritos e o axônio. O núcleo, ou *soma* tem aproximadamente 30 micrômetros de diâmetro e é responsável pelo processamento dos sinais, ou das informações. Cada célula recebe por volta de 10.000 entradas através dos dendritos. Há um processo de integração (soma) no núcleo e uma saída é gerada (impulso elétrico), sendo transmitida através do axônio para o próximo neurônio. A junção de um axônio com um dendrito de outro neurônio é chamada *sinapse*. O axônio pode ter de 50 micrômetros até alguns metros de tamanho (dependendo do tipo de neurônio), contendo 10.000 conexões sinápticas em média.

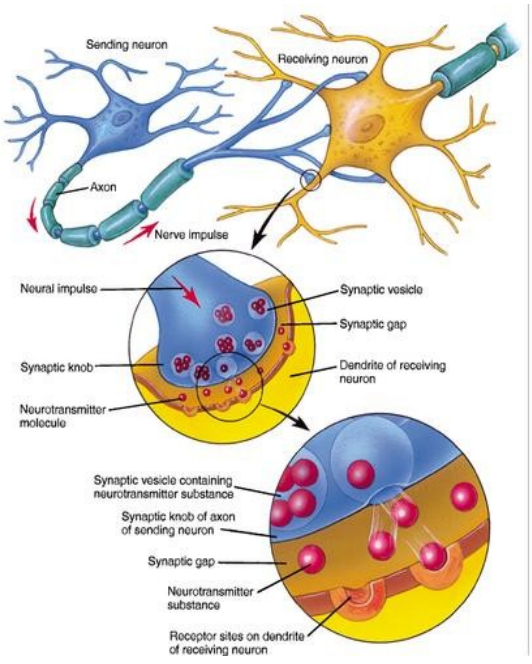


Figura 4.2 – Sinapse

As sinapses(figura. 4.2) são regiões eletroquimicamente ativas, compreendidas entre duas membranas celulares: a membrana *pré-sinaptica*, por onde chega um estímulo proveniente de outra célula, e a membrana *pós-sinaptica*, que é a do dendrito. Nesta região intersináptica, o estímulo nervoso que chega à sinapse é transferido à membrana dendrital através de substâncias conhecidas como *neurotransmissores*. O resultado dessa transferência é a alteração no potencial elétrico da membrana pós-sinaptica[18].

A conexão sináptica pode ser classificada em excitatória ou inibitória, dependendo do tipo de neurotransmissor e da natureza da membrana dendrital. “Apenas um tipo de neurotransmissor é liberado em uma dada ativação e o efeito da sinapse é sempre toda excitatória ou toda inibitória”[22].

Um neurônio recebe uma série de entradas tanto excitatórias como inibitórias e gera uma série de impulsos elétricos com uma freqüência que depende da integração destes sinais de entrada. O impulso nervoso resultante, ou potencial de ação é uma onda de despolarização da membrana axonal e se propaga ao longo da membrana se, e somente se a despolarização for suficientemente acentuada para cruzar o valor conhecido como *limiar de disparo*.

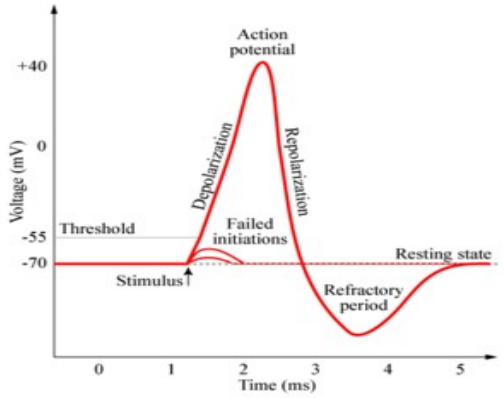


Figura 4.3 – Potencial de ação

A figura 4.3 representa a tensão elétrica observada em um ponto fixo do axônio no momento da passagem do impulso nervoso. Durante um período chamado de *período de refração absoluta*, que no exemplo vai até aproximadamente 3ms, a membrana é incapaz de produzir outro potencial de ação, independente do potencial acumulado pelo soma. Seguindo esse período de refração absoluta, persiste um *período de refração relativa* (até aproximadamente 5ms), onde dependendo do potencial acumulado é possível que o neurônio dispare. Logo após, o neurônio retorna ao seu estado de repouso.

Diante dessas observações, chegou-se numa modelagem matemática que caracteriza o

neurônio biológico, com as seguintes características:

a) Entrada: São os dendritos do neurônio biológico;

b) Pesos: É a área de armazenamento de experiência ou conhecimento do neurônio, ou seja, as sinapses;

c) Soma: É o núcleo do neurônio. Composto da somatória de cada entrada multiplicada por seu peso, aplicando ao resultado uma função de transferência que corresponde ao potencial de ativação. Além disso, um deslocamento no tempo da função de transferência pode ser utilizado aplicando-se o *bias;*

d) Função de transferência: Potencial de ativação da membrana axonal;

e) Saída: É o axônio do neurônio biológico.

A figura 4.4 ilustra o modelo do neurônio artificial.

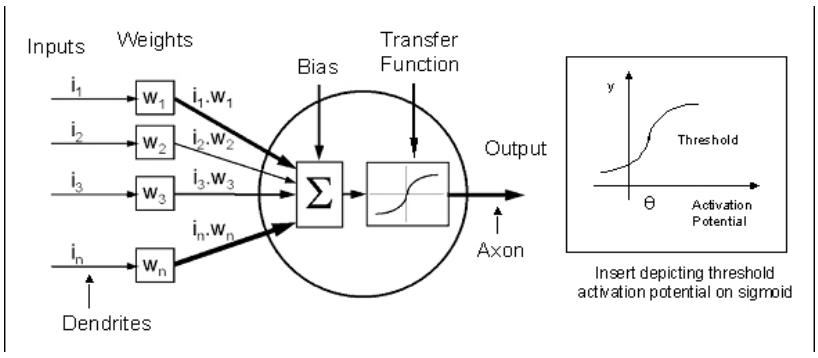


Figura 4.4 – Neurônio Artificial

4.2 O NEURÔNIO ARTIFICIAL, O PERCEPTRON E O ADALINE

4.2.1 O NEURÔNIO BOOLEANO

Em 1943 McCulloch e Pitts em um artigo intitulado "*A Logical Calculus of the Ideas Immanent in Nervous Activity*" propuseram o primeiro modelo de neurônio artificial. "Era um dispositivo binário: a sua saída poderia ser pulso ou não pulso, e as suas várias entradas tinham ganho arbitrário e poderiam ser excitatórias ou inibitórias. Para determinar a saída do neurônio, calculava-se a soma ponderada das entradas com os respectivos ganhos como

fatores de ponderação, positivos nos casos excitatórios e negativos nos casos inibitórios. Se este resultado fosse maior ou igual a um certo limiar então a saída do neurônio era pulso, e caso contrário era não pulso."[18].

A figura 4.5 exemplifica a implementação de 3 funções booleanas através de neurônios artificiais. No caso das funções *and* e *or*, as entradas binárias são u_1, u_2 e o limiar é 1,5 e 0,5 respectivamente. A função *x* realiza a operação $\sum (u \times 1 - limiar)$. Ou seja, caso a somatória das entradas esteja acima do limiar, a saída será 1. Caso contrário, a saída será 0. No caso do *not*, o limiar é 0,5 e as entradas binárias são -1 ou 0. A condição de saída permanece a mesma (acima ou abaixo do limiar) e ela será, portanto, a entrada invertida.

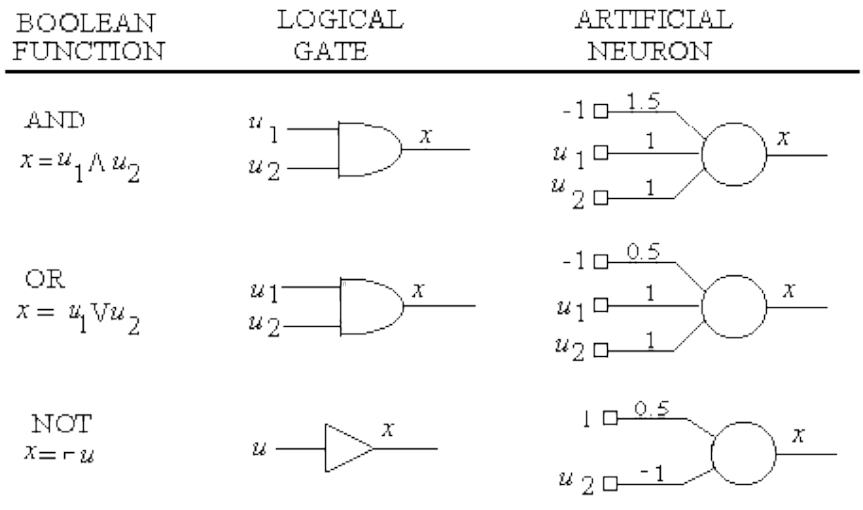


Figura 4.5 – Neurônio Booleano

O neurônio de McCulloch tem uma aplicação de particular importância, chamada de discriminador linear.

Supondo as entradas binárias e a função *y* degrau, temos genericamente um discriminador linear de *n* entradas $\{x_1, x_1, \ldots x_1\}$ e uma saída:

$$y = F\left(\sum_{i=1}^{n} w_i x_i - \Theta\right) = f(w^t x - \Theta) \; quando \; y \in [0;1] \; degrau \quad (4.1)$$

A expressão *(4.1)* divide o espaço euclidiano $\Re^n$ em duas regiões: *A* e *B:*

$$w^t x - \Theta > 0 \rightarrow x \in A \; (y = 1)$$
$$w^t x - \Theta < 0 \rightarrow x \in B \; (y = 0) \quad (4.2)$$

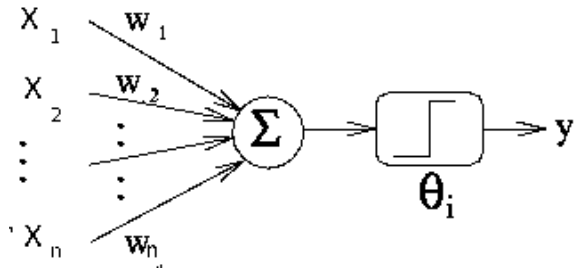


Figura 4.6 – Discriminador Linear

Em uma situação de dimensão *n*=2, temos a equação resultante $w_1 x_1 + w_2 x_2 = \Theta$, ou seja, uma reta separando o plano em duas regiões conforme figura 4.7.

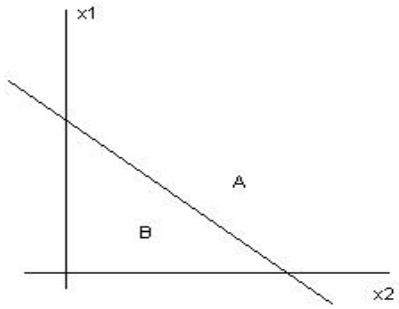


Figura 4.7 – Regiões A e B

Este comportamento do discriminador linear pode ser utilizado para tratá-lo como um classificador de padrões. Por exemplo, sejam 2 coleções de pontos: $\alpha = \{a_1, a_2, \ldots a_k\}$ de *k* vetores *n* dimensionais e $\beta = \{b_1, b_2, \ldots b_m\}$ de *m* vetores *n* dimensionais. O discriminador deve fornecer *y=0* se $x \in \alpha$ e *y=1* se $x \in \beta$. Isto é, se existir um hiperplano que separe os dois aglomerados de pontos. A figura 4.8 ilustra este exemplo em duas dimensões. A primeira coleção, em existe um hiperplano que separa os dois padrões, é chamada de linearmente separável. Já a segunda não é separável linearmente pois não há um hiperplano que consiga separar os dois padrões.

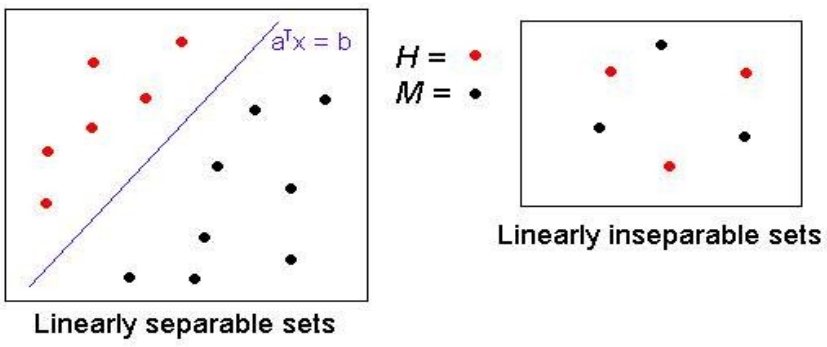


Figura 4.8 – Exemplo de coleções linearmente e não-linearmente separáveis

O neurônio de McCulloch implementa um discriminador linear simplificado, pois as entradas são binárias. A figura 4.9 ilustra três das principais funções booleanas de duas variáveis representadas no plano binário. Nota-se claramente que as funções *AND* e *OR* são linearmente separáveis, enquanto que a função *XOR* (ou exclusivo) não é. Seriam necessárias 2 retas para separar os padrões semelhantes.

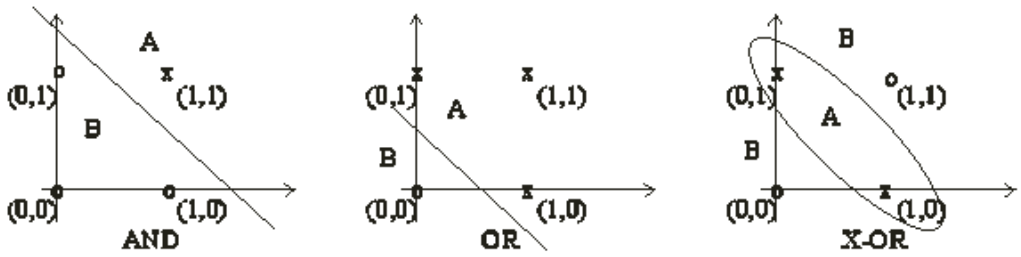


Figura 4.9 – Funções “E”, “OU” e “OU-Exclusivo”

Embora a função *XOR* não possa ser implementada utilizando um simples neurônio de McCulloch, ela poderia utilizar uma rede com três neurônios de McCulloch. E assim, sempre existirá uma rede de múltiplos neurônios capaz de implementar qualquer função booleana.

Das 16 funções booleanas de duas variáveis, apenas 2 não são linearmente separáveis: *XOR* e seu complemento. Porém, o problema se agrava à medida em que o número de variáveis vai aumentando. Em 1960, Widner apresentou um estudo sobre as funções linearmente separáveis: para o caso estudado de 2 variáveis de entrada, existem 16 funções booleanas e 14 são linearmente separáveis, ou seja, 87,5%. Já para o caso de 4 variáveis de entrada, existem 65536 funções lógicas e apenas 1772 são linearmente separáveis, isto é, 2,9%. E este número só decai.

4.2.2 PERCEPTRONS

Em 1959, Rosenblatt publicou um trabalho chamado *Perceptron*. Era uma rede de múltiplos neurônios do tipo *discriminador linear*. A topologia genérica da rede é mostrada na figura 4.10. Os neurônios são dispostos em várias camadas. A primeira camada recebe os padrões de entrada da rede e é chamada de camada de entrada. Todos os seus elementos estão

conectados aos neurônios da camada seguinte. Alguns autores referem-se a esses elementos como neurônios, porém neurônios de ligação direta, ou seja, que não realizam nenhuma operação além de conectar os padrões de entrada, ou estímulos à rede. A última camada recebe as conexões vindas dos neurônios da penúltima camada e é chamada de camada de saída. As demais camadas são as camadas ocultas, ou escondidas. Todas as camadas ocultas e de saída realizam o somatório do produto das saídas anteriores(ou valores de ativação) com os pesos de cada sinapse(ligação entre os neurônios). Aplicando o *bias*(limiar) se necessário e passando, na seqüência, por uma função de ativação gerando as saídas para a próxima camada, se houver.

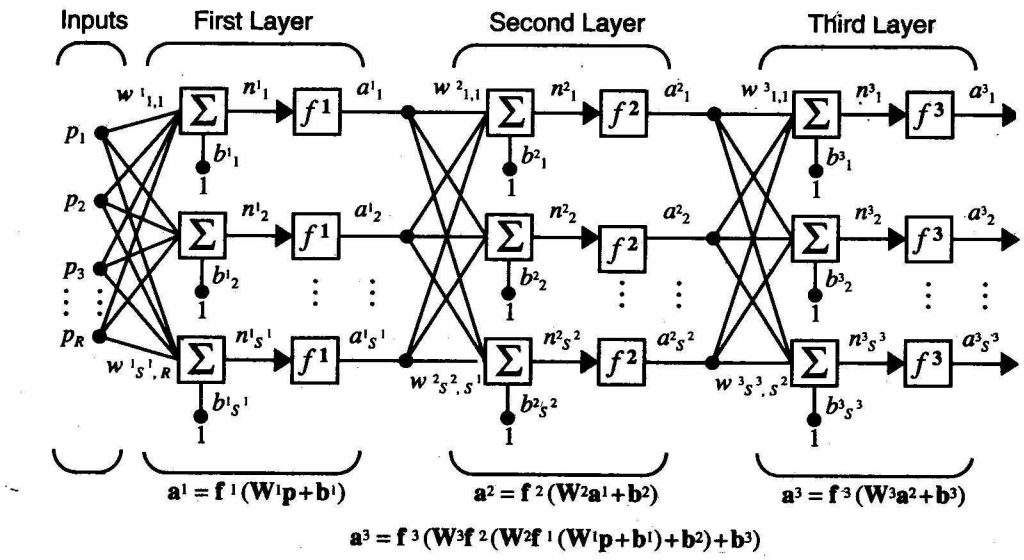


Figura 4.10 – Perceptron multi-camadas

Dada a topologia genérica da rede, o grande desafio para que a rede *perceptron* cumprisse seu objetivo, ou seja, apresentasse a saída correta para determinados padrões de entrada era escolher corretamente cada peso de cada sinapse e cada *bias* de cada neurônio. Esses parâmetros essenciais são chamados de “conhecimento” da rede. Para uma função booleana simples, como o AND, ou o OR, como já demonstrado, é relativamente simples escolher corretamente o conhecimento. Até porque é uma topologia de apenas um neurônio. Entretanto, quando se tem um problema mais complexo com várias variáveis envolvidas é praticamente impossível determinar o conhecimento sem a existência de algum método.

É justamente esse método que Rosenblatt propôs no seu trabalho. Novamente, ele foi buscar inspiração na biologia. Se o neurônio, ou a rede neuronal biológica é capaz de aprender determinadas funções, a rede artificial também seria capaz. O método de ensino consiste

basicamente em ajustar o conhecimento (pesos e *bias*) à medida em que exemplos de entrada são apresentados à rede, juntamente com a saída que se deseja obter. De tal maneira que a saída na última camada esteja o mais próximo possível da saída desejada. Rosenblatt, porém, lançou apenas a primeira semente, pois seu método funciona apenas para rede *perceptron* de uma camada (entrada e saída).

O princípio utilizado foi o princípio de aprendizado do biólogo *Hebb,* formulado em 1949 em seu livro *Organization of Behavior,* após um estudo do comportamento de animais. Segundo Hebb, o aprendizado pode ser reduzido a um processo puramente local, em função dos erros detectados localmente:

$$y = sgn(\sum_{i=1}^{n} w_i x_i - \Theta) = sgn(w^t x - \Theta)\ quando\ y \in [-1\,;1]\ operador\ sinal \quad (4.2)$$

$$w_i^{novo} = w_i^{velho} + \Delta w_i \quad (4.3)$$

$$\Delta w_i = \eta (y^d - y).x_i^d \quad (4.4)$$

$$\Theta_i^{novo} = \Theta_i^{velho} = \eta (y^d - y) \quad (4.5)$$

A alteração do i-ésimo parâmetro depende exclusivamente do produto da i-ésima entrada pelo erro de saída $e = (y^d - y)$. Sendo y a saída com os velhos parâmetros, ou seja, a equação (*4.2*) aplicada ao peso e *bias* velhos(antes do reajuste). O parâmetro η é referido como a taxa de aprendizado, na medida em que reflete a taxa com que os ganhos são alterados em conseqüência dos erros[18].

Empregando o princípio hebbiano de treinamento, Rosenblatt chegou ao seguinte algoritmo de treinamento:

1. Inicia-se as conexões com pesos aleatórios;
2. Seleciona-se um vetor de entrada dentre os exemplos de treinamento;
3. Se a saída for diferente da saída desejada, ou seja, o perceptron deu a resposta incorreta, modifica-se todos os pesos de acordo com a lei de Hebb (equação 4.3);
4. Volta-se ao passo 2.

O *bias* é considerado como a primeira conexão de cada neurônio w_0 . O valor é calculado como se fosse um peso vezes o valor de entrada sempre igual a 1. Quando o valor de saída estiver igual ao valor desejado, nenhum conhecimento é alterado.

Para a lei de aprendizado do perceptron existe um teorema de convergência que diz o seguinte: Se existe uma coleção de conexões $w^{ótimo}$ que é capaz de realizar a transformação $y = y^d$, a regra de aprendizado do perceptron vai convergir para alguma solução (que

poderá ou não ser $w^{ótimo}$) em um número finito de passos para qualquer escolha inicial de pesos[17].

4.2.3 ADALINE

Em 1960, na Universidade de Stanford, Widrow e Hoff desenvolveram um modelo neural linear simples e batizaram de ADALINE (ADAptative LINear Element) e mais tarde sua generalização multidimensional MADALINE (Múltipla ADALINE). O modelo em si ficou restrito ao ambiente acadêmico. A contribuição mais significativa, porém, do trabalho de Widrow foi a invenção de um princípio de treinamento extremamente poderoso para as redes ADALINE conhecido como *regra delta*, que mais tarde seria aproveitada em modelos neurais mais elaborados[18].

A regra delta ou algoritmo do mínimo erro quadrático LMS (*least mean square*) tem como principal diferença do perceptron a maneira na qual é utilizada a saída do sistema. No perceptron, o algoritmo de treinamento utiliza a saída da função de ativação (1 ou -1) como critério para o aprendizado. Na regra delta, a saída da rede é utilizada sem mapear diretamente entre os valores 1 e -1(antes da função de ativação). A figura 4.11 mostra um elemento novo na rede neural em forma de somatório. É justamente o LMS, ou erro médio quadrático.

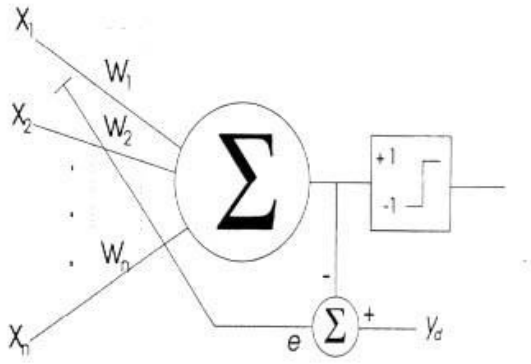


Figura 4.11 – Adaline

A equação do neurônio Adaline é basicamente a equação do perceptron (equação 4.2). O problema é determinar os pesos w_i com i = 0,1,...,n; sendo w_0 o valor do *bias*, de tal maneira que a resposta entrada-saída seja correta para um número arbitrário de sinais. Se um mapeamento exato não for possível, o erro médio será minimizado através do mínimo

quadrático. A regra delta de Widrow define um mecanismo de ajuste dos pesos de tal forma que a rede forneça os valores corretos[17].

A saída do Adaline é uma combinação linear dos componentes do vetor de entrada. Tal dispositivo pode ser considerado como um aproximador linear de funções. Supondo uma função $y(w;x)=\sum_{i=1}^{n} w_i x_i$, deseja-se obter a melhor aproximação linear possível no sentido do mínimo erro quadrático. Pretende-se determinar um $w^{ótimo}$ tal que o erro quadrático sobre o conjunto de treinamento (formado pela entrada x e saída desejada y^d) seja o menor possível. O erro total da função é definido por:

$$E=\sum_d E^d=1/2\sum_d (y^d-y)^2 \quad (4.6)$$

onde o índice *d* representa os padrões de entrada da rede e E^d o erro no padrão *d*. O algoritmo LMS procura os valores para todos os pesos que minimizem a função de erro por um método chamado de *gradiente descendente*. A idéia é modificar o peso proporcionalmente à negativa da derivada do erro do padrão corrente:

$$\Delta_d w_i=-\eta\,\partial E^d/\partial w_i \quad (4.7)$$

Então, a idéia é partindo-se do ponto arbitrário w_0 caminhar pela superfície de E^d em direção ao ponto de mínimo, bastando para isso evoluir sempre no sentido oposto do gradiente naquele ponto. A constante de proporcionalidade η determina o tamanho do *passo* que se dará naquela direção. A derivada pode ser destrinchada:

$$\partial E^d/\partial w_i=\partial E^d/\partial y\,.\,\partial y/\partial w_i \;;$$

$\partial y/\partial w_i=x_i$ por causa da linearidade e $\partial E^d/\partial y=-(y^d-y)$; então:

$$\Delta_d w_i=\eta\gamma x_i \quad (4.8)$$

sendo γ a diferença entre saída desejada e saída (y^d-y) . A figura 4.12 ilustra a representação do erro médio quadrático.

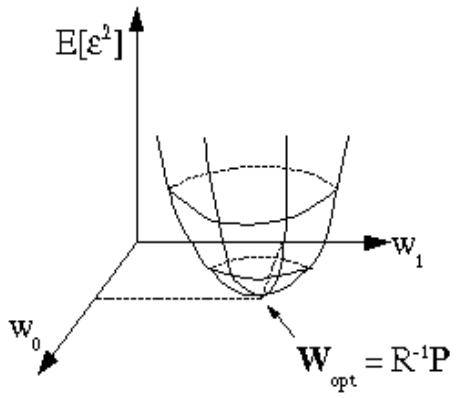


Figura 4.12 – Erro mínimo quadrático LMS

De posse de Δw pode-se atualizar o peso conforme equação 4.3. Essa atualização pode ser feita de 2 maneiras: após a apresentação de todo o conjunto de treinamento $\Delta_d w_i = \eta \sum_{i=1}^{n} \gamma x_i$, ou após a apresentação de cada exemplo $\Delta_d w_i = \eta \gamma x_i$. O primeiro procedimento é mais rápido, porém, em alguns casos o segundo procedimento pode obter uma convergência mais rápida. A eficiência do método do gradiente depende da escolha adequada de um ponto de partida w_0 e da taxa de aprendizagem η . Por exemplo, uma superfície de erro que possui várias concavidades menores que o $w^{ótimo}$, chamados de mínimos locais (figura 4.13). Se for escolhido um ponto de partida errado e uma taxa de aprendizagem baixa, a função poderá não sair do mínimo local. Por outro lado, se for escolhida uma taxa de aprendizagem muito alta, a função poderá passar do ponto de mínimo ótimo ou global, demorando a convergir.

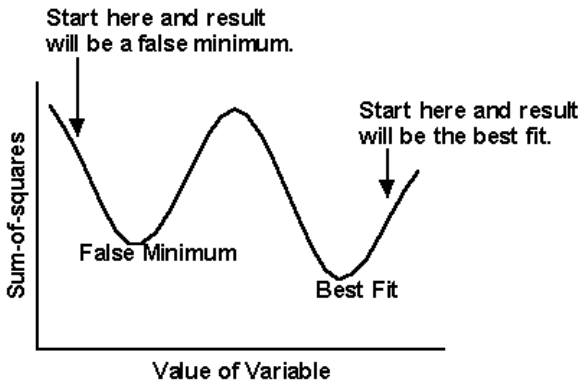


Figura 4.13 – Mínimo Local

Outro aspecto importante é definir o critério de parada do treinamento. Normalmente segue-se três linhas: calcular o erro a cada passo e quando este erro de um passo para o

seguinte decrescer menos que uma proporção; simplesmente definir um erro mínimo; limitar o número máximo de iterações.

O neurônio booleano e o perceptron tiveram grande repercussão quando foram propostos, pois imaginava-se que poderiam servir de modelo para os processos decisórios mentais. A regra de aprendizado de Hebb era bastante intuitiva e podia ser verificada na prática educacional[22].

Em 1969, Minsky & Papert publicaram um trabalho chamado *Perceptrons* onde apontaram sérias dificuldades destas redes representarem funções não linearmente separáveis. Por exemplo, um simples XOR (vide figura 4.9) não poderia ser representado por um perceptron de apenas uma camada. Eles demonstraram que era necessário adicionar uma camada oculta ao perceptron. Entretanto, não conseguiram demonstrar como ajustar os pesos da camada de entrada para a camada oculta.

Uma observação essencial a se fazer é que os algoritmos apresentados até agora conseguem ajustar os pesos apenas para redes de uma camada (entrada e saída). Os problemas não linearmente separáveis necessitam de mais de uma camada.

O trabalho de Minsky e Papert ficou marcado como o divisor da fase de euforia inicial das redes neurais e a fase de desencantamento, que percorreu toda a década de 70 e metade da década de 80. Em 1974, Werbos publicou um algoritmo chamado *Error backpropagation*, ou retropropagação do erro. O trabalho não teve repercussão no meio científico.

Em 1986, porém, o grupo de processamento paralelo distribuído do MIT, formado por Rumelhart, Hinton & Williams publicou um trabalho: "*Backpropagation* para perceptrons multicamadas", marcando o fim da fase de desencantamento e o começo do ressurgimento das redes neurais artificiais. O algoritmo de Werbos é o mais poderoso procedimento que se conhece até hoje para treinamento de redes neurais.

4.3 BACKPROPAGATION

4.3.1 O ALGORITMO

O algoritmo *backpropagation*, popularizado por Rummelhart, Hinton e Williams surgiu para solucionar o problema explorado no trabalho de Minsky e Papert. Uma vez que já se sabia que a rede perceptron funcionava como um aproximador universal, ou seja, qualquer função ou problema, seja linearmente separável ou não, poderia ser solucionada por um perceptron multicamadas. E também já se conhecia o poderoso algoritmo LMS, das redes ADALINE. O *backpropagation* juntou esses dois conceitos, generalizando a regra delta para um perceptron multicamadas, solucionando, assim, o problema de como ajustar os pesos das sinapses anteriores à saída (entrada-oculta, oculta-oculta). O método utilizado é o método da retropropagação dos erros. O ajuste dos erros das sinapses da penúltima camada estão relacionados aos erros das sinapses da última. Os da antepenúltima relacionados aos da penúltima. E assim sucessivamente, até que todos os erros sejam ajustados. Normalmente, é necessária apenas uma camada escondida e a função de ativação mais utilizada é a logística ou sigmóide(figura 4.14), utilizada por RHW(Rummelhart, Hinton e Williams).

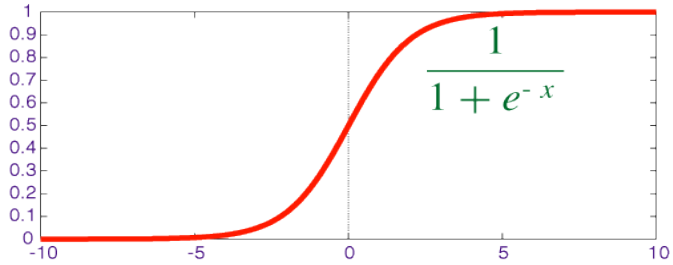


Figura 4.14 – Função de ativação sigmóide

A rede perceptron multicamadas da figura 4.10 será utilizada para o entendimento do algoritmo *backpropagation*. Os índices e notações a seguir serão expostos com o objetivo de melhor entendimento das operações matemáticas:

d - padrão desejado (entrada ou saída);

c - camada;

i - número da sinapse;

s - camada de saída;

o - camada oculta;

s_c^d - somatório: $s_c^d = \sum_i w_i x_i^d + \Theta$;

y_c^d - saída desejada;

y_c - saída da função de ativação $y_c = f(s_c^d)$

E^d - erro no padrão *d*;

η - taxa de aprendizagem;

y_c^d - representação do erro por neurônio de cada camada.

Parte-se do princípio da generalização da regra delta, então, conforme equação 4.7:

$$\Delta_d w_{ic} = -\eta \partial E^d / \partial w_{ic}$$

No caso de mais de um neurônio de saída, o erro na saída é, conforme equação 4.6:

$$E^d = 1/2 \sum_{s=1}^{N} (y_s^d - y_s)^2$$

Aplicando a regra da cadeia na derivada do erro:

$$\partial E^d / \partial w_{ic} = \partial E^d / \partial s_c^d . \partial s_c^d / \partial w_{ic}$$

sendo que $\partial s_c^d / \partial w_{ic} = x_i^d$

Definindo $y_c^d = \partial E^d / \partial s_c^d$,

a regra equivale à regra delta já demonstrada na equação 4.8:

$$\Delta_d w_i = \eta y x_i \quad .$$

O desafio é descobrir o erro y_c^d para cada camada c da rede. Aplicando a regra da cadeia em y_c^d :

$$\partial E^d / \partial s_c^d = \partial E^d / \partial y_c . \partial y_c / \partial s_c^d$$

sendo que $\partial y_c / \partial s_c^d = f'(s_c^d)$

$$\text{e} \quad \partial E^d / \partial y_c = -(y_c^d - y_c)$$

Para a camada de saída a representação do erro fica:

$$\partial E^d / \partial s_c^d = \gamma_s^d = (y_s^d - y_s) f'(s_s^d) \quad (4.9)$$

para cada neurônio de saída s.

Para a camada oculta, a medida de erro pode ser aplicada em função das entradas e pesos da soma dos neurônios da camada de saída:

$$\partial E^d / \partial y_o = \sum_{s=1}^{N} (\partial E^d / \partial s_s^d . \partial s_s^d / \partial y_o) \quad = \quad \sum_{s=1}^{N} (\partial E^d / \partial s_s^d . \partial (\sum_{i=1}^{N} w_i . x_i) / \partial y_o)$$

Como x_i da camada de saída é igual a y_o da camada oculta, ou seja entrada da camada de saída é igual a saída da camada oculta:

$$\partial E^d / \partial y_o = \sum_{s=1}^{N} (\partial E^d / \partial s_s^d . w_s) \quad \text{ou seja} \quad \partial E^d / \partial y_o = \sum_{s=1}^{N} (\gamma_s^d . w_s)$$

$$\text{ou ainda} \quad \gamma_o^d = f'(s_o^d) \sum_{s=1}^{N} (\gamma_s^d . w_s) \quad (4.10)$$

No caso da função sigmóide a derivada é

$$f'(s_c^d) = f'(1/(1+e^{-s_c^d})) = y_c(1-y_c) \quad (4.11)$$

A equação 4.10 mostra um procedimento recursivo para calcular os erros de cada unidade da rede. De posse desses erros, atualiza-se o conhecimento através da equação 4.7.

4.3.2 NA PRÁTICA

Em termos práticos, o algoritmo ocorre da seguinte maneira: em um perceptron multicamadas, quando um padrão é aplicado na entrada, seus valores de ativação (as saídas de cada neurônio depois da função de ativação) são propagados até que na última camada o valor de saída de cada unidade(ou neurônio) é comparado com o valor de saída desejado. Em cada um dos neurônios de saída há um erro relacionado. O objetivo é que o conhecimento (pesos e *bias*) da rede seja ajustado de tal maneira que no próximo conjunto de dados apresentado haja a comparação de valores de saída com desejados, o erro decaia até que, em um determinado momento, seja zero, ou esteja próximo de zero. De acordo com o algoritmo LMS de Widrow, os pesos da camada de saída podem ser ajustados pela equação 4.7 $\Delta_d w_{ic} = -\eta \partial E^d / \partial w_{ic}$. Porém, apenas aplicando esta equação, os pesos relacionados às demais camadas nunca mudariam e a rede não funcionaria como aproximadora universal de funções. A solução é distribuir o erro da camada de saída para todos os neurônios da camada oculta anterior a ela. E desta última para a oculta anterior. E quantas forem, até chegar à camada de entrada. A maneira de se fazer isso, é a atualizando o conhecimento de cada camada através do somatório dos erros de cada neurônio da camada seguinte, multiplicado pelo valor do peso entre elas e ainda pela derivada da função de ativação do neurônio, conforme mostra a equação 4.10. Isto quer dizer que o erro da última camada é calculado e é retropropagado para as camadas anteriores quantas vezes for necessário para o erro chegar no esperado.

Um parâmetro essencial no algoritmo *backpropagation* é a taxa de aprendizagem η . Ela determina a velocidade de convergência da rede ao mínimo erro quadrático. Se for escolhida uma taxa de aprendizagem muito pequena, corre-se o risco de a convergência demorar muito, ou pior, a rede ficar presa em um mínimo local, ou seja, um vale na superfície de erro muito menor do que o mínimo erro quadrático. O ideal, então, é escolher uma taxa maior possível. O problema de taxas muito grandes é a oscilação que ela provoca no momento da convergência para o mínimo erro, ou seja, os “passos” são tão grandes que a rede não consegue estabilizar no vale do mínimo erro, conforme figura 4.15. Uma maneira encontrada de amenizar essa característica foi adicionar um novo parâmetro no ajuste de pesos da rede. O parâmetro é chamado *momentum*, ou momento α e é utilizado para adicionar o impacto do ajuste anterior de pesos da rede ao novo ajuste, ajudando a diminuir a oscilação de

convergência(realiza um filtro *passa baixas*). A equação de ajuste dos pesos com o momento fica:

$$\Delta w_{i+1} = \eta \gamma x_{i+1} + \alpha \Delta w_i \quad (4.12)$$

O momento costuma variar entre 0,5 e 0,9 impactando o ajuste do novo peso a partir do ajuste feito no peso anterior. Dessa maneira a taxa de aprendizagem escolhida pode ser maior para uma convergência mais acelerada e uma rede mais estável.

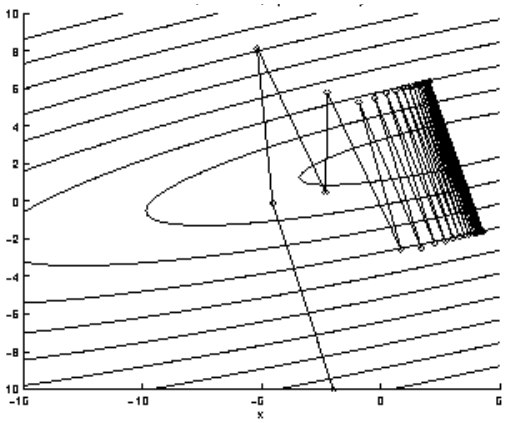


Figura 4.15 – Convergência da rede; oscila na superfície de erro com taxa de aprendizagem muito alta

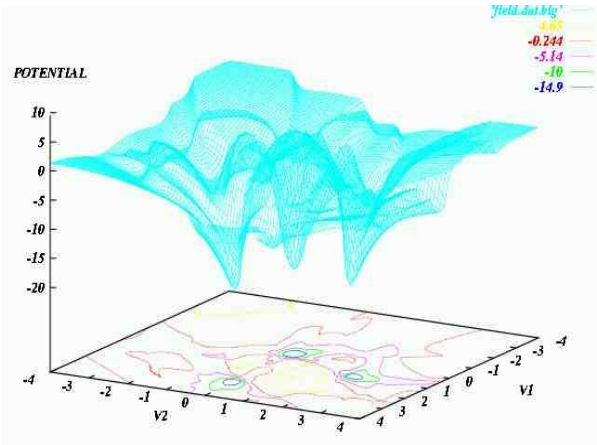


Figura 4.16 – Superfície de erro: mínimo global e mínimos locais

Em uma rede complexa, a superfície de erro é formada por vários topos e vales como mostra figura 4.16. Os vales menores são chamados de mínimos locais e o maior é o mínimo global. O método do gradiente descendente abre a possibilidade do erro “encalhar” em um mínimo local(gradiente zero) dependendo da escolha do ponto inicial e da taxa de aprendizagem. Métodos probabilísticos podem ser utilizados para escapar destes mínimos,

mas eles tendem a ser muito lentos. Outra sugestão é aumentar o número de neurônios da camada escondida, aumentando a dimensão da superfície de erro[17].

Durante o treinamento, pode acontecer de os pesos serem ajustados para valores muito altos, causando a chamada paralisia da rede. A função sigmóide(como pode ser observado na figura 4.14), a partir de certos valores muito altos positivos ou negativos dos pesos, tende a saturar nos valores um e zero respectivamente. Uma vez alcançados esses valores, a derivada $f'(s_c^d) = y_c(1 - y_c)$ tende a zero, consequentemente o erro tende a zero e os valores dos pesos não são ajustados. A rede pára.

Uma decisão muito importante na hora de projetar uma rede neural, é escolher o número de camadas e o número de elementos, ou neurônios de cada camada. Não existe uma resposta pronta ou calculável. Depende muito do tipo de aplicação e da experiência do usuário. O que existe são considerações de autores mais experientes que trabalharam na questão e podem auxiliar na hora de se projetar uma rede:

a) Geralmente três camadas é o suficiente(entrada, oculta e saída). Algumas vezes, porém, o problema poderá ser mais facilmente resolvido adicionando uma camada oculta a mais. Quer dizer, a rede poderá aprender mais rapidamente.
b) O tamanho da camada(número de neurônios) de entrada depende essencialmente da natureza da aplicação. O tamanho da camada de saída depende do tipo de resposta esperado da rede, se são requeridos valores booleanos, binários ou analógicos(Freeman/Skapura, 1991).
c) Definir o número de elementos da camada oculta é o grande desafio. Quanto maior for a complexidade na relação entre dados de entrada e saída desejada, maior será o número de elementos processadores na camada oculta[21].
d) Se o processo a ser modelado foi separado em múltiplos estágios, então serão requeridas camadas ocultas adicionais, caso contrário, essas camadas ocultas apenas servirão para ativar a memorização da rede, não constituindo uma solução de generalização[21].
e) O número de padrões de treinamento utilizados determinam um limite superior para o número de neurônios na(s) camada(s) oculta(s). Para calcular este limite, divide-se o número de padrões de treinamento(pares entrada, saída desejada) pelo número total de elementos processadores de entrada e saída. Divide-se novamente o resultado por um fator escalável entre cinco e dez. Fatores mais altos podem ser utilizados para dados muito ruidosos, enquanto que para dados mais limpos, o fator gira em torno de dois. É importante que as

camadas ocultas tenham poucos elementos processadores. Muitos neurônios nessas camadas podem fazer com que a rede memorize os dados de treinamento. Se isso acontecer, não ocorrerá nenhum tipo de generalização e rede não terá utilidade para reconhecer novos padrões[21].

f) Por último, pode ser interessante, após as primeiras tentativas de treinamento, remover neurônios escondidos que sejam supérfluos. Examinando os valores dos pesos na(s) camada(s) oculta(s) periodicamente, enquanto a rede treina, poderão haver pesos que mudam muito pouco dos valores iniciais. Esses neurônios podem não estar participando do processo de aprendizagem e menos neurônios na camada oculta serão suficientes.

A rede está corretamente projetada. Outro desafio se segue. Quantos pares de treinamento deverão ser utilizados? Quais escolher? Como utilizá-los? Novamente, existe uma série de fatores a serem considerados:

a) Uma normalização dos dados de entrada e saída é necessária para o treinamento de uma rede. Esses valores devem estar na mesma magnitude. Se as variáveis de entrada e saída não estiverem na mesma ordem de magnitude, algumas variáveis podem parecer mais significantes do que realmente são. O algoritmo de treinamento deve compensar a diferença de magnitude ajustando corretamente os pesos. Isso não é efetivo no algoritmo *backpropagation*. Por exemplo, uma variável de entrada de 50000 e outra de 5. O valor do peso para a segunda variável, ao entrar no neurônio 1 da camada oculta tem que ser muito maior do que o valor do peso para a primeira. Com isso a segunda passa a ter alguma significância. Entretanto, uma função de ativação sigmóide, ou outra parecida(como tangente hiperbólica), no neurônio 1 da camada oculta não consegue distinguir entre dois valores muito grandes, por causa da saturação já discutida(ambos terão saídas idênticas iguais a 1) [24].

b) Existe uma relação que pode ser traçada entre o número de padrões a serem treinados e o número de pesos da rede. Normalmente há uma grande quantidade de dados disponíveis para treinamento, mas utilizando apenas uma parte deles, a rede será capaz de generalizar o restante. O cálculo é o seguinte: $W / P = e ; P = W / e$. Quer dizer, para um erro de 0,1, uma rede com 80 pesos vai precisar de 800 padrões para treinar corretamente, acertando 90% dos padrões de generalização, ou 95% dos padrões de verificação[23]. A diferença entre padrões de generalização e verificação será visto a seguir.

c) Como já foi mencionado, para o treinamento de redes neurais, não é necessário utilizar todos os padrões disponíveis. Muitos deles são importantes para fazer as verificações e

generalizações. Para determinar a performance da rede, depois de treinada, utiliza-se dois critérios: (1) apresenta-se padrões à rede os quais já foram utilizados na fase de treinamento, verificando como a rede se comporta, computando o percentual de acertos. São os chamados padrões de verificação; (2) apresenta-se à rede padrões novos, que não foram utilizados na fase de treinamento e verifica-se o percentual de acerto e conseqüente capacidade de generalização da rede(o que ela aprendeu). São os chamados padrões de generalização. Importante frisar que esses passos são realizados após o treinamento e, portanto, os pesos não mudam durante os testes[24].

A inicialização dos pesos e *bias* da rede é feita de maneira aleatória. Porém, é importante que sejam escolhidos pesos numa faixa compreendida entre -0,5 e 0,5. Se os valores iniciais de pesos forem muito grandes, as entradas para os neurônios das camadas ocultas e de saída poderá haver a paralisia(região de saturação). Se os valores forem muito pequenos, o aprendizado poderá ser muito lento.

A última consideração a ser feita sobre o algoritmo *backpropagation* é em relação aos valores de saída da rede. Em geral a faixa de variação da saída desejada é uma, enquanto que a função de ativação cobre outra faixa. Deve-se reescalar esta variável de forma que ela se encontre numa região eficiente de operação do ponto de vista do algoritmo de treinamento[18].

4.4 AS REDES RECORRENTES E A APRENDIZAGEM COMPETITIVA

Por questão de foco do trabalho, os outros tipos de redes neurais existentes (diferentes do perceptron multicamadas com algoritmo backpropagation) não serão explorados tão a fundo neste capítulo. Entretanto, é interessante situar o leitor no que existe de alternativa às redes MLP (multi layer perceptron), até porque outros trabalhos sobre o mesmo assunto desta dissertação utilizam outros tipos de rede. A literatura disponível para os assuntos a seguir é muito vasta.

4.4.1 REDES RECORRENTES

4.4.1.1 Redes de Elman e Jordan

As redes recorrentes são redes neurais parecidas com as vistas anteriormente. A diferença é a realimentação, ou seja, a ativação ou saída dos neurônios da última camada podem ser utilizados como entrada na camada escondida, ou na camada de entrada. A saída da camada oculta pode ser utilizada como entrada dela mesma e outras variações conforme o tipo de rede e a aplicação.

Em 1986, Jordan apresentou a sua rede onde os valores de ativação da saída voltavam à rede como neurônios extras de entrada(figura 4.17). O peso dessas novas entradas era o valor fixo +1 e não participavam do processo de treinamento que permanecia o mesmo das redes MLP. Em 1990, Elman apresentou a sua rede onde os valores de ativação dos neurônios da camada oculta eram reutilizadas como entradas da mesma(figura 4.17) e foram chamados de unidades de contexto. Seus valores de ativação são sempre os valores da camada oculta no passo anterior de aprendizagem (t-1)[17].

As redes de Jordan e Elman podem ser utilizadas para reproduzir seqüências de tempo, por exemplo para controlar um objeto se movendo em 1D. A idéia das redes recorrentes é a de lembrar estados anteriores dos valores de entrada[17].

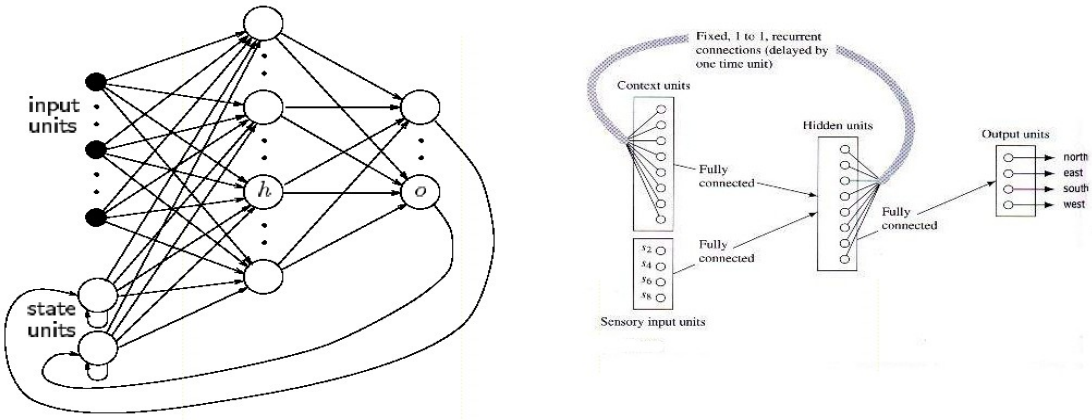


Figura 4.17 – Redes recorrentes de Jordan e Elman

4.4.1.2 Rede de Hopfield

As primeiras redes recorrentes surgiram em 1977 com Anderson e Kohonen. Em 1982, Hopfield apresentou um modelo matemático completo para redes recorrentes, juntando as idéias dos seus antecessores. “A rede de Hopfield possui uma única camada de neurônios realimentados e implementa assim uma memória auto-associativa, isto é, ao ser apresentado

um padrão de *n* bits a rede retorna um padrão armazenado de *n* bits que lhe é mais próximo(associado).”[22]. A figura 4.18 mostra uma rede de Hopfield. A saída de cada neurônio é a entrada para outros neurônios com excessão dele mesmo.

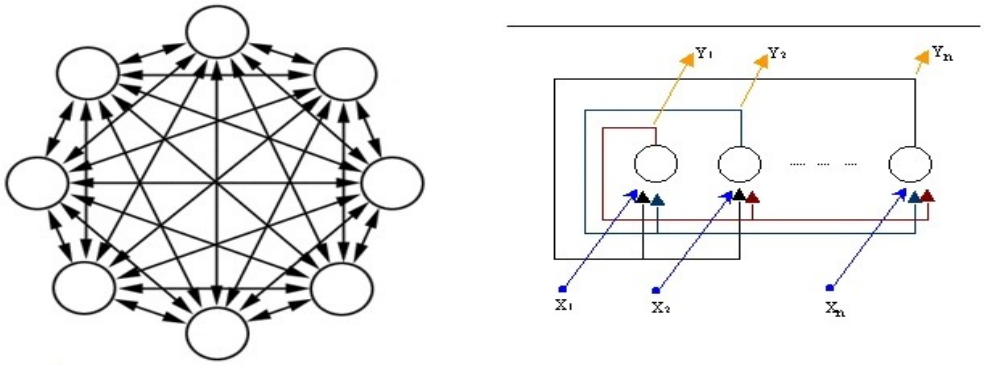


Figura 4.18 – Rede de Hopfield

Desta maneira a ativação de cada neurônio pode ser calculado como

$$a_i = \sum_{i=1}^{N} w_{ij}\, s_j + e_i - B_i \quad (4.13)$$

onde *a* é ativação, *w* é o peso, *s* é a saída, *e* é a entrada, *B* é o *bias* e *i* é o número do neurônio, variando de 1 a *n*.

Em forma de vetor, a fórmula fica

$$A = WS + E - B \quad .$$

A matriz W é a matriz de pesos, simétrica e com diagonal igual a 0, escolhida inicialmente através do treinamento.

Os padrões a serem armazenados na memória associativa são escolhidos *à priori* e treinados de acordo com a equação

$$w_{ij} = \sum_{p=1}^{m} (2a_i^p - 1)(2a_j^p - 1) \quad (4.14)$$

Sendo que cada padrão p é um vetor $a_1^p a_2^p a_3^p \ldots a_n^p$ com a_i^p = 0 ou 1

existindo *m* padrões distintos.

A função $(2a_j^p - 1)$ converte 0 e 1 em -1 e +1.

O peso w_{ij} é incrementado em 1 se $a_i^p = a_j^p$ e decrementado de 1 caso contrário.

Este procedimento é repetido para todos *i, j* e para todos os *m* padrões.

Com o treinamento realizado e a matriz W definida, o estado da rede será definido pelo vetor de saída S. Individualmente, as saídas são calculadas para cada neurônio através da função de ativação. Se o valor da ativação(equação 4.13) for menor que 0, a saída será -1, se for maior que 0, a saída será +1. Desta maneira, por exemplo, uma saída do tipo [1 -1 -1 -1 1 ... 1] define o estado de uma rede de Hopfield. Como apenas 2 valores são possíveis para cada neurônio, 2^n estados são possíveis.

Se a diagonal da matriz de pesos é nula e ela é uma matriz simétrica, então a rede recorrente é estável, provaram Cohen e Grossberg em 1983. Uma vez que a rede está estável, são aplicadas as entradas, com os padrões a serem reconhecidos, causando um estado de transição da rede. Após a entrada, os novos valores de ativação podem ser calculados de 3 maneiras:

a) assíncrona: cada neurônio calculará individualmente sua saída aleatoriamente, um de cada vez. O novo valor de saída de um neurônio acarretará na mudança do valor de entrada dos demais;

b) Seqüencial: semelhante ao anterior, porém não mais de maneira aleatória e sim os neurônios em seqüência;

c) Síncrona: todos os valores de saída dos neurônios calculados simultaneamente, utilizando os valores velhos de peso.

Esta transição continua, até a rede conseguir equilibrar-se, retornando, na saída, o padrão associado. A rede se equilibra quando a condição $sgn[A] = O$ é satisfeita[24]. Quando isso ocorre, o próximo estado da rede é igual ao estado presente e a transição termina.

Em termos da função de energia da rede, “os padrões armazenados na Rede de Hopfield são mínimos locais da função de energia(estados de equilíbrio). A partir de um padrão apresentado(valor inicial) a rede estabiliza no mínimo de energia de sua respectiva “bacia de atração”. Isto é, não retorna necessariamente o padrão geometricamente mais

próximo. Além disso, pode acontecer de que haja mínimos locais não desejados, levando a rede a retornar padrões espúrios."[22]. A figura 4.19 ilustra melhor a situação.

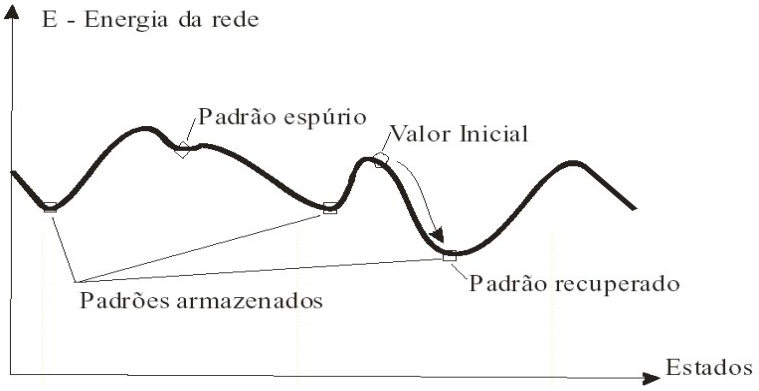


Figura 4.19 – Padrões e função de energia da Rede de Hopfield

A rede de Hopfield é muito utilizada para reconhecimento de caracteres, porém, apenas no âmbito acadêmico, justamente por causa de suas limitações: ocorrência de padrões espúrios, número máximo de padrões limitado e o fato do padrão recuperado não ser necessariamente o padrão mais próximo.

4.4.2 APRENDIZAGEM COMPETITIVA

Até o momento, a discussão sobre redes neurais artificiais conduzida considerou que havia sempre um par de exemplos para treinamento: a entrada e a saída desejada. Todavia, existem problemas nos quais a saída desejada não está disponível. Apenas a entrada é provida e a rede neural tem que ser capaz de gerar uma saída adequada ao problema. Alguns exemplos destes problemas são[17]:

a) *clustering*: a entrada precisa ser agrupada em conjuntos ou *clusters* e a rede neural tem que achar esses conjuntos para os dados de entrada. A saída do sistema é o nome do conjunto a que pertence aquela entrada.
b) *vector quantization*: um espaço contínuo precisa ser discretizado. A entrada do sistema é um vetor *n*-dimensional *x* e a saída é uma representação discreta do mesmo.
c) redução de dimensionalidade: a entrada é agrupada em um sub-espaço que tem dimensão

menor do que a dimensão da entrada.

Este tipo de solução em redes neurais é chamada de aprendizagem não supervisionada, ou seja, não é necessário um “professor” externo para ensinar a rede (saídas desejadas).

4.4.2.1 Neurônio vencedor

A aprendizagem competitiva é um tipo de rede com aprendizagem não supervisionada. A figura 4.20 ilustra um exemplo de uma rede simples de aprendizagem competitiva. Todos os neurônios da camada de entrada estão conectados aos neurônios da camada de saída. Desta maneira, quando um padrão qualquer é apresentado na entrada, apenas um dos neurônios de saída é ativado. Ele é chamado de neurônio “vencedor”. Em uma rede corretamente treinada, todos os padrões que pertencerem a um mesmo conjunto, terão o mesmo neurônio vencedor.

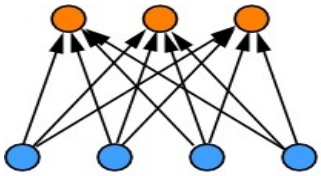


Figura 4.20 – Aprendizagem Competitiva

Para determinar o neurônio vencedor, existem dois métodos principais: produto escalar e distância euclidiana.

O método do produto escalar requer que os vetores de entrada *x* e de pesos *w* estejam normalizados em uma mesma unidade de tamanho. A saída de cada neurônio é o produto escalar do vetor de entrada pelo vetor de peso do neurônio de saída. No fim, o neurônio que tiver a maior saída é o vencedor(ativação 1), o restante permanece com ativação zero. O ajuste dos pesos é feito apenas nos pesos do neurônio vencedor e faz com que o vetor de peso seja rotacionado para a entrada. Toda vez que aquela entrada, ou uma próxima for apresentada à rede, este neurônio, por ter os pesos mais próximos à entrada será selecionado e novamente rotacionado para aquela entrada. Este método porém falham quando os vetores não estão normalizados. Se um dos vetores de peso, por exemplo, for bem maior do que o outro e o vetor de entrada for do tamanho do segundo, mesmo que a entrada esteja mais próxima do primeiro, o produto escalar fará com que o segundo seja o vencedor.

O método da distância euclidiana consiste basicamente no mesmo método do produto

escalar, porém serve para dados não normalizados. O vencedor agora será escolhido pela menor distância euclidiana entre os vetores(entrada e peso de cada neurônio).

Após o treinamento, por exemplo em uma rede com 4 neurônios de saída, os vetores desses neurônios estarão agrupados de acordo com cada conjunto de entrada, como mostra a figura 4.21

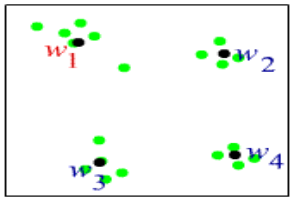


Figura 4.21– Conjuntos de dados entrada classificados por uma rede de aprendizagem competitiva; “clustering”

Uma quantização de vetores divide o espaço de entrada em um determinado número de sub-espaços, representando cada entrada com o nome do sub-espaço correspondente. A grande diferença para a técnica de *cluster* é que o interesse passa a ser todo o espaço de entrada, não só os conjuntos de dados similares. A figura 4.22 ilustra a divisão do espaço de entrada por quantização de vetores. As regiões onde a densidade de entrada é maior(parte de baixo) são representadas por mais neurônios(5) e as regiões onde não há tantos valores de entrada(parte de cima) são representadas com menos neurônios(2). As aplicações para a quantização de vetores podem estar na área de telecomunicações ou *storage*, onde os dados precisam ser comprimidos. Também há métodos que combinam quantização de vetores com aprendizagem supervisionada para aproximação de funções ou reconhecimento de padrões.

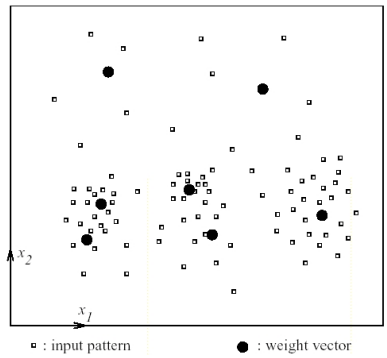


Figura 4.22– Quantização de vetores

4.4.2.2 Rede LVQ

A rede LVQ, ou *learning vector quantization*, foi apresentada por Tuevo Kohonen no meio dos anos 80. É um exemplo de uma aplicação híbrida de aprendizagem supervisionada e aprendizagem não supervisionada. A rede é formada por uma camada de entrada, uma camada intermediária(camada Kohonen) e uma de saída (figura 4.23). As camadas de saída representam o número de diferentes classes da rede. Essas classes são agrupadas pela camada Kohonen. O número de neurônios de cada camada depende da complexidade da relação entrada-saída.

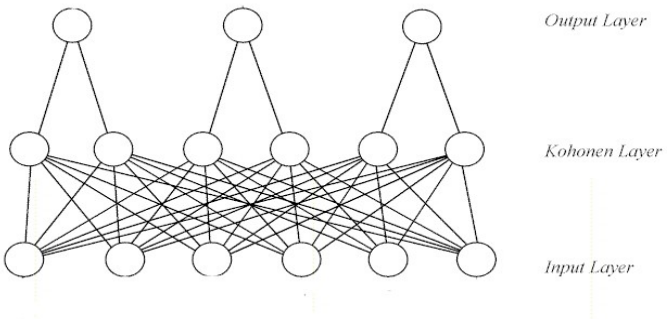


Figura 4.23– LVQ – learning vector quantization

Este tipo de rede define limites de decisão no espaço de entrada. Os dados de treinamento são como decisões exemplares e a partir deles a rede definirá as classes. O algoritmo é basicamente o seguinte[17]:

a) A cada neurônio de saída, uma classe *c* é associada;

b) Um exemplo de treinamento consiste em um vetor de entrada e sua respectiva classe associada;

c) Utilizando alguma medida de distância entre os vetores de peso e o vetor de entrada, os 2 melhores neurônios são selecionados;

d) Estes neurônios são comparados com a saída desejada(escolhe-se o vetor de peso que seja a classe a que a entrada pertence) e um deles é o vencedor;

e) O vetor de pesos do vencedor é rotacionado para o vetor de entrada e o do perdedor é rotacionado para longe do vetor de entrada.

Existem diversos tipos e novos algoritmos LVQ estão surgindo para determinar a melhor maneira de escolher as classes associadas, quantos neurônios vencedores devem ser selecionados, como ajustar os pesos e como adaptar o número de neurônios de saída.

4.4.2.3 Rede Kohonen

A rede Kohonen pode ser considerada como uma evolução das redes LVQ, embora cronologicamente ela tenha surgido antes. Utiliza aprendizagem não supervisionada e se auto-organiza de acordo com características topológicas dos padrões de entrada. É uma rede surgida como mais uma observação das características do cérebro biológico. Em muitas áreas do cérebro as atividades sensoriais são representadas na forma de mapas de duas dimensões. Por exemplo no sistema visual, há muitos mapas topográficos do espaço visual na superfície do córtex visual. Um bebê quando nasce precisa aprender a focar seus olhos e mapear o que ele enxerga. Em poucos dias, sem que ninguém lhe ensine, ele aprende a associar os estímulos visuais que recebe com os objetos e formas do mundo. Ou aprende a trajetória de movimento dos objetos.

Um sistema de auto-aprendizagem proposto por Kohonen procura explicar como redes neurais podem se auto-organizar e aprender sem que lhe seja fornecida a resposta correta para aquele padrão de entrada. A rede Kohonen não é um sistema hierárquico e sim um *array* de neurônios bastante interconectados uns aos outros. A figura 4.24 ilustra uma rede deste tipo em duas dimensões. As entradas estão conectadas aos neurônios através de pesos ajustáveis e estes estão conectados entre si através de pesos fixos. As entradas recebidas por estes neurônios são não só as entradas dos padrões da rede, mas as saídas dos outros neurônios da mesma camada.

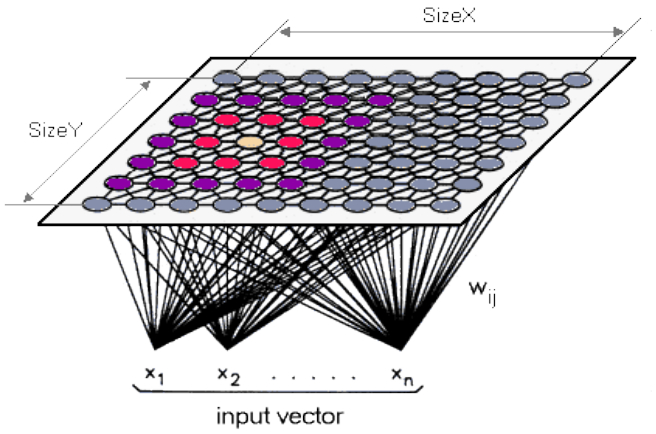


Figura 4.24– Rede Kohonen bidimensional

Durante cada ciclo de treinamento, um padrão completo é apresentado a cada neurônio. A saída é calculada através da função sigmóide da soma de peso multiplicado pela entrada. Após isso, a entrada é removida e os neurônios interagem entre si. O neurônio com a maior ativação é declarado vencedor e é escolhido para prover a saída. Entretanto, os pesos ajustados são os pesos não só do neurônio vencedor e sim todos os pesos de sua vizinhança. O tamanho da vizinhança decresce a cada ciclo.

O treinamento da rede Kohonen é realizado basicamente pelo algoritmo[24]:

a) É desejável que os vetores de entrada e de pesos estejam normalizados. Então, os pesos são inicializados de maneira aleatória;

b) Um vetor de entrada é apresentado para a rede. Todos os neurônios da rede recebem este vetor.

c) O neurônio vencedor é escolhido como o que tiver a maior similaridade de medida entre o seu vetor de pesos e o vetor de entrada. Por exemplo, a medida de menor distância euclidiana pode ser escolhida.

d) O raio da região de vizinhança do neurônio vencedor diminui. Este raio poderá ser escolhido bastante grande no início do treinamento e vai diminuindo para incluir apenas o neurônio vencedor e seus vizinhos imediatos.

e) O peso do neurônio vencedor e seus vizinhos são ajustados de acordo com a fórmula

$$(W_i)_{novo} = (W_i)_{velho} + \alpha [x - (W_i)_{velho}]$$

sendo W_i o vetor de pesos, x o vetor de entrada e α a taxa de aprendizagem. Dependendo do tamanho da vizinhança, uma função gaussiana F_i poderá ser utilizada e a fórmula reescrita para:

$$(W_i)_{novo} = (W_i)_{velho} + \alpha F_i [x - (W_i)_{velho}] \quad (4.15)$$

f) O próximo vetor de entrada é apresentado. Os passos c, d, e são repetidos até o fim de todas as entradas.

Para uma boa convergência, a taxa de aprendizagem e o tamanho da vizinhança deverá ser diminuído gradualmente a cada ciclo. No início do treinamento, como a vizinhança está bastante grande, muitos neurônios irão aprender cada padrão. À medida em que o treinamento se desenvolve, o tamanho da vizinhança vai diminuindo gradualmente e menos neurônios irão aprender cada padrão. Até o momento em que o neurônio vencedor irá ajustar apenas seu próprio peso. Então a rede estará treinada.

A figura 4.25 mostra um exemplo de treinamento da rede Kohonen. No primeiro quadro, 2000 vetores de entrada espalhados. No segundo quadro a inicialização de 30 vetores de peso. No terceiro quadro, a representação dos vetores de peso após 5000 ciclos de treinamento.

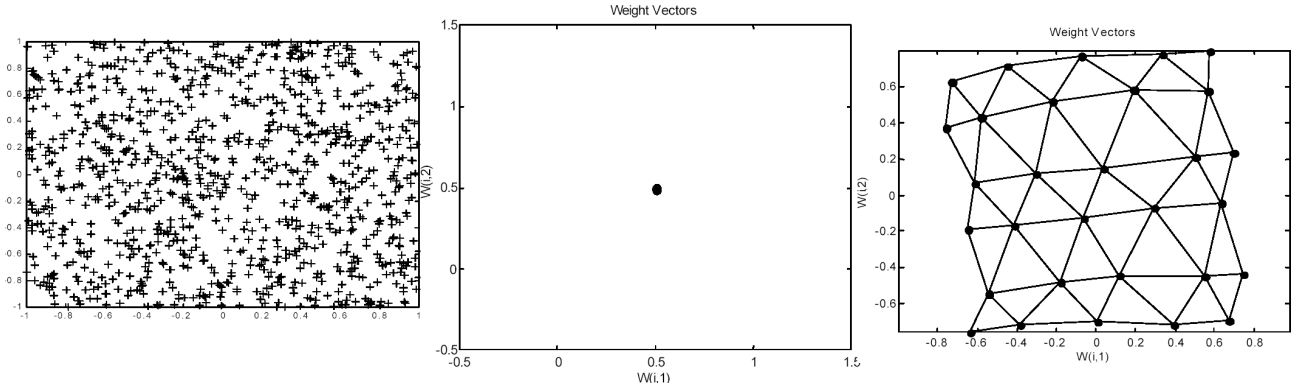


Figura 4.25– Treinamento de uma rede Kohonen bidimensional

4.5 CONSIDERAÇÕES FINAIS SOBRE REDES NEURAIS

*A*s redes neurais podem ser divididas em relação à sua micro-estrutura, meso-estrutura, ou macro-estrutura(figura 4.26).

A micro-estrutura diz respeito às características de cada neurônio. Os neurônios são formados pela soma do produto entre entrada e pesos, podendo ser deslocada pelo valor do *bias* antes de passar pela função de ativação. As funções de ativação são dos mais diversos tipos como linear, sinal, degrau, logística, tangente hiperbólica, etc.

A meso-estrutura diz respeito à topologia das redes neurais. Quantos neurônios por camada, quantas camadas e o tipo de conexão (*forward*, *backward*, lateral). Ela pode ser uma rede perceptron multicamadas, uma rede recorrente de Hopfield, uma rede híbrida, formada por sub-redes ou qualquer outro tipo de configuração que venha dar algum resultado.

A macro-estrutura diz respeito ao tamanho das redes e ao grau de conectividade. Na verdade é uma associação de redes para abordar problemas complexos.

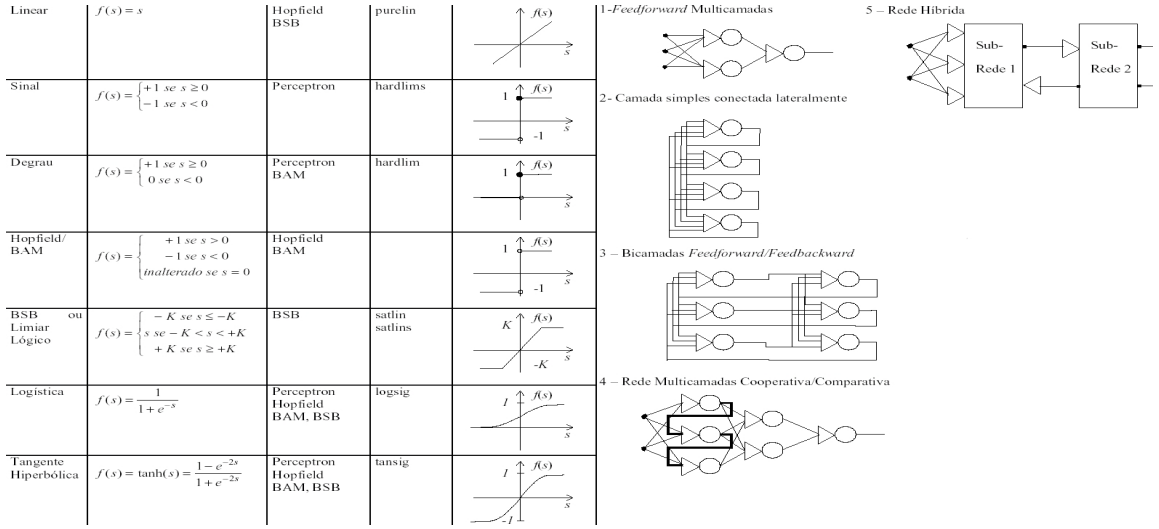


Figura 4.26– Micro e Meso estruturas de redes neurais

O treinamento de redes neurais possui dois paradigmas: treinamento supervisionado ou aprendizagem associativa e treinamento não supervisionado ou auto-organização. No primeiro, o treinamento é realizado com a presença de um "professor externo" que provê o par de exemplos para o treinamento: padrão de entrada e saída desejada. No segundo, o neurônio de saída é treinado para responder com conjuntos de padrões associados às entradas. Neste caso, não há classificações *a priori* das categorias de padrões. O sistema tem que construir sua própria representação dos padrões de entrada.

Quanto ao uso de redes neurais em aplicações reais, a figura 4.27 ilustra alguns tipos de aplicação que podem utilizar redes neurais. Em (1), a classificação ou reconhecimento de padrões. Em (2), a categorização ou divisão em conjuntos. Em (3), a aproximação de funções. Em (4), a capacidade de predição do próximo valor. Em (5), a otimização de uma rota. Em (6), a memória associativa. Em (7), o controle.

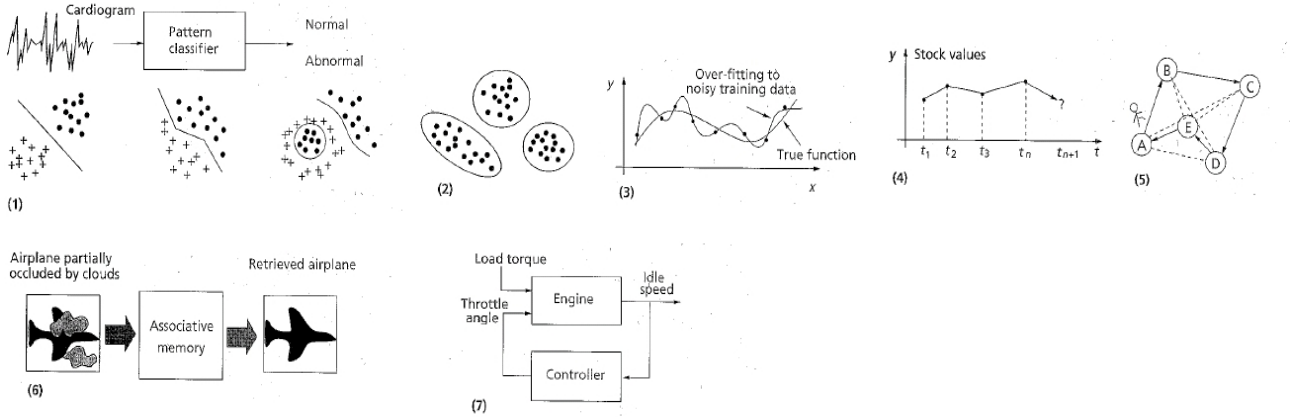


Figura 4.27– Aplicações de redes neurais

Para terminar o estudo, vale a pena ressaltar as características positivas e negativas da

utilização de redes neurais artificiais[22]:

Positivas:

a) capacidade de aprendizagem: as redes neurais não são programadas e sim treinadas com padrões podendo ser adaptadas através da entrada;

b) paralelismo: as redes neurais são massivamente paralelas e muito adequadas para simulação ou implementação de computação paralela;

c) representação distribuída do conhecimento e tolerância a falhas: o conhecimento é armazenado de forma distribuída em seus pesos, aumentando a tolerância do sistema a falhas de neurônios individuais. Entretanto o sistema deve ser treinado prevendo esta possível falha, nem toda rede é automaticamente tolerante a falhas;

d) armazenamento associativo da informação: para um certo padrão de entrada, a rede fornece o padrão que lhe é mais próximo. O acesso não é feito por endereçamento;

e) robustez contra perturbações ou dados ruidosos: quando treinadas para tal, as redes são mais preparadas para lidar com padrões incompletos(ruidosos).

Negativas:

a) aquisição de conhecimento só é possível através de aprendizado: devido à representação distribuída, é muito difícil introduzir conhecimento prévio em uma rede neural;

b) não é possível a introspecção: não é possível analisar o conhecimento ou percorrer o procedimento para a solução como é possível em sistemas especialistas;

c) difícil dedução lógica (seqüencial): é quase impossível obter-se cadeias de inferências lógicas em redes neurais;

d) aprendizado é lento: principalmente em redes completamente conectadas e quase todas as variantes do algoritmo *backpropagation*.

5 DETECÇÃO DE INTRUSÃO UTILIZANDO REDES NEURAIS

Na discussão de sistemas de detecção de intrusão, feita no capítulo 3, os IDSs foram divididos de acordo com sua abrangência de detecção. O HIDS é o IDS de *host*, residente na máquina destino e que possui uma detecção baseada em chamadas de sistema e gerenciamento de arquivos e registros. O NIDS é o IDS de rede, residente em um ponto estratégico na rede, onde consegue enxergar todo o tráfego, inspecionando pacote a pacote. Além disso, dividiu-se o mecanismo de detecção de intrusão em dois casos: o mecanismo baseado em assinatura, onde o sistema precisa de um conhecimento prévio do ataque cadastrado em sua base e o mecanismo baseado em anomalia, que procura por atividades que o sistema aprendeu como não normais.

A pesquisa na área de detecção de intrusão é antiga e os primeiros passos foram dados por James P. Anderson em 1972. De lá pra cá, muitos IDSs foram construídos. A maioria dos IDS comerciais existentes implementam uma detecção baseada em assinatura. Normalmente se preocupam com questões como esquemas de gerenciamento, flexibilidade de adaptação a uma rede específica a ser monitorada, proteção própria contra códigos maliciosos, interoperabilidade com outras ferramentas de segurança, monitoração de novas aplicações, gerência de eventos, atualização da base de assinaturas e resposta ativa a ataques (mudança em regras de *firewall*, reconfiguração de roteador).

Alguns dos principais HIDS comerciais existentes são o Cybercop da Network Associates (NAI), KaneSecurity Monitor da RSA Security e o Tripwire. Os NIDS são o RealSecure da ISS, o Cisco Secure IDS ou NetRanger da Cisco Systems, o Centrax da CyberSafe corporation e a Network Flight Recorder. Além desses, existe um NIDS que será especialmente detalhado mais tarde e utilizado, o SNORT, um *software* de código aberto disponível na Internet (http://www.snort.org).

O grande problema dos sistemas baseados em assinatura, como já foi discutido, é constante atualização que eles têm que receber dos administradores com as novas assinaturas. São sistemas vulneráveis a ataques desconhecidos. Além disso há alguns tipos de ataque, como os de fragmentação que conseguem “enganar” esses IDSs.

Os sistemas baseados em anomalia, ou comportamento podem ser construídos através de mecanismos de limiar (por exemplo detectar atividade anormal através do consumo excessivo de CPU), medidas estatísticas (valores históricos), medidas baseadas em regras de

sistemas especialistas e algoritmos não lineares. A utilização mais comum é a de análise estatística, onde o usuário ou o comportamento do sistema é medido por um número de variáveis ao longo do tempo. Por exemplo, número de *logins*, consumo e duração de recursos durante uma sessão. O maior desafio é achar uma medida sensata para considerar fora da normalidade, evitando uma quantidade grande de falso positivos.

A utilização de algoritmos não lineares para detecção por anomalia é o objeto do presente trabalho. Isso porque estes algoritmos podem ser os algoritmos de redes neurais artificiais, detalhados no capítulo 4. Este capítulo é dedicado a apresentar estudos de diversas Universidades espalhadas pelo mundo a respeito de detecção por anomalia através de redes neurais. Diversos artigos estão disponíveis na Internet e as mais diversas técnicas de redes neurais já foram testadas para detecção de intrusão.

5.1 A BASE DARPA DE DETECÇÃO DE INTRUSÃO

Com o objetivo de testar e melhorar a performance dos sistemas de detecção de intrusão em tráfego real, uma base de dados realista e em larga escala foi construída e patrocinada pela US Defense Advanced Research Projects Agency (DARPA) em 1998[43]. Foram 9 semanas de tráfego observado e capturado entre sites, operando efetivamente como se fossem da força aérea americana, e a Internet. O tráfego e os ataques foram gerados numa rede simulando milhares de *hosts* UNIX e centenas de usuários. Das 9 semanas, 7 foram separadas para dados de treinamento e 2 para dados de teste.

Os ataques resultantes da base foram divididos em 4 grandes categorias: ataques de negação de serviço (DoS); ataques de *exploit* R2L *(remote to local*), isto é, acessos não autorizados de uma máquina remota; ataques U2R (*user to root*), isto é, acessos não autorizados de usuário normal para usuário privilegiado (*root*, ou super usuário); ataques de levantamento de dados, inspeção, *scanning* e outros ataques.

Uma vez colhidos os dados via *tcpdump* e classificados por especialistas, foi realizado um critério de avaliação para que diversos IDSs fossem testados. O critério levava em consideração um aspecto muito importante que é a diferença entre ataques que envolvem uma simples ou poucas conexões como ataques U2R e R2L e ataques que envolvem múltiplas conexões como ataques de DoS. Além disso avaliava a quantidade de falso positivos e falso

negativos gerado pelos IDSs e a velocidade de detecção. Mais detalhes em http://www.ll.mit.edu/IST/ideval.

A base de dados resultante do projeto DARPA, contendo todo o tráfego capturado durante as 9 semanas e com 300 ataques classificados em 32 diferentes tipos encaixados naquelas 4 categorias foi disponibilizada publicamente e praticamente todos os estudos posteriores sobre detecção de intrusão com redes neurais utilizaram esta base para fazerem seus testes de detecção.

5.2 APLICAÇÃO EM HIDS

A detecção de intrusão por anomalia utilizando um algoritmo não linear como redes neurais pode realizada através da análise de diversos parâmetros disponíveis. Monitorar arquivos ou registros importantes, processos, chamadas de sistema, pacotes de rede são alguns exemplos de métricas que podem ser utilizadas para treinar uma rede neural e depois aplicar na detecção. Os trabalhos acadêmicos referentes a cada um dos testes e aplicações construídas sobre o assunto normalmente possuem sua própria classificação do que seja IDS de *host* ou de rede. Por questões de classificação didática, o presente trabalho apresenta o IDS de rede como aquele que monitora os pacotes e protocolos de rede capturados. Os demais parâmetros se encaixam na categoria *host*.

5.2.1 RST CORPORATION

A Reliable Software Technologies Corp. (RST) financiou, em 1999, uma importante pesquisa realizada por A. Ghosh e A. Schwartzbard[29] na aplicação de redes neurais artificiais para detecção de intrusão por anomalia na análise de perfis de comportamento de programas. O sistema foi construído com base na captura de chamadas de sistema feitas pelos programas para monitorar o comportamento dos mesmos e apontar possíveis “desvios” de comportamento que possam significar uma intrusão.

Primeiramente implementaram uma rede Multi-Layer Perceptron (MLP), utilizando o algoritmo *backpropagation* juntamente com o algoritmo *Luck Bucket* para memorizar eventos anômalos recentes através do gerenciamento de um contador. Para os testes a base DARPA foi

utilizada e os resultados foram satisfatórios: 77% dos ataques (entre conhecidos e novos) foram detectados e a taxa de falsos positivos de 3% foi considerada alta. O principal ataque detectado pelo mecanismo foi o ataque U2R (*user to root*).

Depois, com a ajuda de M. Schatz *et all*[29], foi implementado um modelo de rede neural com uma nova topologia. A rede recorrente de Elman demonstrou ser mais efetiva que a anterior na detecção de ataques, principalmente o U2R. Ela foi utilizada no reconhecimento de características recorrentes em trilhas de execução de programas e o resultado aplicado à base DARPA foi de detecção de 77% dos ataques sem nenhum falso positivo. Foi construído um protótipo em linguagem C de implementação de redes Elman para detecção em tempo real.

5.2.2 TEXAS UNIVERSITY

Em 1998, em Austin: Jake Ryan, Meng-Jang Lin e Risto Mikkullainen[31] prepararam um *paper* com os resultados de seus estudos com redes neurais artificiais para detecção de intrusão. A argumentação era de que cada usuário deixa uma impressão própria ao utilizar o sistema. Isso decorre dos comandos que são utilizados por este usuário. Uma rede neural pode ser treinada para aprender a impressão para cada usuário. Se um comportamento de usuário não corresponder mais à sua impressão, o administrador do sistema deverá ser avisado de uma possível intrusão.

Foi construído, então, um sistema de detecção de intrusão *offline*, ou seja, que analisa o tráfego bem depois que ele ocorreu. O sistema aprende os padrões de comportamento legítimo do usuário baseado na distribuição de comandos que o mesmo executa. Os comandos (sem os argumentos) foram coletados durante 12 dias e eram computados o número de comandos e a quantidade de vezes que ele aparecia. Um total de 100 comandos foram cadastrados e classificados com um valor de 0 a 1. Esse valor dependia do número de vezes que o comando aparecia. O valor 0 era para comandos nunca utilizados, 0,1 para comandos usados 1 ou 2 vezes. O número ia aumentando até chegar no valor 1 para comandos que apareciam mais de 500 vezes.

O treinamento foi realizado em uma rede MLP *backpropagation* com 100 entradas correspondentes aos comandos, 30 neurônios na camada oculta e 10 na camada de saída, correspondendo ao número de usuários monitorados. Foram utilizados 8 dias de comandos, escolhidos aleatoriamente, para o treinamento (65 vetores) e os 4 dias restantes para

simulação (24 vetores). O número de épocas testadas para treinamento foram de 30, 50, 100, 200 e 300, sendo que o número de 100 épocas resultou na melhor performance.

A avaliação da simulação era feita pelo valor de ativação dos neurônios de saída. A saída referente a um determinado comando tinha que ser maior no neurônio correspondente ao usuário que executou o comando e tinha que ser maior do que 0,5. Caso contrário, era relatado um falso positivo. A taxa de detecção no melhor caso foi de 96% das intrusões. O número de acertos quanto ao usuário que realizou o comando ficou em 93% e o número de falso positivos ficou em 7%.

5.2.3 UFMG

No Brasil, em 1999, na Universidade Federal de Minas Gerais, Ricardo Bernardo dos Santos, Walmir Matos Caminhas e Luciano de Errico[28] trabalhavam em uma pesquisa de detecção de intrusão *offline* por anomalia através de uma rede MLP *backpropagation* muito parecida com a pesquisa da Universidade do Texas. A idéia era monitorar comandos importantes do sistema operacional em busca de anomalias que indicassem ataques. A topologia da rede utilizada era de 81 neurônios de entrada, 30 neurônios na camada escondida e 15 neurônios de saída.

O sistema operacional utilizado para a monitoração foi o Linux. Os 81 neurônios de entrada representavam 81 métricas representantes dos comandos auditados pela pesquisa. Os 15 neurônios de saída representavam a quantidade de usuários que estavam sendo monitorados. Um total de 405 vetores foi utilizado na pesquisa, sendo que 80% foi utilizado para treinamento e 20% para validação. Além disso 68 desses vetores foram utilizados como anômalos.

Os resultados foram satisfatórios. A taxa de falso positivos encontrada nos dados de teste foi de 3%. A taxa de detecção de comportamento intruso foi de 95,5%, enquanto que 4,5% foi a taxa de falso negativos, ou seja, intrusões que não foram detectadas.

5.3 APLICAÇÃO EM NIDS

As pesquisas para aplicação de redes neurais artificiais para IDS de rede existem em

maior número do que as anteriores. Além disso, aproximam-se mais da proposta do atual trabalho, como será visto em capítulos mais adiante. Um número muito grande de trabalhos acadêmicos foram publicados sobre o assunto. Alguns dos principais são relatados neste tópico.

5.3.1 MIT

Os pesquisadores Richard P. Lippman e Robert K. Cunningham[34], os mesmos que comandaram as pesquisas sobre a base de dados DARPA, foram responsáveis pela condução de testes aplicando redes neurais artificiais no modelo de detecção de intrusão em redes. O trabalho deles foi no sentido de procurar "palavras-chave" específicas para ataques de intrusão dentro do tráfego de rede e a partir delas detectá-los.

Eles utilizaram uma rede MLP *backpropagation* de 2 camadas para detecção de ataques do tipo U2R. A primeira camada tinha k neurônios, a camada oculta $2k$ neurônios e a camada de saída 2 neurônios, representando tráfego normal ou ataque. A melhor performance foi obtida com 30 "palavras-chave".

A taxa de detecção final utilizando a base DARPA ficou em 80%, incluindo detecção de ataques conhecidos e desconhecidos (que não foram treinados). Ataques distribuídos em múltiplas sessões também foram detectados.

5.3.2 NOVA SOUTHEASTERN E GEORGIA UNIVERSITY

Em 1998, James Cannady[30] pela escola de ciências da informação da Universidade Nova Southeastern publicou um artigo sempre citado em trabalhos posteriores sobre o assunto. A proposta era realizar uma prova de conceito afim de mostrar que a maneira tradicional de detecção de intrusão, ou seja a detecção baseada em assinatura, poderia ser substituída por mecanismos de redes neurais.

Segundo Cannady, os IDSs existentes são focados em sistemas especialistas baseados em regras pré-existentes e são necessárias constantes atualizações e análises humanas posteriores. Além disso, se preocupam apenas com fatos isolados e não no estado de transição do sistema durante o ataque. Desta forma, ataques distribuídos e coordenados são difíceis de detectar. Variações de um ataque também podem enganar o sistema. As redes neurais artificiais não possuem esse tipo de problema, à medida que basta treiná-la com um conjunto

de dados ilustrativos e esperar que ela realize a análise sem muitas explicações. Porém, ao contrário dos sistemas especialistas, as redes neurais artificiais não conseguem prover uma resposta definitiva se determinada característica constitui um ataque ou não. Ela fornece uma estimativa da probabilidade de ser ataque. A precisão dessa estimativa depende da experiência do sistema na análise de exemplos (treinamento e validação).

Cannady construiu um protótipo de IDS com uma rede MLP *backpropagation* com uma camada de entrada com 9 elementos, duas camadas ocultas com número de elementos variável e escolhido empiricamente e dois elementos na camada de saída representando tráfego normal ou tráfego de ataque. A função de transferência utilizada é a sigmóide. A base de dados utilizada não foi a DARPA. Cannady utilizou o monitor de rede RealSecure da empresa ISS (Internet Security Systems, Inc) para coletar os pacotes da rede, analisando como ataque ou não. E utilizou o Internet Scanner, também da ISS e o Satan Scanner para atacar a rede. Desta maneira, ele criou amostras de tráfego normal e tráfego de ataque para treinamento da rede neural. As 9 entradas da rede foram escolhidas de acordo com sua importância na descrição das informações transmitidas pelo pacote. São campos dos protocolos IP, TCP, UDP e ICMP. São eles: protocolo, porta de origem, porta de destino, endereço de origem, endereço de destino, ICMP tipo, ICMP código, tamanho do pacote e número identificador do pacote. Um décimo elemento foi acrescentado para o treinamento, o campo ataque (0 ou 1).

Após 10.000 épocas de treinamento e 1.000 épocas de testes, o erro médio quadrático ficou em 0,058 e a taxa de acerto nos testes ficou em 97,5%.

Ainda em 1998 pela Georgia Tech Research Institute (GTRI em Atlanta), Cannady[33] acrescentou às suas pesquisas de detecção de intrusão através de redes neurais as redes não supervisionadas SOM (Self-Oganizing Maps) de Kohonen. Com isso, ele criou um segundo protótipo de detecção que ele chamou de MLP/SOM com a missão de identificar ataques mais complexos como os ataques colaborativos(múltiplos atacantes) e dispersos no tempo(único atacante).

Para criar o protótipo, ele dividiu o tráfego de rede entre normal e ataque, cada qual contendo 180 eventos, ou seja 3 minutos considerando 1 evento por segundo. Cada evento era constituído pelos 50 primeiros caracteres de rede (números dos campos de protocolos) adicionando-se o número da porta de destino no final para facilitar a identificação do protocolo de aplicação. O exemplo aplicado como prova de conceito foi o ataque de força bruta em um servidor FTP, com 9 tentativas de *login* espalhadas em 3 minutos.

Os eventos (50 primeiros caracteres mais porta de destino) serviam como entrada para uma rede SOM 25x20 que classificava os dados em *clusters* e depois calculava uma saída numérica para cada evento através de uma fórmula. A saída numérica era utilizada como entrada da rede MLP e o treinamento e testes eram realizados. O resultado final com testes de força bruta no FTP foi muito bom. A rede acertou 97,2% das vezes com 18 eventos de FTP incluídos no conjunto de dados.

Em 2000, Cannady, juntamente com Brandon Craig Rhodes e James A. Mahaffey[32] pela divisão de ciência da computação da Universidade da Geórgia, publicaram um artigo reafirmando a utilização de redes não supervisionadas SOM como um poderoso suplemento às técnicas tradicionais de IDS e as técnicas de IDS utilizando redes neurais supervisionadas. Segundo eles, os mapas de Kohonen durante o treinamento podem aprender o comportamento normal do tráfego do usuário e apontar as anomarlidades depois de treinadas.

Em seu artigo, eles defendem a adoção de mapas mais especializados e sensíveis ao tráfego que o anterior. Para isso propõem um pré-processamento determinístico. A partir daí apresentam questões teóricas sobre os mapas como sua estrutura bi-dimensional e retangular (podendo ser hexagonal) e o treinamento que consegue segregar vetores de entrada diferentes em categorias diferentes. O grande desafio seria apresentar esses vetores na entrada de maneira correta. No caso de detecção de anomalias, os vetores teriam que espelhar o tráfego de rede de tal forma que pudesse alimentar um mesmo tipo de análise. Eles apontam dificuldades imensas em criar esses vetores devido à grande disparidade dos pacotes de rede: diferentes objetivos, destinos, conexão com duração muito variável, *payload* muito variável e dependente da aplicação.

A solução seria a criação de diferentes monitores de rede, cada um especializado em uma área. Por exemplo, um analisador IP, outro TCP, UDP, DNS, HTTP. Cada qual treinado para reconhecer a atividade normal de um único protocolo e enviar um alarme quando um desvio significante for encontrado. Nem todos esses especialistas precisam ser redes neurais.

Por exemplo, um ataque de fragmentação de IP é facilmente reconhecido através de regras. A idéia é que o tráfego seja decomposto em especialistas de maneira viável computacionalmente. No caso de mapas de Kohonen, é possível existir diversos mapas com diversos vetores diferentes analisando diferentes características dos protocolos. Não se pode, porém, exagerar no número de mapas, pois corre-se o risco de perder a capacidade de correlação de eventos.

Seguindo esse pensamento, eles criaram um mapa de Kohonen com o objetivo específico de identificar ataques de *buffer overflow*, ignorando, dessa forma, qualquer outro tipo de ataque. O experimento foi realizado em linguagem C no Linux e a linguagem Python foi utilizada para melhor visualização dos mapas. Os dados de rede coletados com o *tcpdump*. O protocolo que eles escolheram para monitorar foi o DNS via TCP, devido à sua relativa simplicidade: suas requisições e respostas são definidas para serem simétricas utilizando uma mesma rotina. Uma amostragem de aproximadamente 40 pacotes foi utilizada. Os 30 primeiros para o treinamento da rede e o restante para testar a capacidade de generalização da mesma. Então 2 *exploits* foram submetidos à rede e lançados no mapa e observou-se o resultado. Os vetores de entrada eram muito simples e contavam quantos octetos do pacote se encaixavam em cada categoria de caracter (alfanumérico, numérico, de controle e não-ASCII). Isso porque o ataque de *buffer overflow* tende a ser em pacotes muito grandes e com caracteres binários onde se espera texto.

Os resultados foram muito bons. Os mapas permaneceram estáveis com variação de menos de 10% a cada inicialização randômica de treinamento. Quando aplicados os *exploits*, o mapa mostrou claramente a diferença entre o tráfego normal e o ataque. Enquanto os valores normais da medida de distância do tráfego ficava entre 0 e 3, os *exploits* produziram pacotes que ficaram acima de 8 e algumas aberrações com valores de 30, 630 e até de 13.000 (no caso do pacote que carregava o *payload* do *exploit*).

5.3.3 NEW MEXICO INSTITUTE

Srinivas Mukkamala, Guadalupe Janoski e Andrew Sung em 2002[35] pelo New Mexico Institute publicaram um artigo onde descreviam e comparavam a utilização de redes neurais artificiais e *Support Vector Machines* (SVMs) na detecção de intrusão. A base de dados utilizada por eles foi a base DARPA de 1998, assim como a classificação dos ataques (DOS, R2L, U2R e *scann*) e as 41 variáveis de tráfego características de cada conexão.

Support Vector Machine (ou SVM) é uma técnica de aprendizado de máquina que implementa um mapeamento não-linear dos vetores de entrada para um espaço característico de alta-dimensão, em que o hiperplano ótimo é construído para separar os dados linearmente em duas classes. No caso da detecção de intrusão essas classes são ataque e normal. A SVM classifica os dados determinando uma série de vetores de suporte, membros dos dados de treinamento que contornam o hiperplano no espaço característico. O entendimento profundo das SVMs está além do objetivo deste estudo. As vantagens da SVM são a alta velocidade de resposta e a escalabilidade, visto que as SVMs são relativamente insensíveis ao número de dados de entrada e independente da dimensionalidade do espaço característico.

No experimento prático, foram utilizados 22 ataques diferentes e a classificação foi de +1 para normal e -1 para ataque. Os vetores de entrada tinham 41 campos, mas os 13 mais importantes foram selecionados empiricamente e também foram testados. Foram realizados 4 experimentos. O primeiro treinamento consistiu em 65000 entradas, sendo 10000 para testes com os 41 campos. O segundo consistiu no mesmo número de entradas, porém com os 13 campos mais importantes. O terceiro, 14000 entradas, sendo 7000 para testes com 41 campos e o quarto as mesmas entradas para 13 campos. As taxas de acerto encontradas foram respectivamente 99,6%, 99,57%, 99,53% e 99,52%.

No experimento com redes neurais, foram utilizados 14000 dados de entrada incluindo dados normais e de ataque. 7000 para treinamento e outros 7000 para testes. Uma rede MLP foi utilizada e o resultado foi uma taxa de acerto de 99,48% com 13 entradas e 99,41% com 41 entradas. A grande diferença, entretanto, foi o tempo de resposta do método. Enquanto as SVMs demoraram de 1 a 2 segundos para testar os experimentos com 7000 entradas, a rede neural demorou de 30 a 40 minutos para os mesmos testes. Eles terminam o artigo concluindo que como as SVMs são limitadas a fazerem classificações binárias, elas possuem propriedades superiores de velocidade, escalabilidade e generalização em relação às redes neurais.

5.3.4 BRASIL

Renato Maia Silva e Marco Antônio G.M. Maia[36] do Centro de Estudos em Telecomunicações (CETUC) da PUC-RJ publicaram um artigo onde demonstram a utilização de uma rede MLP para implementar uma rede neural artificial para detecção de intrusão. A base de dados utilizada foi a DARPA. Houve um pré-processamento para adaptar os dados ao

intervalo [-1;1]. A *toolbox* de redes neurais do software Matlab R11 foi utilizada. A rede era composta por duas camadas ocultas com 10 neurônios cada. A função de ativação era tangente hiperbólica. Quatro testes foram realizados. O primeiro possuía 9 entradas referentes às características básicas do TCP/IP e uma saída (valor -1 para normal e 1 para ataque). O segundo possuía 41 entradas (características da conexão) e uma saída, a mesma do anterior. O terceiro possui as 9 entradas do primeiro e uma saída (-1 normal, 0 ataque do tipo Neptune e 1 ataque do tipo Smurf). O quarto também possuía 9 entradas, porém 4 saídas (normal, *neptune, back e smurf*).

Os resultados obtidos foram muito bons. A rede 1 apresentou em seu melhor resultado uma taxa de acerto de 96% e 3,9% de falso positivo. A rede 2, 95% de acerto e 1,4% de falso positivo. A rede 3, 97,5% de acerto e 0% de falso positivo. A rede 4, 92% de acerto e 0% de falso positivo.

Em 2004, Fábio Bombonato e Flávia E.S. Coelho[37] divulgaram um artigo apresentando um projeto que batizaram de beholder. Consiste em um IDS programado em linguagem C que captura o tráfego de rede utilizando a biblioteca libpcap. A partir daí com o auxílio de duas ferramentas de redes neurais: EasyNN e SNNS, a rede MLP é treinada e testada. Os campos de entrada foram os mesmos do MLP de Cannady. A diferença é que os campos IP origem e IP destino foram divididos em 4 partes cada para diminuir a magnitude. O treinamento foi realizado aplicando-se um ataque manual de *port scanning* com a ferramenta *nmap*. A rede possuía 15 entradas, uma camada oculta com 4 neurônios e uma saída binária: ataque ou normal.

Os resultados novamente foram muito satisfatórios. A taxa de validação ficou em 99.58%. O treinamento durou 44.059 épocas. Eram 5577 pacotes de treinamento, 200 de validação e 18 de teste. A taxa de aprendizado era de 0,3 e o *momentum* de 0,8. Os resultados após o treinamento foram de 100% de acerto para tráfego normal e 99,5% para diversas técnicas de *port scanning* exceto o TCP *connect()* que não foi detectado, segundo eles por uma semelhança com padrões de situação normal.

5.3.5 UBILAB LABORATORY

Luc Girardin da UBILAB, em 1999[39], publicou um artigo que viria a ser muito influente em pesquisas posteriores. Girardin apresenta uma representação visual para análise da atividade de rede. Ele utilizou uma rede não supervisionada SOM de Kohonen para

projetar os eventos de rede em um espaço apropriado para visualização. Após isso, e ao contrário de outros sistemas, seria necessária uma análise humana para explorar as atividades de rede à procura de padrões de intrusão.

Segundo Girardin, o ser humano tem uma habilidade natural em lidar com a complexidade e uma metáfora visual desperta o senso de tomada de decisões. Abre a possibilidade de realizar julgamentos a respeito de certa coisa: se é normal ou anormal, crítico ou isolado, benigno ou maligno. Uma classificação topológica através de algoritmos de inteligência artificial para visualização e exploração de *logs* provê novos caminhos para explorar, trilhar e analisar intrusos. Torna a monitoração amigável e suporta uma representação mental em busca do inesperado. Cada vetor de entrada representa um ponto no espaço de alta-dimensão. A distância entre dois pontos é inversamente proporcional à sua similaridade. A redução de dimensionalidade provocada pelos mapas de Kohonen, representa os vetores em 2 ou 3 dimensões preservando a informação essencial.

Para realizar a prova de conceito a base dados utilizada foi a do Exploration Shootout project de Georges Grinstein de 1997. Essa base continha 4 diferentes tipos de ataques: *IP spoofing, ftp password guessing, network scannig* e *network hopping*. A distância euclidiana é o método utilizado para medir a disparidade entre os vetores e, em conseqüência classificá-los quanto à semelhança. Para tentar compensar a diferença de magnitude entre os campos *flags*, *type of service* e os demais campos, esses atributos foram tratados com uma dimensão para cada valor, no vetor de entrada. Foi utilizado um SOM de duas dimensões. O processo de aprendizado foi realizado da seguinte maneira: inicializa-se aleatoriamente o valor dos pesos; apresenta-se o vetor de entrada; encontra-se o neurônio vencedor(menor distância); modifica-se o peso dos vetores próximos a ele; apresenta-se novo vetor de entrada e repete-se os passos seguintes. A cada entrada uma medida de qualidade do mapa pode ser calculada utilizando-se o erro de quantização, a distância do vetor de entrada para o vetor que mais vezes venceu. O tempo de processamento computacional necessário para criar os mapas é proporcional à sua dimensionalidade e o número de neurônios, mas não depende do tamanho dos dados de entrada. Entretanto, dados de entrada com relações muito complexas demoram mais para convergir durante o treinamento. Essas relações devem ser definidas empiricamente.

A figura 5.1 ilustra a visualização do mapa de Girardin. Além das características explicadas na própria figura, as características a seguir foram utilizadas: a cor dos neurônios corresponde ao valor do peso para cada um; o tamanho corresponde ao número de eventos

mapeados naquele neurônio, permitindo que se identifique imediatamente os tipos de tráfego mais e menos freqüentes; a cor de fundo representa o erro de quantização média de todas as entradas resultantes do neurônio. Um erro alto é conseqüência de eventos raros e atípicos, o que pode facilitar bastante a análise de detecção por anomalia à medida que os eventos anômalos são eventos raros e atípicos.

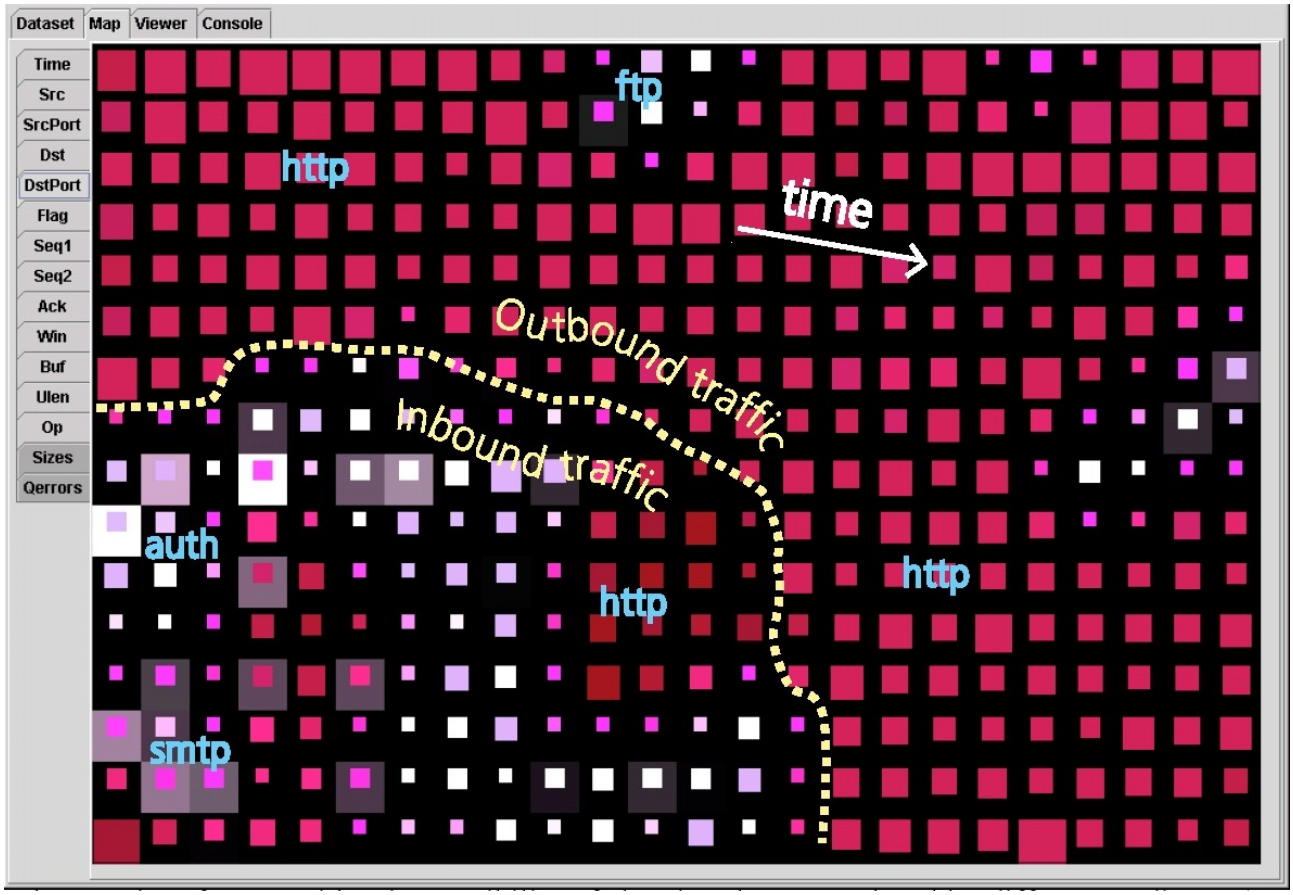


Figura 5.1 – Mapa de Kohonen (Girardin, 1999)

Os campos utilizados para formar os vetores de entrada foram capturados via *tcpdump* eram os seguintes: *timestamp*, endereço de origem, endereço de destino, porta de origem, porta de destino, *flags* (TCP), número seqüencial de dados(TCP), número seqüencial esperado para o retorno(TCP), número de bytes disponíveis no buffer receptor(TCP), indicação de urgência dos dados(TCP) e tamanho do pacote(UDP). Cada arquivo tinha aproximadamente 30MB e os campos eram separadas por vírgulas. Os arquivos continham tráfego normal e os 4 tipos de ataques(um em cada arquivo) já citados.

O grande problema da análise é não conseguir identificar claramente quais hosts estão

se comunicando com quais. Entretanto é relativamente simples identificar o tráfego que desvia do normal. O ataque de IP *spoofing* do primeiro arquivo, por exemplo, foi reconhecido ao se observar o comportamento não usual de algumas conexões que não completaram o *three-way handshake* e ficavam isoladas no mapa. Além disso, haviam muitos pacotes redundantes, além de outros comportamentos estranhos. No segundo arquivo, um elevado número de conexões FTP iniciadas da rede interna, com direção a 12 *hosts* diferentes criavam mais de 60 conexões suspeitas que não estavam ligadas com nenhum tráfego de resposta. O diagnóstico foi ataque de *password guessing*. Os principais parâmetros dessa descoberta foram: o intervalo curto de tempo em que os *hosts* eram testados, um após o outro, o endereço de origem era sempre o mesmo, as portas de origem eram muito próximas uma das outras, a porta de destino era sempre *ftp*. O ataque de *scann* foi identificado facilmente, pois o *host* de destino respondeu à sondagem de pacotes da mesma origem em 40 portas diferentes em 1 minuto. O resultado no mapa foi que apenas 3 neurônios foram os vencedores e concentraram todos os pacotes de ataque. O último ataque, *network hopping* é quando uma conexão é usada para iniciar outra conexão para fora, possivelmente para confundir o defensor. Ele foi descoberto através de uma correlação de tráfego de entrada e saída. Por exemplo muitos pacotes de *telnet* direcionados a um *host* interno, mas idênticos a pacotes de *rlogin* destinados à rede externa.

Girardin termina o artigo reconhecendo que o trabalho ainda está no seu início. Os *logs* utilizados não são suficientes para criar padrões complexos e de fácil visualização. Ele recomenda um trabalho com outras fontes de *log* como *firewall,* roteadores e *sniffers* juntos e correlacionados. Também recomenda um uso mais sofisticado de noção de tempo, criando outras características, como periodicidade por exemplo.

5.3.6 OHIO UNIVERSITY

Em 2003, Manikantan Ramadas, Shawn Ostermann e Brett Tjaden[40] publicaram um artigo descrevendo um sistema chamado de *Integrated Network-Based Ohio University Network Detection Service* (INBOUNDS), um IDS de rede desenvolvido pela universidade de Ohio. O *Anomalous Network-Traffic Detection with Self-Organizing Maps* (ANDSOM) seria um módulo do sistema. O artigo é um resumo da tese de mestrado de Ramadas, orientado por Ostermann. O diagrama do INBOUNDS é mostrado na figura 5.2.

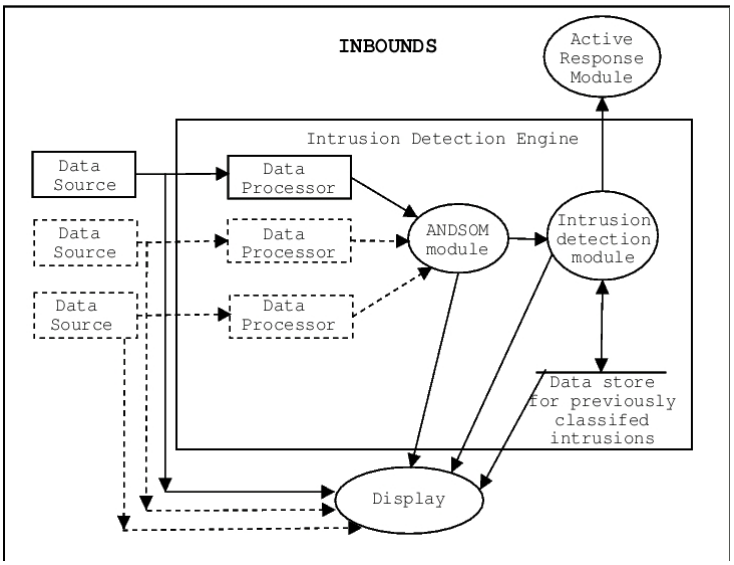


Figura 5.2 – Arquitetura INBOUNDS (Ramadas/Ostermann/Tjaden, 2003)

O módulo Intrusion Detection é o coração do sistema, pois é onde se toma a decisão *online* se o tráfego é um ataque ou não. O módulo *Display* é uma GUI onde pode-se acompanhar o tráfego. o módulo de *Active Response* realiza ações de resposta, como adicionar regra ao *firewall*, limitar a banda, fechar uma conexão TCP com o *flag RST*, etc. Ele destaca ainda que outros módulos poderão ser adicionados no futuro, como por exemplo um IDS baseado em assinatura (como o SNORT), ou um modulo Bayesiano.

O módulo *Data Source* é o módulo que captura os pacotes da rede através do programa *tcpurify* (de Ethan Blanton, universidade de Ohio). Ele reporta apenas os 64 primeiros bytes de cada pacote (protocolos TCP e UDP). Depois disso, os dados vão para o módulo *Data Processor*, onde o programa de análise de conexão desenvolvido por Ostermann e chamado *tcptrace* processa o tráfego e fornece a saída de 3 maneiras diferentes para cada conexão: *Open* quando uma nova conexão é aberta na rede. Existe um campo *Status* indicando o valor 1 se apenas um dos lados da conexão mandou o pacote com *flag SYN* e 0 para os demais casos e para os pacotes UDP; *Update* é reportado periodicamente durante o tempo de vida da conexão. Possui os campos INTER(interatividade da conexão, número de questões por segundo), ASOQ(tamanho médio das questões), ASOA(tamanho médio das respostas), QAIT(intervalo de tempo entre pergunta e resposta, por segundo) e AQIT(intervalo de tempo entre resposta e pergunta, por segundo); *Close* quando uma conexão anteriormente aberta é fechada na rede. Também há um campo *Status*, 0 quando os pacotes com *flag FIN* foram mandados e 1 se a conexão foi fechada com *flags RST*. Em comum, os campos *timestamp, protocol, source host:port, destination host:port*.

O módulo ANDSOM precisa ser treinado para depois entrar em operação. O treinamento é feito de acordo com o tipo de serviço a ser modelado (*web, email, telnet, etc*). Por exemplo, se o objetivo for modelar o tráfego *web*, então é feita uma coleta de dados do maior número possível de tráfego *web* normal (usa-se o SNORT para evitar que ataques sejam capturados nesse tráfego). Desta maneira, a rede SOM aprende o comportamento normal de tráfego. O vetor de entrada é composto por 6 campos: os cinco campos da saída *Update* do módulo anterior mais o campo DOC, que é o tempo de duração da conexão, ou seja, a diferença do *timestamp* da saída *Open* para o *timestamp* da saída *Close* para determinada conexão. Os campos QAIT e AQIT são tratados em base logarítmica, devido à diferença de magnitude. Após isso, todos os campos são normalizados baseados na variância de cada dimensão. O software utilizado para treinamento é o software livre desenvolvido em C SOM_PAK e a visualização é feita pelo *toolbox* SOM para software MatLab.

Após o treinamento, o módulo ANDSOM entra em operação. Ele recebe as mensagens de conexão (*Open, Update* e *Close*) e as converte no vetor de entrada de 6 dimensões. Um neurônio vencedor é escolhido para cada vetor. Depois normaliza-se o vetor com os valores estatísticos recebidos do treinamento. Um neurônio vencedor é escolhido. Se a distância deste vencedor para o vencedor em 6 dimensões for maior do que 2 unidades, então o tráfego é considerado anômalo. Os experimentos foram realizados em cima de tráfego DNS e HTTP. Por exemplo, o treinamento de tráfego DNS foi realizado com 8857 vetores e foi construído um mapa 19x25. Um *exploit* para o software de DNS BIND 8.2.x foi testado e foi classificado corretamente como anômalo. A distância do neurônio normalizado vencedor do *exploit* para o neurônio vencedor em 6 dimensões ficou em 22.314, ou seja, muito maior do que 2 unidades.

Após algumas experiências com HTTP, SMTP e DNS, eles concluíram que o sistema é bastante eficiente em detectar ataques de *buffer overflow*, porque eles diferem bastante do padrão normal de tráfego. Porém ataques que lembram o tráfego normal não foram detectados com eficiência.

5.3.7 UNIVERSITY OF NEW BRUNSWICK

Em 2004, John Zhong Lei e Ali Ghorbani[41] publicaram um artigo pela Faculdade de Ciência da Computação da Universidade de New Brunswick em Fredericton no Canadá. O objetivo do artigo era propor um novo método de detecção de intrusão baseado em aprendizagem competitiva e comparar esse novo método ao SOM. Esse novo método é

derivado do Standard Competitive learning Network (SCLN) e foi chamado de Improved Competitive Learning Network (ICLN).

O SCLN é composto de uma única camada, onde os neurônios de entrada estão totalmente conectados aos neurônios de saída. Quando um vetor de entrada é apresentado a rede há um neurônio vencedor e apenas o seu peso é reajustado através de parâmetros como peso anterior, taxa de aprendizagem e vetor de entrada. A performance do SCLN é extremamente dependente do número de neurônios de saída e da inicialização do vetor de pesos. Por sua vez, o ICLN utiliza o mesmo critério para reajustar o peso do neurônio vencedor, porém, todos os outros neurônios têm o peso reajustado por uma função chamada *Gaussian Kernel*. Através desse parâmetro, os demais pesos se afastam do vetor de entrada, enquanto que o vencedor se aproxima. Este processo elimina as limitações do SCLN, além de acelerar a classificação dos dados em *clusters*. O ICLN utiliza basicamente a mesma fórmula do SOM para reajustar os pesos. A diferença é que no SOM apenas os pesos da vizinhança são reajustados dependendo da distância e da variância e no ICLN todos os pesos são reajustados também dependendo da distância.

Na prática, os experimentos foram realizado utilizando a base de dados DARPA. 101.000 conexões foram utilizadas para treinamento e 400.000 para teste. De 21 tipos de intrusão, apenas 7 foram utilizados no treinamento, para testar a capacidade de descobrir novas intrusões. As características da conexão utilizadas para os vetores de entrada foram protocolos (tcp, udp ou icmp), tipo de serviço (70 diferentes, como http, smtp, etc) e *flags* (11 diferentes). Essas características foram mapeadas em valores quantitativos na preparação dos dados. O número de neurônios de saída foi testado com vários valores iniciais (9,15,18 e 20). Os mesmos experimentos foram realizados para o SOM e para ICLN e o caso de 15 neurônios de saída iniciais foi analisado mais a fundo. No caso do SOM, o mapa 3x5 resultou em 5 neurônios obsoletos, ou seja, sem nenhum vetor classificado. O número de *clusters*, portanto ficou 10 e o tempo de classificação foi de 2162s. A precisão ficou em torno de 98%. O ICLN resultou em 6 *clusters* para todos os valores iniciais de vetores. A precisão ficou em torno de 98% e o tempo de classificação foi de 608s.

A conclusão é que o novo método ICLN tem uma performance melhor do que o SOM em termos computacionais, sendo que a precisão é praticamente a mesma. Além disso, o número de *clusters* independe do número de neurônios de saída iniciais.

5.3.8 KING MONGKUT'S UNIVERSITY OF TECHNOLOGY THONBURI

Chaivat Jirapummin, Naruemon Wattanapongsakorn e Prasert Kanthamanon em 2003[42] publicaram um artigo pela King Mongkut's University of Technology Thonburi em Bangkok, na Tailândia. A proposta era a de um sistema de detecção de intrusão híbrido baseado em Self-Organizing Maps (SOM) e Resilient Propagation Neural Network (RPROP) para visualização e classificação de padrões normais e de intrusão. O RPROP é uma versão acelerada da rede supervisionada MLP com o algoritmo *backpropagation.*

A base de dados DARPA foi utilizada nos experimentos e foram testados os ataques *SYN flooding* e *port scanning*. 121.820 padrões de treinamento foram divididos em 8 sessões. Primeiro cada sessão foi classificada em *clusters* por um SOM de 1.234 unidades. Depois os dados passavam por uma rede RPROP com 70 neurônios na camada de entrada, 12 na oculta e 4 na saída (indicando tráfego normal, ataque *Neptune*, ataque *Portsweep*, ou ataque *Satan*). Foram realizados 20 diferentes treinamentos e testes.

Dois testes foram escolhidos para avaliação de resultado. O teste 1 continha 98.648 padrões capturados da mesma rede do treinamento. O teste 2 continha 126.373 dados de uma rede diferente. O resultado do teste 2 foi de 99% de detecção e 0,05% de falso positivos para ataques de Neptune, 97% de detecção para *Port Sweep* e 4,19% de falso positivos, 90% de detecção para *Satan* e 4,49% de falso positivos. Além disso, os tipos de ataques ficaram claramente separados em *clusters* no mapa SOM.

5.3.9 RENSSELAER POLYTECHNIC INSTITUTE

Em 2002, Chandrika Palagiri, Alan Bives, Rasheda Smith, Boleslaw Szymanski e Mark Embrechts publicaram pelo Rensselaer Polytechnic Institute de Nova Iorque um artigo descrevendo um sistema híbrido de redes neurais artificiais para detecção de intrusão. Utilizou-se uma rede SOM para classificação dos dados e uma rede MLP *backpropagation* para detecção de intrusão. Ao relatar trabalhos relacionados, Chandrika citou a diferença deste sistema para alguns dos citados acima. A diferença na parte de treinamento da rede supervisionada para o sistema de Lippmann do MIT é que este procura por palavras chave dentro do *payload* dos pacotes, enquanto que o deles não utiliza o *payload*, além de monitorar as portas utilizadas. A diferença na parte não supervisionada para o de Girardin é que eles não tem necessidade dos efeitos visuais do mapa e nem precisam de análise humana. A rede SOM

apenas divide o tráfego em *clusters* fornecendo uma classificação.

A análise de intrusão começa com a necessidade de retirar parâmetros do tráfego de rede. A idéia inicial é saber quantas vezes um *host* é acessado em um determinado intervalo de tempo e quais portas foram acessadas neste intervalo de tempo. No processo de treinamento as portas mais importantes são escolhidas, através de um algoritmo. Primeiramente são escolhidas as portas conhecidas e um número de portas extras a serem monitoradas. Após um intervalo de tempo no treinamento, essas portas conhecidas são acrescentadas do número de portas extras escolhido *n*, que corresponde às *n* portas mais acessadas no intervalo. O resultado final é um número N de portas acessadas em um intervalo. Além disso, o número de endereços de origem M é computado. Assim, a entrada da rede fica com um tamanho N*M. Por exemplo, 5 portas foram escolhidas para serem monitoradas e 4 diferentes origens foram identificadas. A rede teria 20 entradas, a primeiro as 5 portas da primeira origem, depois 5 da segunda origem e assim sucessivamente. O grande problema dessa técnica é variação constante do número de entradas da rede, mudando praticamente a cada intervalo de tempo. Uma rede supervisionada MLP necessita de um número de entradas estático, tornando o método inutilizável. A solução encontrada foi utilizar a rede SOM para classificar os dados de entrada em um número fixo de *clusters* que servem de entrada para a rede MLP. Assim, é possível que durante a fase de aprendizagem, o número de entradas seja diferente do número na fase de detecção, desde que o número de clusters seja igual.

Um software em C é utilizado para fazer o treinamento da rede e um software em java é utilizado para os testes, a partir dos pesos da rede treinada (salvos em um arquivo). Os dados são normalizados em valores entre 0 e 1. O detector java é construído para funcionar em tempo real. Para os testes, a base DARPA foi utilizada. Foram testes bem modestos. Primeiro foi testado a capacidade de detectar todos os ataques da base. O resultado foi 100% do tráfego normal corretamente classificado. 24% do tráfego de ataque corretamente classificado e 76% de falso positivos. O segundo teste, em um ataque específico (DOS *sshprocesstable*) foi bem mais animador. 100% do tráfego normal foi corretamente classificado e 100% do tráfego de ataque foi corretamente classificado.

5.4 CONSIDERAÇÕES FINAIS

Neste capítulo foi visto que as redes neurais artificiais têm sido amplamente estudadas como um método de detecção de intrusão alternativo ao método baseado em assinaturas.

Os estudos, apesar de serem bem recentes, percorrem os mais diversos parâmetros tanto na escolha do paradigma da rede (supervisionada ou não supervisionada), quanto na escolha da configuração da rede (número de camadas, número de neurônios), quanto na escolha dos parâmetros de entrada, quanto na escolha da base de dados para testes. Possibilidades infinitas de utilização ainda estão em aberto, até que se construa um modelo sólido de utilização.

Praticamente todos os métodos apresentaram bons resultados práticos. Apesar disso, são resultados ainda muito limitados ao meio acadêmico e com pouca aplicação prática comercial. Por conta disso, ainda não existe um sistema comercialmente viável de utilização de redes neurais para detecção de intrusão.

A proposta deste trabalho é apresentar uma configuração baseada nos trabalhos estudados. A grande inovação, entretanto, é na base de dados utilizada para a coleta de amostras de ataque para treinamento e testes. Ao invés de utilizar a base DARPA, já antiga(1998) e pertencente a um contexto específico, ou utilizar ataques simulados manualmente, pretende-se utilizar uma base coletada de uma rede real.

Seria desejável que esta rede fosse a de produção. Todavia, coletar dados de ataque de uma rede em produção requer que essa rede seja atacada, o que não é desejável. Requer, também, que exista um mecanismo que classifique corretamente todos os ataques. Justamente este mecanismo(IDS existente) está sendo criticado no trabalho.

A solução encontrada é configurar uma rede, com a maior proximidade possível da rede de produção(endereçamento, serviços, sistemas operacionais, etc) para coletar os ataques que seriam sofridos na produção e treinar redes neurais artificiais específicas para cada ataque. Esta rede, construída para ser atacada é o mecanismo chamado *honeynet* e será visto no próximo capítulo.

6 HONEYNET

Honeynet é uma rede construída exclusivamente para ser invadida. O objetivo principal é investigar os métodos utilizados pelos invasores para acessar o sistema e, posteriormente, as ações realizadas por eles. Esta rede é constituída por *hosts* intensionalmente vulneráveis, com um endereço válido na Internet e por toda uma estrutura de captura, análise de dados e proteção da rede. O atacante acha que está invadindo o sistema e tirando proveito da situação, mas na verdade ele está fornecendo subsídios para que se possa estudar suas técnicas de sondagem, invasão e comprometimento do sistema. O termo mais genérico de *Honeynet* é *honeypot*, ou seja, pote de mel. Uma isca, ratoeira, armadilha. Na realidade uma *honeynet* é um tipo de *honeypot* onde os sistemas são reais. Existe um outro tipo, onde *softwares* simplesmente emulam os sistemas operacionais e a própria rede.

Historicamente, a primeira experiência a respeito de análise de intrusões surgiu em 1988, quando Clifford Stoll fez um relatório sobre uma invasão sofrida nos laboratórios da LBL (Lawrence Berkley Laboratory). Ao invés de expulsar os atacantes (ou “Black Hats”), Clifford decidiu estudar suas atividades, registrando seus passos e sua origem. O estudo durou cerca de 1 ano e revelou aspectos importantes, o motivo dos ataques e quais redes interessavam mais aos atacantes. Em 1992, Bill Cheswick publicou um artigo descrevendo os resultados do acompanhamento de invasões em um sistema da AT&T especialmente projetado para ser comprometido. Steven Bellovin também participou do projeto, construindo ferramentas utilizadas para armadilhas e captura de *logs*. Em 1998 surge o termo *honeypot*. Foi neste ano que Fred Cohen desenvolveu o DTK (Deception Toolkit), uma ferramenta de código aberto feita com o objetivo de enganar os atacantes, emulando diversas vulnerabilidades e coletando informações sobre os ataques sofridos. Em 1999 Lance Spitzer liderou um grupo de pesquisadores e especialistas em segurança criando o Honeynet Project, que será mais detalhado adiante no capítulo. A partir daí, o termo *honeynet* ganhou repercussão mundial e o desenvolvimento de novas ferramentas e sistemas de defesa passaram a ser realizados no mundo todo.

6.1 HONEYNET PROJECT

O *Honeynet Project*[44] (www.honeynet.org) foi criado por Lance Spitzer e tem como principal objetivo compartilhar experiências, documentação, ferramentas e conhecimento a respeito de invasões e técnicas utilizadas por invasores. Foi criada em 1999 e funciona através de uma constante e gratuita pesquisa de seus membros e colaboradores.

Além do *Honeynet Project*, existem diversos outros projetos de *honeypots* espalhados pelo mundo. Muitos deles estão interligados através do fórum *Honeynet Research Alliance*, criado em 2002. Eles não são membros do *Honeynet Project* e sim entidades independentes que contribuem com a aliança com o propósito de pesquisar e desenvolver soluções de *honeynet*, além de compartilhar as lições aprendidas. Os membros da aliança estão espalhados pelo mundo: Alemanha, China, Espanha, Portugal, França, Itália, Paquistão, Brasil, Inglaterra e outros.

Cada projeto, como entidade independente, possui sua própria rede, seus próprios *papers* e ferramentas para estudo das atividades dos *black hats*. Algumas ferramentas e estudos importantes são publicados pelo *Honeynet Project*. No Brasil, por exemplo, existe o *HoneynetBR* que faz parte da aliança mundial, mas possui a sua própria aliança, a *Brazilian Honeypots Alliance*, formada basicamente por Universidades e organizações espalhadas por todo o país. Além disso, existem grupos de pesquisadores independentes, como a *Honeypot-BR* que não participam das alianças.

No portal *honeynet.org* é possível encontrar todo o tipo de documentação a respeito do assunto, incluindo artigos, apresentações, monografias e textos. São documentos revisados pela equipe do projeto. Também são disponibilizados documentos individuais escritos pelos membros do projeto e apresentados em conferências. Além disso, existe um livro: “Know your Enemy”, escrito por especialistas do projeto e reconhecido como um dos melhores na área. É possível, também, baixar ferramentas de código aberto para a construção de *honeynets*. Outra prática muito interessante é a construção e disponibilização de desafios. São análises de ataques reais feitas pelos diversos projetos de *honeynet* e compartilhadas na forma de desafios para que toda a comunidade de segurança possa analisar e descobrir como o ataque foi realizado.

O *Honeynet Project* preparou uma apresentação do projeto, onde identifica as

principais ameaças na Internet: são centenas de *scanners* todos os dias; os ataques são inicialmente focados em sistemas Win32; a expectativa de vida para um sistema Win32 vulnerável é abaixo de 3 horas, enquanto que um sistema linux vulnerável é de 3 meses; um *honeypot* vulnerável pode ser compromissado em 15 minutos manualmente e em menos de 1 minuto através de *worms*; os atacantes podem controlar milhares de sistemas em uma rede zumbi (*botnet*).

Os motivos para a invasão variam muito, mas estão se tornando cada vez mais criminosos, em busca de dinheiro e menos por curiosidade, em busca de conhecimento. Os alvos são os usuários em massa, despreocupados com a segurança. O projeto descobriu algumas tendências interessantes: os ataques tendem a ser originados de países economicamente deprimidos (por exemplo a Romênia); os ataques estão se tornando continuamente mais sofisticados; as ferramentas também avançam rapidamente e são em sua maioria automáticas (*worms*).

Ferramentas e ataques avançados cada vez mais comuns incluem *botnets* e ataques de *phishing*. As redes zumbis são redes enormes (milhares ou dezenas de milhares de sistemas) controladas por apenas um usuário através de comandos automatizados. Após comprometerem o sistema com um *exploit* bem sucedido, o invasor instala um binário através de HTTP, FTP, ou TFTP. Quando iniciado, o binário conecta-se a um servidor IRC mestre utilizando um *nickname* especial e uma senha conhecida pelo invasor. Os sistemas mais facilmente infectados são Windows Xp e 2000 e as portas mais utilizadas são a 445, 139 e 135 TCP e 137 UDP. As redes são utilizadas para realizar ataques de DDoS, *scanners*, *spams* e outros. Em 4 meses, o *Honeynet Project* cadastrou mais de 100 *botnets*. Apenas um canal possuía mais de 200.000 endereços IP. Apenas um computador tinha sido compromissado por mais de 16 invasores de *botnets*. Foi estimado mais de 1 milhão de sistemas compromissados nesses 4 meses.

O ataque de *phishing* consiste em mandar e-mails falsos ou *spams* escritos como se fossem de bancos, órgãos do governo ou instituições não-governamentais com a intensão de capturar dados importantes do usuário, como senhas, números de cartão de crédito, IDs, etc. É um ataque de engenharia social direcionado a usuários. Normalmente, máquinas comprometidas ou *botnets* são usadas para mandar os e-mails com origem falsa ou hospedar páginas falsas fingindo ser de instituições.

6.2 HONEYPOTS

Um *honeypot* é um sistema de informação cujo valor está no acesso não autorizado ou uso ilícito de seus recursos (Spitzner, 2003). É um elemento de segurança que diferentemente de um *firewall* ou um IDS não resolve um problema específico. É uma ferramenta extremamente flexível construída para identificar qualquer tipo de ataque. Teoricamente, *honeypot* é um recurso que não possui tráfego autorizado ou legítimo, isto é, qualquer interação que houver é uma atividade não autorizada ou maliciosa. Lance Spitzner, em 2003, identifica suas principais vantagens e desvantagens.

A primeira vantagem é a coleta de poucos dados, porém de alto valor. Todo tipo de tráfego neste mecanismo é uma atividade maliciosa, portanto tudo que for coletado é uma tentativa de ataque de grande valor; outra vantagem é a descoberta de novas ferramentas e táticas de invasão; a necessidade de recursos computacionais mínimos para sua construção constitui outra vantagem; um *honeypot* consegue detectar e capturar tráfego IPv6 e tráfego criptografado; tem a capacidade de coletar informações profundas e possui uma simplicidade de configuração. Existem, entretanto, duas dificuldades na sua utilização. A primeira é a visão limitada que ele possui, conseguindo enxergar apenas o tráfego que interage diretamente com ele e ignorando outros sistemas existentes; a segunda, é o risco que se corre de um atacante conseguir dominá-lo e utilizá-lo para comprometer outros sistemas.

Honeypots são divididos em dois tipos, de acordo com o seu nível de interação com o atacante. *Honeypots* de baixa interação são basicamente *softwares* que emulam a existência de uma rede de sistemas e procuram interagir com o atacante tentando fazê-lo pensar que ali existem sistemas reais. *Honeypots* de alta interação são sistemas reais, construídos propositalmente vulneráveis e desatualizados para que o atacante consiga invadir e revelar suas táticas e ferramentas. Este segundo tipo é chamado de *honeynet* e possui não apenas *hosts* vulneráveis, mas elementos de controle e análise de dados.

Normalmente, constrói-se um *honeypot* por dois motivos: utilizar na produção ou realizar pesquisa. Quando usado na produção, o objetivo é prevenir, detectar ou ajudar uma organização a responder a um ataque. Quando usado para pesquisa, possui o objetivo de coletar informação e a partir daí analisar das mais diversas maneiras e entender o modo de agir dos atacantes. Freqüentemente *honeypots* de baixa interação são utilizados para produção

e de alta interação para pesquisa. Contudo, tanto os de alta quanto os de baixa podem ser utilizados para os dois motivos.

É interessante comentar um pouco mais sobre os seguintes objetivos de um *honeypot*: prevenção, detecção e resposta a incidentes. Para prevenir um ataque contra ataques automáticos, como *worms,* existe o *sticky honeypot*, que através da monitoração do espaço de IPs não utilizados consegue diminuir a velocidade dos *scanners* podendo até pará-los. Eles utilizam os IPs não utilizados emulando um sistema real. Isso pode confundir também um atacante real, fazê-lo perder tempo e recurso interagindo com o *honeypot,* enquanto a organização detecta e prepara a resposta. O tipo de honeypot mais utilizado para prevenção é o de baixa interação. A detecção utilizando um *honeypot* pode ajudar a resolver algumas dificuldades da detecção tradicional através de IDS. Ele reduz a quantidade de falso positivos pois captura pequena quantidade de dados de grande valor. Captura ataques novos e desconhecidos como *exploits* e *shellcodes*. Por fim, consegue trabalhar com tráfego encriptado e IPv6. O tipo de *honeypot* mais utilizado para esse fim também é o de baixa interação. Para a resposta a incidentes, a vantagem de um *honeypot* é a disponibilidade de ser colocado *offline* a qualquer momento para uma análise forense completa, o que é impossível de fazer com um servidor de produção. Outra vantagem é a pequena quantidade de dados de grande valor, ante a grande quantidade de dados que não interessam de um servidor de produção. O melhor *honeypot* para realizar esse tipo de tarefa é o de alta interação, devido à necessidade de conhecimento profundo do ataque e das ferramentas.

6.2.1 HONEYPOTS DE BAIXA INTERAÇÃO

As grandes vantagens deste tipo de *honeypot* são a sua simplicidade de manutenção e o pequeno risco que ele oferece. O usuário só precisa escolher quais serviços e sistemas operacionais ele deseja emular e monitorar. O atacante não consegue utilizá-lo para atacar outros sistemas. Todavia, a informação capturada é bastante limitada e é bem mais fácil para o atacante perceber que trata-se de um *honeypot*.

O *software* de baixa interação mais popular é o *Honeyd*, um *software* de código aberto criado por Niels Provos e que trabalha monitorando os IPs que não estão sendo utilizados. Quando uma requisição chega para esses IPs, o *Honeyd* intercepta a conexão e pode interagir com o invasor simulando ser a vítima. Com isso, o *software* registra e detecta qualquer

conexão TCP ou UDP. É possível configurar a monitoração de serviços específicos, como FTP. Quando o atacante conecta-se à porta 21, o *Honeyd* interage com o atacante e registra todas as atividades. O grande problema é que a interação apenas ocorre quando o atacante executa as ações esperadas pelo *software*. Se o atacante faz algo que o emulador não espera, consegue descobrir através de uma mensagem de erro que trata-se de uma armadilha. Além disso, o *Honeyd* emula sistemas operacionais, tanto através dos serviços quanto através da pilha TCP/IP (para enganar os *scanners*).

Existem outros diversos *softwares* de baixa interação. O DTK (Deception Toolkit) consiste em uma série de *scripts* que emulam diversas vulnerabilidades conhecidas. Por exemplo, uma vulnerabilidade no programa *Sendmail* que permite que o atacante capture um arquivo de senhas falso. É um *software* muito utilizado na prevenção de ataques, já comentada, pois o atacante perde um tempo precioso apenas para descobrir se os sistemas vulneráveis são reais ou não. Porém, o DTK é limitado para outros objetivos, já que apenas vulnerabilidades conhecidas são exploradas. O *Honeyperl* é outro *software*, desenvolvido pelo grupo *Honeypot-BR*. Ele integra diversos serviços *fakes*, ou falsos, como Squid, Telnet, HTTP, FTP e SNMP com o objetivo de ludibriar o atacante e funciona em plataformas Linux e Windows. *Specter* , *KFSensor*, *LaBrea Tarpit*, *Cybercop Sting* e *Recourse Mantrap* são outros exemplos de *softwares* de baixa interação.

6.2.2 HONEYPOTS DE ALTA INTERAÇÃO

São soluções mais completas e complexas, que envolvem sistemas e aplicações reais. Nada é emulado. Possuem a vantagem da captura de maior quantidade de informação, o que possibilita o aprendizado completo do comportamento dos *black hats*, até mesmo o que não era esperado. Captura-se toda a atividade, sem fazer qualquer previsão de um possível comportamento que o atacante teria no futuro. Descobre-se novos *rootkits*, sessões de IRC. O grande problema, entretanto, é o aumento do risco dos invasores utilizarem esses sistemas reais para atacarem outros sistemas que não sejam *honeypots*. Os sistemas de alta interação, em geral, são mais difíceis de construir e manter.

Existem sistemas comerciais, como o *Symantec Decoy Server*, que implementam *honeypots* de alta interação. Todavia, os sistemas mais populares desse tipo são as *Honeynets*. Não é uma solução pronta em *software* que se instala em um computador. É uma arquitetura

de rede construída apenas para ser comprometida, com um rigoroso controle e processo de captura de toda a atividade. Os invasores encontram, atacam e conseguem entrar no sistema sem saber que se trata de uma armadilha. O sistema captura o tráfego de entrada e ao mesmo tempo controla o tráfego de saída. Uma segunda fase consiste em uma análise minuciosa de todo o tráfego.

6.3 HONEYNETS

Uma *honeynet* é um tipo de *honeypot*, no sentido mais amplo. Simultaneamente, é composta por diversos *honeypots*. Eles podem ser qualquer tipo de sistema, dependendo do objetivo da rede. Um *Solaris*, um *Windows 2003 Server*, um *FreeBSD*, um roteador *Cisco.* Podem também hospedar os mais diversos tipos de serviço. A flexibilidade fornece o verdadeiro poder das *honeynets*. Conceitualmente elas são muito simples. Qualquer tráfego de entrada é considerado atividade maliciosa e qualquer tráfego de saída é iniciada pelo invasor. Isso facilita imensamente a análise dos *logs* quando comparado a outros elementos de segurança como IDS ou *firewall*, que capturam tráfego de todo tipo. É importante salientar que o modo de construção de uma *honeynet* pode determinar o perfil de ataque que ela receberá. Sistemas extremamente vulneráveis são facilmente comprometidos por *script kiddies, worms, bootnets,* ou qualquer outro tipo de ataque automático. Raramente serão vítimas de ataques avançados. Redes sem vulnerabilidades aparentes são eficientes em detectar ataques novos e sofisticados.

A fase de construção de uma *honeynet* é a mais importante. Ela deve seguir alguns requisitos de controle e de captura de dados que são essenciais para o sucesso do projeto. O controle dos dados é sempre prioritário em relação à captura. Isso porque o controle é destinado a minimizar os riscos da *honeynet*. A captura é destinada a registrar cada atividade do atacante sem que ele saiba.

6.3.1 CONTROLE DE DADOS

O controle existe principalmente para mitigar o risco que se corre de danificar o sistema de terceiros. Os invasores precisam de um bom grau de liberdade para agirem, até porque quanto mais liberdade tiverem, mais se aprenderá sobre eles. No entanto, quanto mais

liberdade tiverem, maior o risco que se corre de conseguirem vencer o sistema de controle. O grande desafio da construção de uma *honeynet* é saber dosar esse grau.

Outro desafio é ter que esconder o controle do atacante e garantir que ele seja efetivo. Uma prática essencial para evitar que se tenha apenas um ponto de falha é realizar o controle em camadas. Quanto mais elementos de controle houverem tanto melhor para prevenir ataques novos e não conhecidos. Contador de conexões externas, *gateway* de prevenção de intrusão e restrição de banda são alguns dos mecanismos possíveis. Porém, é necessário ter em mente que qualquer que seja a proteção, ela serve apenas para minimizar o risco, não para eliminá-lo.

6.3.2 CAPTURA DE DADOS

É o processo de monitorar e registrar todas as atividades realizadas pelos invasores sem que eles percebam. É a coleta da matéria-prima de qualquer análise posterior com o objetivo de aprender as técnicas, ferramentas e motivos dos *black hats.* O grande desafio é capturar o máximo de infomações possíveis sem que o atacante saiba. Novamente a arquitetura em camadas é a melhor solução. Quanto mais mecanismos houverem de captura (*logs* de *firewall, sniffers, logs* de IDS, *logs* do sistema, etc), mais aprendizado eles trarão.

Para evitar que os atacantes detectem a armadilha, uma solução é mexer o menos possível nos *honeypots,* ou seja, armazenar as informações da captura em um outro local, longe do invasor. Se este tiver acesso aos dados, não apenas detecta, mas modifica ou exclui.

6.3.3 RISCOS

O risco em uma *honeynet* é o preço que se paga por autorizar aos atacantes o acesso privilegiado a sistemas e aplicações. Cada organização detentora de uma *honeynet* deve medir por si própria o risco que pretende correr. Não existe um consenso quanto às questões legais de uma *honeynet* e cada organização é responsável pela sua. Os principais riscos percorrem quatro áreas: dano, detecção, desabilitação e violação.

O risco de dano ocorre quando uma *honeynet* é utilizada para atacar ou danificar outros sistemas reais, que não são *honeynets*. Qualquer que seja a proteção do sistema, ela poderá sofrer um ataque desconhecido que permita que o atacante provoque sérios danos a

terceiros. A melhor forma de proteger-se é realizar uma proteção com muitas camadas de controle de dados. Apesar disso, ter em mente que o sistema não está totalmente protegido.

Quando o atacante detecta que está numa *honeynet*, ele passa a ignorá-la ou procura apagar os registros de sua invasão, eliminando toda a capacidade de captura e, portanto, diminuindo o valor da *honeynet*. O atacante pode fazer pior, registrar informações falsas e confundir a análise posterior dos *logs*.

Além de detectarem a identidade de uma *honeynet*, atacantes podem querer desabilitar seus mecanismos de controle e captura sem que o administrador saiba. Por exemplo, um atacante ao conseguir acesso a uma *honeynet* desabilita seu mecanismo de captura, mas simula algumas informações para que o administrador pense que o mecanismo ainda está funcionando perfeitamente. A maneira de reduzir esse risco é novamente através de múltiplas camadas de controle e captura de dados.

O risco de violação refere-se ao sentido legal. Atacantes podem realizar atividades criminosas sem fazer nenhum tipo de ataque a terceiros. Um exemplo é utilizar a *honeynet* como servidor de materiais ilegais, como cópias piratas de filme, música, pornografia infantil, número de cartões de crédito roubados, etc.

Em todos os casos, a maneira mais eficiente de diminuir os riscos, além de se construir uma arquitetura em camadas, é a constante monitoração humana e customização do sistema. É aconselhável que se tenha profissionais treinados capazes de fazer uma análise da *honeynet* em tempo real. Para um ataque ser identificado, é necessário monitorar e analisar todos os dados capturados. Ter um profissional de segurança nessa tarefa é essencial para evitar que ataques novos e desconhecidos provoquem os riscos citados. As técnicas automáticas de análise neste caso não são eficientes devido à complexidade das informações e a possibilidade de novos ataques. Em último caso, a *honeynet* permite o recurso de ser desconectada da rede para eliminar os riscos. A customização do sistema para cada *honeynet* também é essencial para reduzir os riscos. Com os sistemas e aplicativos mal configurados, ou em configuração padrão, o atacante facilmente descobrirá o sistema, pois documentação sobre o assunto não falta na Internet.

A mensagem principal a respeito dos riscos é que não importa quão eficiente seja o controle e captura de dados em uma *honeynet*, o risco nunca é eliminado (a não ser que a *honeynet* esteja fora do ar), ele apenas pode ser mitigado.

6.3.4 NA PRÁTICA

O exemplo prático explicitado neste tópico é baseado na *honeynet GenII*, ou segunda geração, do *Honeynet Project*. Serve para ilustrar os elementos que podem ser utilizados em uma *honeynet* para garantir os pilares anteriormente citados de controle, captura e redução dos riscos.

6.3.4.1 Arquitetura

A vantagem de uma *honeynet* sobre os demais *honeypots* é a sua flexibilidade. Um *software*, por mais complexo que seja é limitado às possibilidades imaginadas por seus programadores. Uma arquitetura de rede completa provê interação real com o atacante e aumenta infinitamente as possibilidades de captura e aprendizado de novas técnicas de ataque. Além disso, cada organização tem a liberdade de escolher sua própria arquitetura, de acordo com seu objetivo e planejamento.

Apesar de cada *honeynet* poder ser construída de maneira diferente, algumas semelhanças de projeto são inevitáveis para garantir o controle, captura e análise de dados de maneira mais fácil e centralizada. A principal delas é a presença de um *gateway* separando a *honeynet* do resto do mundo. Todo o tráfego entrante ou sainte da *honeynet* obrigatoriamente tem que passar pelo *gateway*. Ele age como um muro de proteção permitindo que o atacante invada e, ao mesmo tempo, controlando sua saída. Este muro torna-se o centro de controle e comando de toda a rede.

No projeto *GenII*, um *linux* 2.4.x em arquitetura x86 foi utilizado para funcionar como um *gateway* camada 2. Isso quer dizer que a máquina não possui endereço IP e, portanto, é transparente ao atacante e muito mais difícil de ser detectada. Como pode ser observado na figura 6.1, o muro, chamado de *honeywall* no projeto foi utilizado para separar a rede de produção da rede armadilha e os endereços IPs de ambas as redes são endereços internos. O objetivo dessa arquitetura foi o da captura de ameaças tanto externas quanto internas. Observa-se também que a *honeywall* possui 3 interfaces de rede: a eth0 interage com a rede de produção, a eth1 interage com a *honeynet* e a eth2 é a única que possui endereço IP e tem o propósito de prover uma administração remota ao *gateway* na medida que quaisquer dados capturados pela *honeynet* são movidos através dessa interface para uma rede administrativa

segura sem nenhum contato com a Internet.

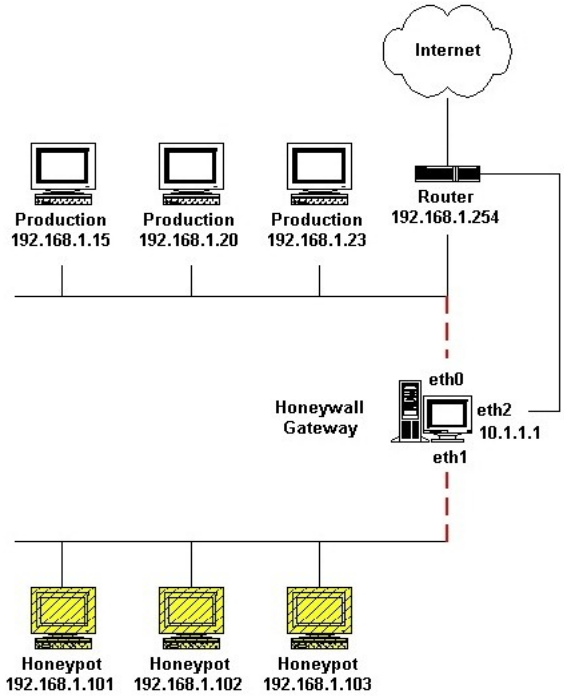


Figura 6.1 – *Honeynet GenII* do *Honeynet Project*

O *gateway* centralizado camada 2 possui grandes vantagens. A primeira é a dificuldade de detecção por parte do atacante. Não há *hops* de roteamento e não há decremento no campo *TTL* do cabeçalho IP. Além disso, simplifica muito a tarefa de controle e captura dos dados, pois implementa as duas funcionalidades de maneira centralizada e segura. Nos próximos tópicos, uma breve pincelada nos *softwares* utilizados para controle e captura dos dados.

6.3.4.2 Controle

O controle de dados da *honeynet* é realizado de maneira centralizada no *gateway*. Para segurança do *gateway* e para o próprio controle de dados, é necessário que o *firewall IPTables* do *linux* esteja habilitado no modo *bridge*(camada 2). Existe um *path* para o *kernel* do *linux* chamado *ebtables* que implementa essa funcionalidade.

Já foi discutido que o grande desafio do controle de dados é saber equilibrar a liberdade do atacante e o controle de tráfego de saída com o objetivo de mitigar o risco principalmente de ataque a terceiros. Quanto mais liberdade ele tiver, mais se aprenderá sobre ele. Entretanto é necessário contê-lo sem que ele perceba isso. Duas tecnologias foram

implementadas para cumprir estes objetivos: um contador de conexões e NIPS (*Network Intrusion Prevention System*).

O contador de conexões realiza a contagem e bloqueio de um determinado número de conexões iniciadas pela *honeypot*. Quando um certo limite é atingido, qualquer outra conexão é bloqueada. O objetivo é evitar ataques massissos de *scanner*, de DoS ou *worms* que requerem muitas conexões de saída. O *IPTables* é utilizado para contar e bloquear essas conexões através do *script rc.firewall.* Ele utiliza um limite para cada tráfego: TCP, UDP, ICMP e "OUTROS". Além disso, utiliza uma escala de tempo afim de zerar o contador quando a escala for atingida. Assim, o atacante inicia a conexão por objetivos diversos (*download* de *toolkits*, *IRC chats*, *emails*, *scanners*, etc) e quando o limite é alcançado, aquela *honeypot* é bloqueada. Passado o intervalo de tempo determinado, ela é desbloqueada e o contador zerado. Os limites e o intervalo são escolhidos de acordo com a consciência de quem implementar.

O NIPS é um mecanismo que bloqueia ou desabilita ataques conhecidos. Isto é feito inspecionando cada pacote que passa pelo *gateway*. Ele funciona como um IDS, mas ao invés de apenas mandar alertas ao administrador, ele modifica ou descarta o pacote, reduzindo o risco de sucesso de ataques conhecidos a terceiros. Quer dizer, o número limite de conexões que foi permitido no *firewall*, se se tratar de um ataque conhecido, poderá ser neutralizado pelo NIPS. O *software* utilizado para funcionar como NIPS chama-se *Snort_inline*. É uma versão modificada do IDS *Snort* e utiliza o mesmo mecanismo de reconhecimento dos ataques. Os pacotes precisam ser roteados para o *Snort_inline* analisar. Isto é feito novamente pelo *IPTables* no modo *ip_queue*, através do *script rc.firewall*. Não há como não contar os pacotes que forem modificados pelo *Snort_inline*. Isso porque eles passam primeiro pela *interface* de rede e são contados. Só depois são analisados. Nem todas as regras do *snort* precisam ser utilizadas pelo *snort_inline*. O administrador deve escolher que tipos de ataque ele considera passível de bloqueio ou modificação. A ferramenta *snortconfig* pode ser utilizada para converter regras atualizadas do *snort* para o *snort_inline*.

6.3.4.3 Captura

O propósito da captura de dados é logar das mais diversas maneiras possíveis todos os passos do atacante. Sem essa captura, a *honeynet* não teria nenhum valor. A coleta de dados é

realizada em camadas. O *Honeynet Project* definiu três camadas principais para a captura: *logs* de *firewall*, tráfego de rede e atividade do sistema.

Os *logs* de *firewall* são facilmente coletados. O *script rc.firewall* já foi construído para registrar todas as atividades de entrada e saída em */var/log/messages*. É uma informação crítica e é a primeira indicação a respeito da atividade de um atacante.

O tráfego de rede, por sua vez, pode ser capturado através do *snort* atuando em modo *sniffer*, ou mesmo um *sniffer* comum, como o *tcpdump*. Ambos capturam no mesmo formato(*tcpdump*) qualquer pacote incluindo o *payload* que passa pela rede. É importante observar que a *interface* a ser capturada deve ser a interna. Se for configurada a externa, corre-se o risco de capturar tráfego de outras máquinas, poluindo os registros.

O terceiro elemento de captura é o mais sofisticado de todos e age capturando as atividades do atacante na *honeypot* e transferindo para um local central, o *gateway*. O *sebek* é um programa cliente que fica instalado em cada *honeypot* enviando *logs* de todo e qualquer tipo de comando ou download que o invasor venha a fazer (inclusive scp e ssh) para o *gateway* via porta UDP. Para realizar o envio, ele utiliza o endereço MAC do *gateway*. A grande vantagem desse mecanismo é que o atacante nem com um *sniffer* consegue capturar os pacotes de envio. Isso porque o *sebek* é um módulo carregável do kernel que grava todos os dados no cliente acessando a chamada de sistema *read()*, ou seja, antes mesmo de ser encriptado pelo ssh. O módulo fica escondido do invasor e não pode ser desativado. O tráfego UDP é gerado sem usar a pilha TCP/IP e é transmitido diretamente à interface de rede. Desta maneira, o invasor não é capaz de capturar os dados através de um *sniffer*.

6.3.4.4 Alerta

Um último elemento importante na construção de uma *honeynet* é o sistema de alerta. Considerando que não haverá pessoas analisando o tráfego a todo momento, é essencial que o administrador tenha um mecanismo de alerta para notificar as invasões. O *software swatch* é uma opção para uma monitoração automática. Ele procura em arquivos de *log* por padrões descritos em seu arquivo de configuração. Quando um padrão é encontrado, ele pode enviar alertas via e-mail, alarmes de sistema, chamadas telefônicas ou rodar comandos e programas. É importante que o administrador acompanhe de perto sua *honeynet* para reduzir mais ainda os riscos.

6.4 HONEYPOT FARMS E HONEYPOTS DINÂMICOS

Em 2003, Lance Spitzner escreveu para o portal *Security Focus* dois artigos que fogem um pouco do escopo do presente projeto. A escolha em comentá-los neste tópico foi devido à importante discussão que ele levanta e que é essencial para o entendimento de alguns problemas enfrentados por implementações reais de *honeypots* e algumas possíveis soluções para estes problemas.

6.4.1 HONEYPOT FARMS

Uma pequena organização necessita apenas de um ou dois *honeypots* em sua rede local, enquanto que grandes organizações se tivessem em cada rede local um *honeypot* deixariam o trabalho de manutenção e gerenciamento praticamente impossível. O conceito de *honeypot farms* é aplicável principalmente para grandes organizações distribuídas geograficamente em diversas redes. Em última instância, elas podem ser consideradas como grandes *honeynets* com a adição de um elemento redirecionador de tráfego, como será visto adiante.

Como já foi discutido antes, o grande problema de um *honeypot* é a sua visão limitada de tráfego. Ele consegue monitorar apenas o tráfego da rede em que foi construído. Apesar disso, é de grande valor para as organizações, à medida que consegue trabalhar com tráfego criptografado, IPV6, reduz falso positivos e captura ataques novos e desconhecidos. Grandes organizações possuem diversas redes espalhadas pelo mundo e têm interesse em monitorar o máximo possível à procura de ataques. Entretanto, quanto mais *honeypots* forem construídas, mais recursos humanos são requeridos para administrá-las. No caso de *honeynets*, essa dificuldade torna-se ainda maior. Uma vez construída, uma *honeynet* requer um time próprio dedicado para sua manutenção e análise. É preciso manter a solução segura, reduzindo o risco de ataque a terceiros. Muita informação é capturada e precisa ser cuidadosamente analisada. Estima-se que para cada trinta minutos de ataque são gastas trinta horas em uma análise forense completa. Pode-se imaginar este trabalho multiplicado por dezenas, centenas ou milhares de redes.

Uma possível solução para esta dificuldade seria a construção da chamada *honeypot*

farm, uma única e consolidada rede de *honeypots* utilizando recursos dedicados de segurança de maneira centralizada. Essa "fazenda" de *honeypots* pode fazer parte, por exemplo, de uma central de operações de segurança de uma organização e contar com recursos humanos e materiais dedicados para a solução, tornando-a muito mais simples de manter, gerenciar e analisar. Um redicirecionador pode ser construído para transportar o atacante de uma rede real em qualquer parte da organização para a *honeypot farm* sem que ele perceba. A "fazenda" pode ser simples, implementando muitos *honeypots* de baixa interação, ou complexa, implementando largas *honeynets* com centenas de sistemas e aplicações reais esperando pelo ataque. Os redirecionadores podem ser vistos como "caixas-pretas" que simplesmente são pré-configuradas e plugadas em cada rede, monitorando (IPs não utilizados e/ou comportamento malicioso) e redirecionando os ataques para a "fazenda". Além das vantagens de administração e atualização, o risco também é bastante reduzido, uma vez que há uma equipe especializada em manter a *honeypot farm* em um único ponto de controle.

Pelo fato de ser um conceito extremamente novo e poderoso, a *honeypot farm* não possui uma implementação completa, ou auto-suficiente. Os redirecionadores são extremamente difíceis de construir. Como saber quais atividades transportar, para quais *honeypots* e como transportar sem que o atacante descubra? Seria necessário monitorar todos os IPs não utilizados, ou apenas alguns mais importantes? Como assegurar que o que o atacante procura é similar ao que a *honeypot* implementa? São muitas questões ainda a serem consideradas na construção de uma solução. Enquanto isso, alguns *softwares* já existentes podem auxiliar na implementação de algo próximo. O *honeyd*, por exemplo, implementa a monitoração do espaço de IPs não utilizados e possui uma função de redirecionar uma conexão interceptada para um outro endereço IP. Neste caso, poderia-se combinar *honeypots* de baixa e de alta interação. O *honeyd* fazendo o papel de redirecionador e a *honeynet* esperando para ser invadida. Como exemplo de *software* comercial que implementa o conceito de *honeypot farm*, existe o *NetBait*.

6.4.2 HONEYPOTS DINÂMICOS

Segundo Lance Spitzner, a solução ideal e o futuro dos *honeypots* é o chamado *honeypot* dinâmico. Uma solução *plug and play* que funcionaria como um espelho da rede de produção. Automaticamente, a solução determinaria quantos e quais *honeypots* seriam

desenvolvidos. Além disso, os *honeypots* se adaptariam à rede de produção. Quer dizer, uma vez adicionado um servidor Linux, ou um roteador Cisco na rede de produção, automaticamente o *honeypot* dinâmico adicionaria este sistema em sua solução. Uma vez retirado um serviço obsoleto, como o telnet, o *honeypot* o retiraria também da solução. O objetivo é ter um elemento, provavelmente um *appliance plug and play*, que consiga aprender o ambiente de produção e desenvolver o número correto de *honeypots* com suas devidas configurações. Esses *honeypots* sendo totalmente adaptáveis às mudanças ocorridas na produção.

O grande argumento a favor dos *honeypots* dinâmicos é a enorme dificuldade de configuração de um *honeypot* comum, principalmente uma *honeynet*. A necessidade de ter um acompanhamento e manutenção de perto por especialistas, a dificuldade de acompanhar mudanças no ambiente de produção e mudanças na própria tecnologia das redes acabam por transformar o *honeypot* em um investimento caro. Para uma maior efetividade, é necessário que os serviços e sistemas operacionais de um *honeypot* estejam rodando o mais próximo possível da rede de produção. Contudo, a produção está sempre em mutação por necessidade da própria tecnologia. Sistemas estão sempre sendo adicionados ou desativados, o mesmo acontece com os serviços. Por esses motivos, um *honeypot* dinâmico é considerado como o *honeypot* perfeito. Apesar de parecer utópico, existem tecnologias hoje que, trabalhando em conjunto, podem ser utilizadas para construir um primeiro *honeypot* dinâmico.

A primeira parte de um *honeypot* dinâmico consiste em “aprender” quais sistemas a organização utiliza na produção e como eles estão sendo utilizados. É um mecanismo que permite que o *honeypot* mapeie e atue como a rede real. Uma possibilidade é utilizar uma ferramenta de *scanner*, como o *nmap*. Entretanto, ela traz alguns problemas, como o aumento de tráfego na banda, a possibilidade de prejudicar alguns serviços, a possibilidade de deixar de capturar algum sistema (atrás de um firewall, por exemplo) e o fato de as atualizações não ocorrerem em tempo real e ser necessário um *scanner* ativo para saber a situação de cada sistema no momento. A grande opção parece ser o *passive fingerprinting*, ou seja: ferramentas como o *p0f* ,ou semelhantes. O objetivo é capturar o tráfego passivamente, analisá-lo e determinar a identidade do sistema. Ele mantém uma base de assinaturas conhecidas para cada sistema operacional, captura os pacotes da rede e analisa comparando com a base. Desta maneira, sabe quais são os sistemas ativos no momento (em tempo real). Ao contrário do *scanner* ativo, essa não é uma técnica intrusiva, não utiliza “ataque” de pacotes à maquina

alheia e tem menos chance de prejudicar algum sistema ou serviço. Pode identificar máquinas por trás de um *firewall* através de mapeamento de endereço MAC. Por fim, é útil para manter um *honeypot* por longo tempo, à medida que consegue perceber uma mudança na rede. A grande desvantagem é a incapacidade de trabalhar por redes roteadas, ou seja, só serve para redes locais. A solução é desenvolver diversos *honeypots* espalhados pela organização.

Outra parte consiste em desenvolver os *honeypots* de acordo com o mapeamento realizado pelo *passive fingerprinting:* diversos sistemas e serviços diferentes e diversos endereços IPs não utilizados. Novamente, o *honeyd* pode ser utilizado para resolver o problema. Ele tem a capacidade de construir *honeypots* virtuais e emular cerca de 500 sistemas operacionais, além de monitorar os IPs não utilizados. Evidentemente precisaria de uma customização para realizar as função de construir automaticamente *honeypots* de acordo com o mapeamento. A grande vantagem é que ambos os projetos (*p0f* e *honeyd*) são de código aberto e permitem customização a gosto do freguês.

6.5 CONSIDERAÇÕES FINAIS

Durante o capítulo, o leitor pôde entender o que é uma *honeynet,* quais são as suas aplicações e diferenças em relação à outros *honeypots*. Pôde acompanhar os projetos em escala mundial para análise e compartilhamento de experiências, em especial o *Honeynet Project*. Além disso, compreendeu os elementos principais na construção de uma *honeynet,* seus riscos e problemas práticos. Por último, enxergou algumas dificuldades adicionais na construção em larga escala e algumas possíveis soluções.

O grande objetivo foi preparar teoricamente o leitor para o próximo capítulo, onde uma *honeynet* nesses moldes será construída e utilizada em conjunto com os outros conceitos apresentados no decorrer do trabalho, a fim de constituir um ambiente de treinamento para redes neurais artificiais utilizadas para detecção de intrusão.

7 HONEYNET, UM AMBIENTE PARA TREINAMENTO DE REDES NEURAIS ARTIFICIAIS PARA DETECÇÃO DE INTRUSÃO

O objetivo dos capítulos anteriores foi o de fomentar uma base teórica para a construção de uma proposta real de detecção de intrusão através de algoritmos não lineares, mais especificamente redes neurais artificiais. Este capítulo apresentará uma proposta amadurecida pelos conceitos vistos, bem como seus resultados experimentais.

7.1 A PROPOSTA

O capítulo 3 discutiu a importância dos sistemas de segurança da informação nos dias de hoje e no mundo da informática, bem como as grandes ameaças de ataques as quais os sistemas e serviços disponíveis na Internet estão sujeitos. Um dos muitos mecanismos de proteção discutidos, foi o mecanismo de detecção de intrusão, realizado por sistemas chamados de IDS (*Intrusion Detection Systems*). Estes sistemas são classificados de acordo com o seu tipo de monitoração em IDS de *host* (ou HIDS) e IDS de rede (NIDS). Também são classificados de acordo com seu tipo de detecção: baseados em assinatura ou baseados em anomalia.

O capítulo 4 discutiu redes neurais artificiais, um algoritmo não linear que procura, através de um modelo matemático, emular as funções neurais do corpo humano. Na prática, o objetivo é que os neurônios artificiais consigam “aprender” determinado problema e ter a capacidade, assim como os seres humanos, de generalizar as decisões baseados em uma experiência adquirida no treinamento. A etapa mais importante na contrução de uma rede neural artificial é justamente o treinamento. Ele pode ser dividido em dois grandes grupos, com diferentes paradigmas. O treinamento supervisionado necessita de um “professor”, ou seja, uma entidade externa que classifique as amostras de treinamento dizendo a qual classe de saída ela pertence. O treinamento não supervisionado não classifica as amostras de entrada e deixa para a rede neural construir sua própria representação.

A capacidade das redes neurais de generalização e reconhecimento de padrões já foi utilizada na área de detecção de intrusão em diversas pesquisas acadêmicas, como foi mostrado no capítulo 5. A idéia é superar a dificuldade dos sistemas de detecção de intrusão atuais (que são baseados em assinatura) de detectar ataques novos ou desconhecidos. Além disso, otimizar a detecção, à medida que um sistema baseado em redes neurais não precisaria comparar um pacote de rede, por exemplo, com toda a sua base antes de decidir se trata-se de uma intrusão. Uma rede neural, caso tenha uma rotina de treinamento eficiente, poderá aproveitar sua capacidade de reconhecimento de padrões para detectar ataques conhecidos. E sua capacidade de generalização para detectar ataques novos e desconhecidos.

No capítulo 6, um elemento de segurança extremamente novo e promissor foi apresentado. Uma *honeynet* é uma rede real construída unicamente para ser invadida por atacantes na Internet. O objetivo é o estudo das técnicas e ferramentas utilizadas para a invasão. Desta maneira, ataques novos e desconhecidos serão descobertos e a proteção contra eles poderá ser construída. Uma grande vantagem da *honeynet* é que todo o tráfego capturado tem origem ilícita ou maliciosa. Quer dizer, todo o tráfego é ataque e é de igual importância.

A proposta prática deste capítulo é utilizar a capacidade de uma *honeynet* de capturar ataques, conhecidos ou não, e treinar uma rede neural artificial supervisionada para que ela possa reconhecer os padrões de ataque na rede quando em produção (funcionando como um NIDS). Os desdobramentos deste objetivo são diversos.

Primeiro, como já foi discutido, existe na implementação de uma *honeynet* a dificuldade enorme de análise do tráfego capturado. Apesar de todos os pacotes serem considerados como ataque, é necessária uma análise humana afim de filtrar esses pacotes. Isso porque o treinamento de um algoritmo *backpropagation* não pode ser construído simplesmente inserindo todos os pacotes capturados e considerando-os ataque. É necessária uma otimização dos pares de treinamento. A filosofia do algoritmo não é quanto mais amostras melhor e sim a de escolher a quantidade certa e mais significativa tanto quanto for possível.

Outra dificuldade é a diferença brutal entre os padrões de cada ataque diferente. Um ataque de *port scannig* é constituído de diversos pacotes enviados para diversas portas de um mesmo *host*. Um ataque de DDoS é constituído de milhões de pacotes idênticos vindos de diversos *hosts* distribuídos. Já um ataque de *exploit* é constituído de poucos pacotes, contendo código malicioso. Seria um problema juntar todos os tipos no mesmo treinamento e construir

uma rede neural única que reconheça qualquer ataque. As especificidades de cada ataque têm que ser preservadas. Do contrário, a rede neural não saberá identificar que os poucos pacotes de um ataque de *exploit* valem tanto quanto os milhões de pacotes de um ataque de DDoS.

Não há como escapar de uma análise humana feita por um especialista de segurança para diferenciar o tráfego capturado pela *honeynet*. Para isso existem diversas ferramentas de análise de *logs*, de tráfego, de correlação, etc. Nos projetos de *honeynet* espalhados pelo mundo e nos portais de segurança da informação essas ferramentas são recomendadas e disponibilizadas. Entretanto, no capítulo 5, uma alternativa foi apresentada através do trabalho de Luc Girardin: uma rede neural artificial não supervisionada, baseada em aprendizagem competitiva e criada pelo finlandês Tuevo Kohonen. A rede SOM (Self Organizing Maps), ou simplesmente rede de Kohonen, pode ser utilizada como ferramenta auxiliar no processo de análise do tráfego da *honeynet*. Ela cria um mapa de visualização do tráfego que, dependendo da experiência do especialista e dos parâmetros de construção do mapa, consegue identificar que tipo de ataque foi predominante naquele tráfego.

Uma vez construída a *honeynet* e capturadas as atividades ilícitas através dos pacotes de rede, os diferentes tipos de ataque podem ser identificados pela visualização dos mapas de Kohonen, análise dos pacotes de rede e alertas do *snort*.

A partir do momento em que são identificados, a proposta é que cada tipo de ataque seja treinado em uma rede neural diferente de forma supervisionada. Desta maneira, haveria a construção de redes neurais especialistas para identificar cada tipo de ataque remoto. Um sistema de detecção de intrusão de rede pode, a partir daí, ser construído utilizando a análise de cada uma das redes neurais para cada pacote. Quando qualquer uma delas identificar um ataque, o IDS tomará suas providências bloqueando o ataque ou alertando o administrador.

Para a maturidade e o sucesso da solução, a experiência dos especialistas será fundamental. A tarefa de analisar o mapa das atividades de rede e de escolher as amostras mais significativas para o treinamento só depende da habilidade e experiência do analista. Além disso, a *honeynet* deverá estar o mais próximo possível dos sistemas e serviços existentes na produção, para que os ataques sofridos e treinados sejam, no mínimo, parecidos com os de produção.

A seguir, uma modesta prova de conceito será testada para comprovar a viabilidade da solução. Todavia, ainda há muito a ser feito e testado para que se torne realidade. A pretensão da pesquisa é apenas no sentido de contribuir de maneira acadêmica para a evolução dos

sistemas de detecção baseados em redes neurais artificiais.

7.2 A PROVA DE CONCEITO

Neste tópico, será descrita passo a passo a construção da prova de conceito para a solução proposta. Foi desenvolvida uma *honeynet* real, observando obviamente todos os cuidados descritos no capítulo 6. Na verdade, apesar de possuir todos os mecanismos de captura e controle de dados, os aplicativos efetivamente utilizados, depois que a *honeynet* foi ao ar, foram o *tcpdump:* utilzado para captura dos pacotes de rede e o *snort:* utilzado para análise dos alertas de ataque.

Após um período no ar, a *honeynet* foi desligada e passou-se à fase de análise do tráfego. Para isso, foi desenvolvido um aplicativo em linguagem C (ferramenta *dsni*), utilizando a biblioteca *libpcap*, para captura e análise de pacotes, além da formatação e normalização dos campos dos protocolos para servirem de entrada das redes neurais. Essa ferramenta foi chamada de *dsni*.

A rede de Kohonen foi construída com base no *software* livre SOM_PAK, escrito em linguagem C e que permite o treinamento, construção e visualização dos mapas correspondentes a cada período de tráfego capturado.

A rede MLP foi implementada pelo *software EasyNN*, em versão *trial*, que permite uma facilidade de configuração dos parâmetros de treinamento: número de camadas, de neurônios, taxa de aprendizagem, critério de parada, manipulação dos dados. Além da visualização do treinamento com gráficos em tempo real

O funcionamento da solução é mais ou menos desta maneira:

- Ataques são recebidos na *honeynet* durante um determinado período de tempo e são registrados nos mecanismos de captura (*logs* de IDS, firewall, sistema, *sniffer*);
- Após determinado período, a *honeynet* é desligada e passa-se a fase de identificação dos ataques;
- Para identificação são utilizadas análises de todos os mecanismos de captura, com destaque para os *logs* de IDS(*snort*) e *sniffer* (*tcpdump);*
- Como auxílio na identificação de ataques, são utilizados os mapas de Kohonen, construídos através do treinamento de uma rede não supervisionada, utilizando parâmetros de pacotes IP, TCP e UDP. Estes parâmetros são coletados e normalizados através da

ferramenta *dsni;*

- Uma vez identificado um ataque, por exemplo, *port scanning*, ele é apresentado à rede neural supervisionada MLP, com algoritmo *backpropagation*. Os parâmetros de treinamento são os mesmos campos utilizados na rede de Kohonen. Além disso, é apresentado ao mesmo *software* um tráfego normal, capturado da rede de produção. Desta maneira, é realizado o treinamento e a rede pode reconhecer outros pacotes como normal ou ataque. Para uma maior garantia que o tráfego classificado como normal no treinamento seja realmente normal, o *software snort* foi configurado e os pacotes capturados como normais na produção comparados com seus *logs*. Caso algum fosse classificado pelo s*nort* como ataque, ele seria retirado da amostragem de pacotes normais para o treinamento.

7.2.1 A HONEYNET

7.2.1.1 Arquitetura

A *honeynet* montada dispõe de 4 servidores e um HUB. Eles foram configurados de acordo com a arquitetura abaixo:

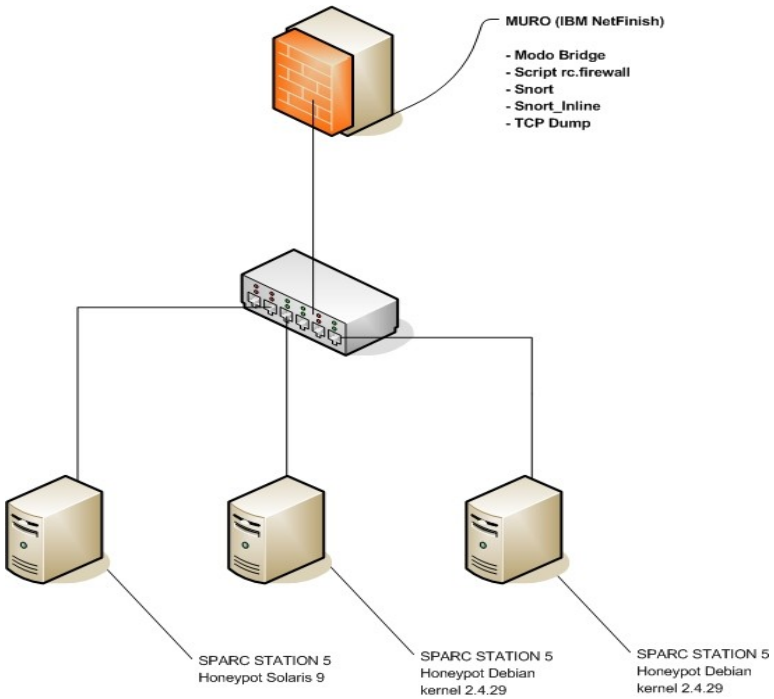


Figura 7.1 Arquitetura da *honeynet*

O primeiro servidor (IBM Netfinish) possui mais capacidade de armazenamento e está configurado em modo *bridge*(camada 2), sem endereço IP, apenas "analisando" o tráfego de entrada e "controlando" o tráfego de saída. Nele estão configurados os serviços: Snort(mysql, Apache SSL e ACID), Snort_Inline e o script rc.firewall. Os demais servidores (Sun Sparc Station 5) estão funcionando apenas como *honeypots*. Eles foram instalados e a configuração *default* foi mantida. Cada um possui um endereço IP válido na Internet.

7.2.1.2 Controle de Dados

O controle de dados na *honeynet* é feito pelo elemento de entrada na rede, ou seja, o *gateway* camada dois (IBM Netfinity em modo *bridge*), chamado no projeto de "muro". O muro está configurado com o sistema operacional linux, distribuição *Debian*, kernel 2.4-29. O kernel foi customizado para trabalhar em camada 2, sendo mais difícil para o atacante detectar sua presença, ou alterar seus *logs*.

Ao mesmo tempo, o script *rc.firewall* (do *honeynet project*) é utilizado para configurar o *iptables* no intuito de prevenir ataques de dentro da *honeynet* para o mundo externo realizados pelos invasores. Uma das principais funcionalidades deste *script* é a implementação da limitação de conexões. O objetivo é conter o número de conexões que podem ser estabelecidas para fora da *honeynet*. A partir de um certo número conexões, o *iptables* bloqueia qualquer tentativa de mais conexões, evitando assim que, por exemplo, as máquinas se tornem "zumbis" dos atacantes para realizarem ataque de DoS ou DDoS. Assim, dentro do script *rc.firewall* é definido a taxa de conexões UDP, TCP, ICMP, ou OUTROS e a escala de tempo (horária, diária, etc). O invasor poderá iniciar uma conexão por diversos motivos (fazer download de toolkits, iniciar chats IRC, enviar e-mails, etc). A cada conexão iniciada, o *rc.firewall* irá incrementar seu contador. Assim que o número definido no *rc.firewall* for alcançado, o *iptables* irá bloquear as demais tentativas. Assim que a escala de tempo definida for alcançada, o *iptables* irá reiniciar, permitindo novamente o número de conexões definidas.

Outro recurso configurado no muro para controle dos dados é o chamado NIPS(Network Intrusion Prevention System), implementado através do s*nort_inline*. Este software é uma versão modificada do *snort*, que age não apenas como um detector de intrusão, mas como prevenção de intrusão. Ele é capaz de rejeitar ou modificar pacotes que

ele identifique como um ataque conhecido (de acordo com as regras do *snort*). Assim, mesmo que o ataque esteja dentro dos limites de conexão definidos pelo script *rc.firewall*, ele poderá ser bloqueado pelo s*nort_inline*. Para que ele funcione, o *iptables* precisa estar trabalhando em modo QUEUE, pois é ele que irá entregar os pacotes para análise do *snort_inline* e depois continuar roteando os mesmos. Para transformar as regras do *snort* em regras do *snort_inline*, é necessário uma ferramenta chamada de *snortconfig*.

Inicialmente, nesta *honeynet*, o *snort_inline* está configurado sem nenhuma regra de bloqueio ou modificação de pacotes. Isso porque, no período inicial, é necessário avaliar que tipos de ataques efetivamente serão iniciados da *honeynet*. O *snort_inline* está focalizado apenas para ataques de saída da *honeynet*, pois os ataques de entrada são todos permitidos e desejados. Para os ataques de saída já existe uma proteção bastante efetiva que é o limitador de conexão implementado pelo script *rc.firewall*.

Um último mecanismo de controle implementado na *honeynet* são os scripts *md5-cria.sh*, *md5-cria-sol* e *md5-verify.sh*, contruídos para criar uma base de *hashs* md5 dos principais arquivos binários e de configuração dos *honeypots* e depois de um tempo no ar, compará-los para verificar se algum arquivo importante foi modificado ou substituido pelo invasor. Foi feita a tentativa de usar a solução *tripwire* para criação e verificação dos *hashs*, mas não foi possível devido à incompatibilidade com a arquitetura SPARC.

7.2.1.3 Captura de dados

O objetivo principal da captura é logar todas as atividades do atacante. O segredo é fazer essa captura no maior número de camadas possível. Por exemplo: log do *firewall*, tráfego de rede, atividades do sistema, etc.

No caso desta *honeynet*, uma das principais ferramentas utilizada para captura de atividades do sistema, o *sebek*, não pôde ser utilizada, devido à incompatibilidade com a arquitetura SPARC.

O script *rc.firewall* habilita o iptables a logar qualquer conexão de entrada ou saída via *syslog* em /var/log/messages. É uma informação crítica e importante na hora de determinar quando um ataque foi iniciado.

Outro elemento de captura de dados utilizado é o *snort*. Ele foi configurado com as regras mais atualizadas para logar qualquer tipo de ataque conhecido que chegar pela interface

de entrada do muro. Alem disso, ele foi configurado para logar tudo numa base de dados *mysql* e foi acrescentado o software ACID, uma interface *web* que facilita a visualização e controle dos possíveis ataques.

Finalmente, a última camada de captura de dados fica por conta do *tcpdump*. Ele captura todo e qualquer tráfego de rede destinado ou a partir dos endereços da *honeynet*, salvando em um arquivo no formato binário, possibilitando assim futura análise e filtragem em diversas ferramentas existentes, inclusive a ferramenta *dsni*, desenvolvida em C.

7.2.2 AS REDES NEURAIS

Foi utilizado um modelo de redes neurais híbrido para a solução. De um lado, uma rede neural artificial baseada em treinamento não supervisionado foi utilizada para tirar uma impressão de cada período de tráfego individualmente. De outro lado, uma rede neural artificial baseada em treinamento supervisionado foi utilizada para construção de um sistema de detecção de intrusões baseado em algoritmos não lineares.

7.2.2.1 Mapas de Kohonen

O *software* de código aberto SOM_PAK, escrito em linguagem C, foi construído na Finlândia pela equipe do próprio criador do algoritmo de mapas auto-organizáveis: Tuevo Kohonen. A partir deste *software* foi possível construir representações visuais em duas dimensões do tráfego de rede capturado na *honeynet* e tratado pelo *dsni*.

A *honeynet* permaneceu ligada de 08/06/2005 a 30/07/2005. Todo o tráfego de rede que chegava ou saía do muro (*gateway* da *honeynet*) foi capturado pelo *tcpdump* e salvo em formato binário. Após o período de captura, a *honeynet* foi desligada e os pacotes de rede foram tratados pelo *software dsni*. A tabela 7.1 mostra quais foram os parâmetros de rede utilizados para o treinamento da rede neural não supervisionada.

Nome	***Descrição***	***Modificação***
Protocolo	Define o protocolo: TCP, UDP ou ICMP	Sem modificações
IP_Origem	Endereço IP de origem do pacote	Utilizada conversão para escala logaritmica

Nome	*Descrição*	*Modificação*
IP_Destino	Endereço IP de destino do pacote	Utilizada conversão para escala logaritmica
Porta_Origem	Porta de origem	Utilizada conversão para escala logaritmica
Porta_Destino	Porta de destino	Utilizada conversão para escala logaritmica
Flags	Flags utilizados pelo protocolo TCP.	Sem modificações
Tamanho	Tamanho do pacote IP.	Utilizada conversão para escala logaritmica

Tabela 7.1 – Parâmetros de entrada da rede neural

É importante notar que apenas o protocolo IP é utilizado. O campo “Protocolo” define o protocolo da camada superior que utilizará o IP. A escala logaritmica foi utilizada na maioria dos campos devido à diferença de magnitude entre os parâmetros. Todos eles pretendem ter igual importância para o treinamento da rede e portanto todos devem estar na mesma ordem de grandeza, problema resolvido pela escala logaritmica, implementada pelo *dsni*. O campo “Flags” tem o valor zero quando o protocolo não é o TCP. O campo “Tamanho” diz respeito a todo o pacote, incluindo o *payload*.

Estes parâmetros foram escolhidos como entradas tanto da rede de Kohonen, quanto da rede MLP. São os parâmetros fixos dos protocolos da camada de rede e de transporte que melhor definem um tráfego de rede. Além deles, seria interessante utilizar o campo de *payload*. Porém, as dificuldades de tratamento deste campo (possui formato, mensagem e tamanho imprevisíveis) impossibilitaram sua utilização nesta prova de conceito. O parâmetro *timestamp*, disponível no *log* do *tcpdump* também seria importante para determinar a frequência de chegada dos pacotes. A dificuldade de normalização deste parâmetro impossibilitou sua utilização neste primeiro momento. Na verdade, tentou-se utilizar um parâmetro chamado *data_id*, que seria apenas um contador de pacotes. Porém, após algumas tentativas, percebeu-se que este parâmetro estava influenciando decisivamente todos os treinamentos e decidiu-se por tirá-lo. Por último, uma análise do estado da conexão TCP seria importante no treinamento, porém é uma análise muito complicada de se fazer e necessitaria de um *software* bem mais complexo que o *dsni*.

O tráfego capturado na *honeynet* foi dividido em 30 arquivos contendo as amostras de treinamento e validação. Cada arquivo continha 1, 2, 3 ou até 4 dias de tráfego capturados. A partir daí, seguiu-se o treinamento da rede, utilizando os mesmos parâmetros para cada

arquivo. A representação resultante foi constituída de mapas 6x6, com 36 neurônios. A tabela 7.2 apresenta os parâmetros utilizados no treinamento.

Número de épocas	10
Tipo de topologia	Hexagonal
Medida de vizinhança	Gaussiana
Dimensões	6x6
Tamanho do treinamento	22500
Taxa de aprendizagem do treinamento	0,5
Raio inicial do treinamento	10
Tamanho da validação	2500
Taxa de aprendizagem da validação	0,4
Raio inicial da validação	5

Tabela 7.2 – Parâmetros de treinamento da rede SOM

Os parâmetros de treinamento foram determinados após exaustivos testes com os mais diversos parâmetros possíveis. Ou seja, só foi possível um resultado satisfatório empiricamente após inúmeras tentativas.

Após o treinamento, é gerado um arquivo contendo o valor de ativação de cada neurônio, resultante da atualização dos pesos de vizinhança implementada pelo algoritmo de Kohonen. A partir destes valores, são criados mapas de matrizes bidimensionais em escala de cinza, em formato *postscript*. Todas essas funções são implementadas pelo *software* SOM_PAK. Os mapas resultantes serão analisados posteriormente para identificar a semelhança de tráfego e, conseqüentemente, dos ataques recebidos.

7.2.2.2 Rede Perceptron Multicamadas

O *software EasyNN plus* versão 6.0d *trial* foi utilizado para o treinamento da rede neural supervisionada utilizando o algoritmo *backpropagation*. A facilidade de configuração dos parâmetros de treinamento incluindo número de camadas, de neurônios, taxa de aprendizagem, critérios de parada, manipulação dos dados de treinamento e validação, além da visualização do treinamento com gráficos em tempo real foram determinantes para a escolha deste *software*.

Os parâmetros de entrada escolhidos foram os mesmos da rede de Kohonen, pelos

mesmos motivos expostos, o número de neurônios da camada oculta foram escolhidos empiricamente após vários treinamentos fracassados e a camada de saída foi constituída de um neurônio. O valor era aproximado para zero, indicando tráfego normal ou para um, indicando tráfego de ataque.

Serão discutidos, no próximo tópico, os resultados da análise realizada no tráfego da *honeynet*. Os tipos de ataque foram identificados principalmente através de alertas do *snort* e análise dos pacotes TCP/IP (software *dsni*). Uma vez definidos os tipos de ataque, redes especialistas eram construídas para treinar cada tipo separadamente.

7.3 OS RESULTADOS

Depois de quase dois meses no ar com a configuração apresentada anteriormente, a *honeynet* gerou *logs* do *snort*, *syslogs* e principalmente *logs* de tráfego de rede, que foram divididos em 30 arquivos binários, seguindo uma sequência cronológica de 51 dias corridos.

Estes arquivos sofreram uma filtragem com o programa *dsni*. Dois parâmetros foram utilizados para enxugar o tráfego e separar o que realmente interessava para identificar os ataques: foram considerados apenas pacotes que utilizaram o protocolo IP e pacotes cuja origem ou destino seja o endereço dos *honeypots*.

Após isso, os 30 arquivos resultantes foram apresentados. Havia uma disparidade muito grande no número de pacotes de cada um destes arquivos. Enquanto o dia 09/06 apresentava 41 pacotes de rede, o dia 22/07 apresentava 146.410 pacotes. Diversos treinamentos foram realizados com arquivos de tamanhos diferentes e o resultado final não foi o esperado. Decidiu-se, então, treinar as redes não supervisionadas SOM com o mesmo número de pacotes para obter mapas uniformes quanto ao tamanho.

Todo o tráfego foi dividido em 17 arquivos com 25.000 pacotes cada um e mais um arquivo com os 2333 pacotes restantes, de acordo com sua ordem cronológica. Os 18 arquivos foram treinados um a um na rede neural não supervisionada SOM de acordo com os parâmetros de treinamento da tabela 7.2. A partir dos valores de ativação resultantes, foram criados os mapas de duas dimensões apresentados no próximo tópico.

7.3.1 A REDE SOM E OS ATAQUES SOFRIDOS

As figuras 7.2, 7.3 e 7.4 ilustram os mapas 6x6 resultantes da filtragem dos pacotes da *honeynet* e treinamento da rede SOM. A tabela 7.4 resume a representatividade de cada mapa de acordo com o período de tráfego capturado.

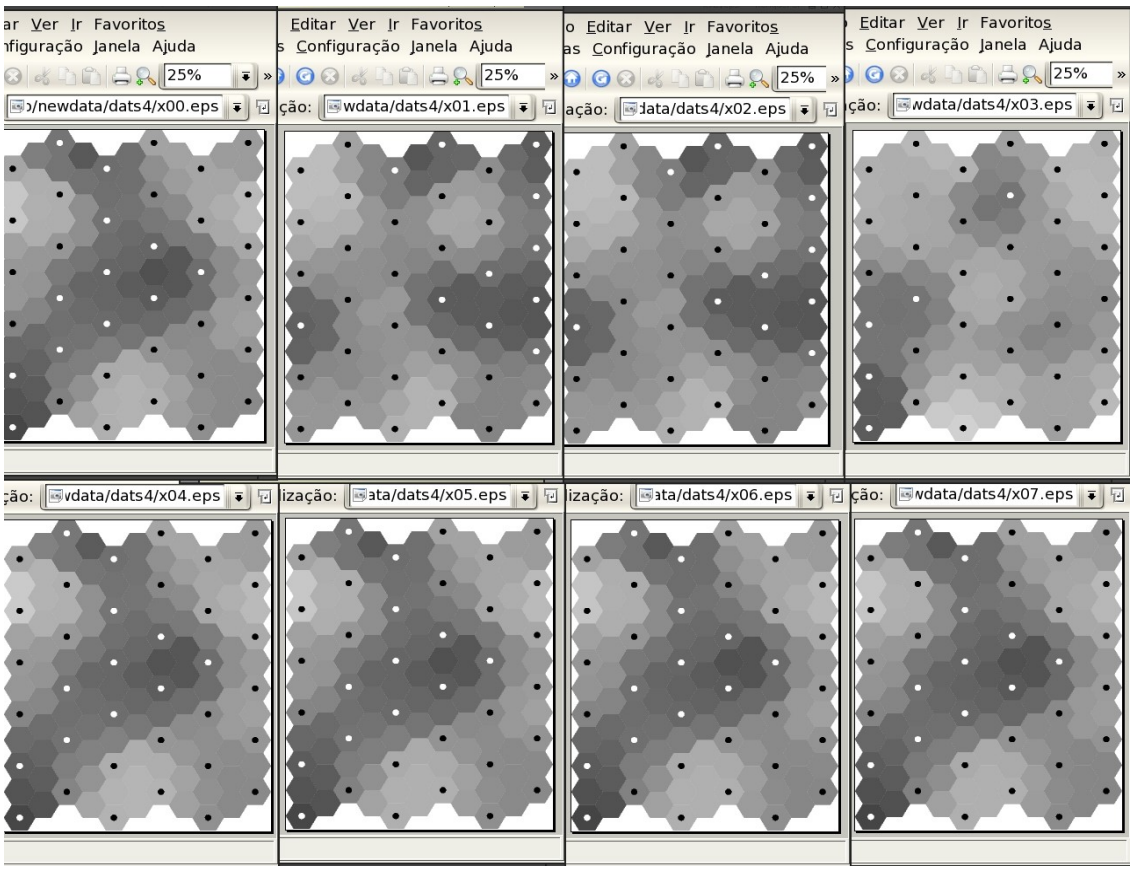


Figura 7.2 – Mapas de Kohonen, 00 a 07

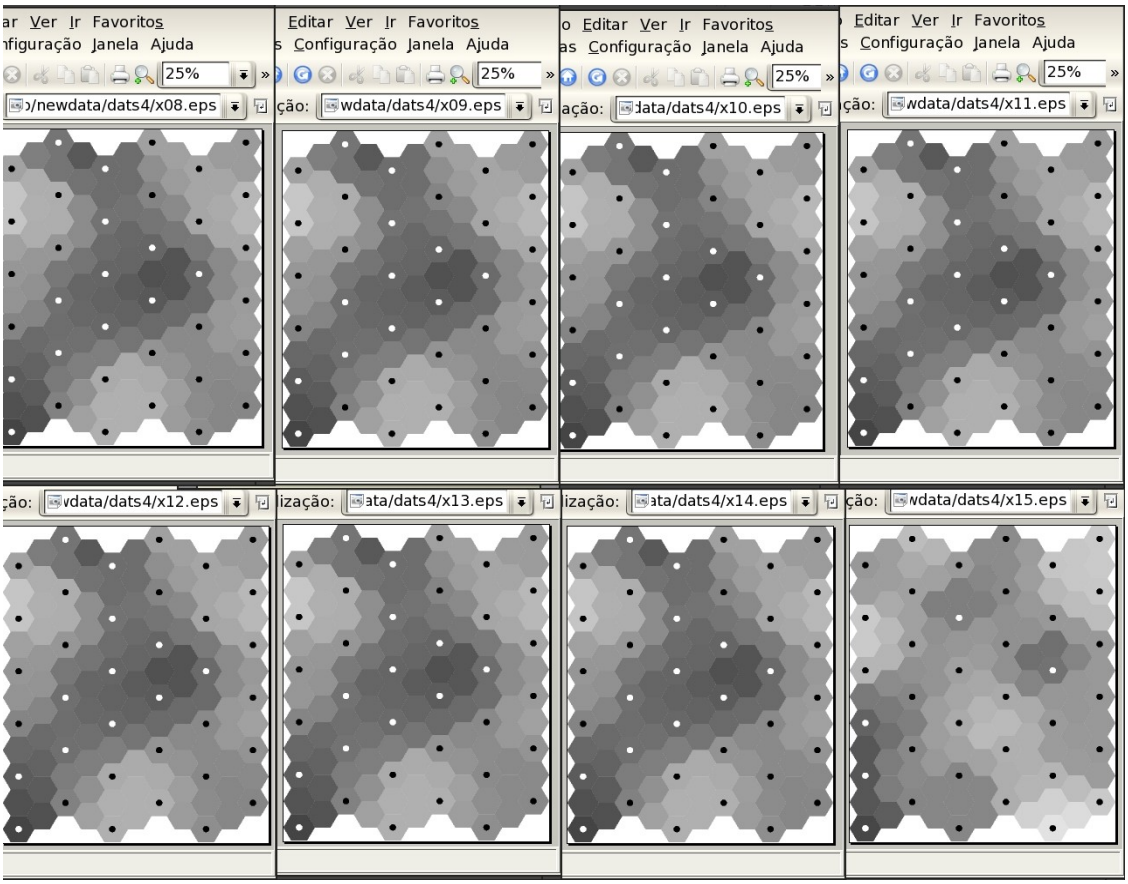


Figura 7.3 – Mapas de Kohonen, 08 a 15

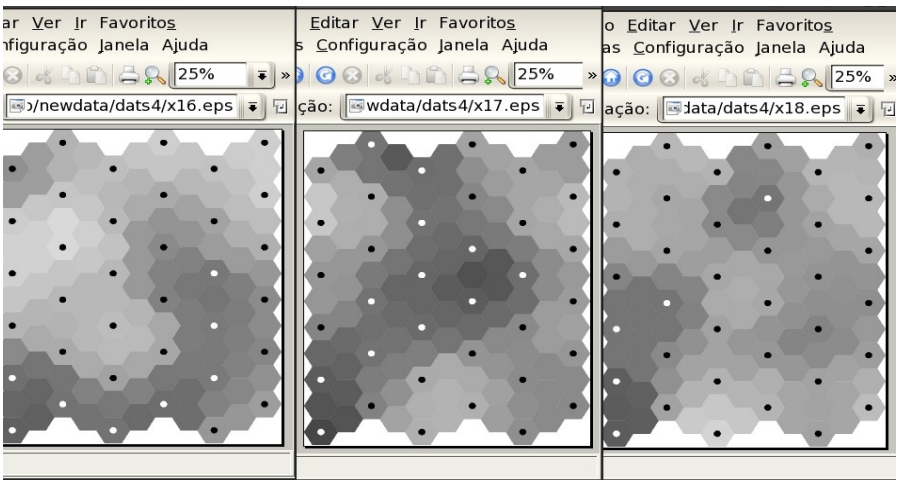


Figura 7.4 – Mapas de Kohonen, 16 a 18

Mapa	***Período***	***Mapa***	***Período***
0	08 a 21 de junho	10	22 de julho
1	21 de junho	11	22 de julho
2	21 de junho	12	22 de julho
3	21 de junho a 13 de julho	13	22 de julho

Mapa	*Período*	*Mapa*	*Período*
4	13 a 21 de julho	14	22 de julho
5	21 a 22 de julho	15	22 de julho
6	22 de julho	16	22 a 27 de julho
7	22 de julho	17	27 a 28 de julho
8	22 de julho	18	28 de julho
9	22 de julho		

Tabela 7.3 – Divisão dos mapas por período

Nota-se claramente que os mapas x00, x04, x05, x06, x07, x08, x09, x10, x11, x12, x13, x14 e x17 são extremamente próximos, ou quase idênticos. Isso fornece uma noção de que um mesmo ataque, ou ataques com um mesmo padrão aconteceram durante estes períodos.

Uma análise profunda foi realizada nos alertas gerados pelo snort e nos pacotes de rede capturados na *honeynet*. Os ataques foram identificados e classificados em 5 categorias principais. São elas: ataque de "*brute force*" (força bruta) no serviço de *ssh;* ataque de *scanning;* ataque de exploração de HTTP (porta 80); ataque de exploração de SMTP (porta 25); ataques automáticos (basicamente *worms*).

Uma rápida análise nos alertas do *snort* apontou o dia 21/06 com um alto índice de alertas de *port scanning*. Ao analisar mais profundamente o tráfego de rede deste dia, constatou-se que a grande maioria dos pacotes TCP tinham a mesma origem e varriam diversas portas do *honeypot* aleatoriamente, com o *flag SYN* setado e recebendo como resposta pacotes com o *flag RST,* além de pacotes UDP varrendo diversas portas. Além disso, diversos outros alertas idênticos foram encontrados em outras datas e com outras origens. Um filtro foi utilizado no software *dsni* para retirar os pacotes provenientes destas origens, nos dias especificados. A primeira rede especialista estava pronta para ser treinada.

Ao analisar o tráfego pacote por pacote, percebe-se claramente a predominância de ataques de *brute force*, ou seja, o atacante, através de uma ferramenta automatizada, tenta se autenticar no serviço de *ssh* utilzando um dicionário de usuários e senhas mais comuns. Entretanto, as senhas utilizadas nas *honeypots* eram bastante fortes (utilizavam caracteres numéricos, alfa-numéricos e especiais) e é praticamente certo que nenhum atacante conseguiu entrar desta maneira. Através de um filtro no *dsni* para retirar pacotes na porta 22, a segunda rede especialista estava pronta para ser treinada.

Além do serviço de *ssh*, mais dois serviços foram deixados habilitados propositalmente. Eram eles: o HTTP e o SMTP. Desta maneira, algumas explorações foram realizadas nestas portas durante o período de vida da *honeynet*. A exploração consiste na fase de levantamento de dados (*fingerprinting* para saber o *software* e a versão instalados) e de exploração propriamente dita (*exploit*). A grande dificuldade em treinar ataques deste tipo seria analisar o *payload* dos pacotes, pois o ataque, neste caso, está todo no *payload*. Desta maneira, para esta prova de conceito, decidiu-se por ignorar estes ataques no treinamento das redes especialistas.

A última categoria de ataque foi identificada primeiramente através dos *logs* do *snort*. O *worm Slammer*, que ataca sistemas que possuem *SQL Server* 2000 da *Microsoft*, causou uma série de alertas. Ele funciona através de datagramas UDP enviados para a porta 1434 com determinados *bytes* (por exemplo 0x08) que causam efeitos indesejados no servidor, como negação de serviço. Analisando mais profundamente os pacotes de rede, descobriu-se diversos outros ataques automáticos disseminados. A porta 137 era constantemente atacada em busca de compartilhamentos *Netbios*; as portas 1026, 1027 e 1028, utilizadas pelo *Windows Messenger* eram alvo constante de *spams*; a porta 1080 também sofria ataques na busca de *socks;* portas 10000, 139, 135, 5554, 8078, 4065, 4073, 500, 1030 e assim por diante. A rigor, um treinamento para ataques deste tipo tinha que levar em consideração também o *payload* dos pacotes, pois é lá que está o ataque propriamente dito. Todavia, desta vez, optou-se por realizar o treinamento por se tratar de portas específicas, normalmente acima de 1024. A terceira e última rede neural especialista estava pronta para ser treinada, apesar de não se esperar uma precisão muito grande da mesma.

Os serviços prestados pela *honeynet*, apesar de serem reais, não pretendiam ludibriar o atacante para que ele pensasse que fossem serviços importantes, ou pertencentes a alguma organização. Por exemplo, o servidor HTTP apesar de funcionando só exibia a página teste do Apache. Assim como os servidores SMTP e SSH. A impressão que o atacante teria é a de que o servidor foi mal configurado e colocado na Internet, mas não serviria para nada. Sendo assim, já era de se esperar que o atacante não teria interesse em realizar um ataque de DoS, a menos que fosse para teste. As suspeitas vão se confirmando ao desconectar a *honeynet* após os 51 dias e verificar que todos os serviços permanecem disponíveis, assim como a máquina. Dos arquivos de tráfego resultantes, apenas o dia 22/07 tem um tamanho exageradamente maior do que os outros(característica típica de um ataque de DoS). Analisando melhor este

arquivo descobre-se que a metade do tráfego refere-se a uma comunicação na porta 22(ataque *brute force*) entre um endeteço da UFRJ e um *honeypot* e a outra metade refere-se a uma comunicação do mesmo tipo entre um endereço da Dinamarca e o mesmo *honeypot*. Uma rede especialista de DoS não poderia ser treinada.

A tabela 7.5 ilustra a distribuição dos ataques encontrados por mapa SOM construído (a cada 25.000 pacotes). Os números foram obtidos através da filtragem dentro dos arquivos de tráfego, realizada por *scripts bash* que faziam estatística de acordo com a característica dos ataques (filtros explicitados anteriormente).

	BF22	***SCAN***	***Ex80***	***Ex25***	***Ataut***
TOTAL	***82,97%***	***13,41%***	***2,99%***	***0,13%***	***0,5%***
x00	70,0%	27,0%	2,1%	0,3%	0,6%
x01	2,9%	92,1%	4,7%	0,3%	0,00%
x02	0,4%	92,9%	6,0%	0,7%	0,00%
x03	57,2%	15,4%	17,4%	0,6%	9,4%
x04	82,1%	0,00%	11,4%	0,00%	6,5%
x05	97,2%	0,00%	1,6%	0,00%	1,2%
x06	99,9%	0,00%	0,00%	0,00%	0,1%
x07	99,9%	0,00%	0,00%	0,00%	0,1%
x08	99,9%	0,00%	0,00%	0,00%	0,1%
x09	99,9%	0,00%	0,00%	0,00%	0,1%
x10	99,6%	0,00%	0,1%	0,00%	0,3%
x11	99,9%	0,00%	0,00%	0,00%	0,1%
x12	99,9%	0,00%	0,00%	0,00%	0,1%
x13	99,9%	0,00%	0,00%	0,00%	0,1%
x14	99,8%	0,00%	0,1%	0,00%	0,1%
x15	100,0%	0,00%	0,00%	0,00%	0,00%
x16	97,7%	0,1%	0,2%	0,00%	2,00%
x17	92,7%	0,00%	5,0%	0,00%	2,3%
x18	18,5%	0,00%	56,7%	1,6%	22,9%

Tabela 7.4 – Tipos de ataque, por mapa

Os números demonstram a total predominância de ataques de força bruta na *honeynet*, sendo que o dia 22/07 foi o principal responsável. Eles comprovaram o que os mapas haviam

ilustrado antes da análise, ou seja, a presença desproporcional de um mesmo ataque.

Os mapas x01 e x02 demonstram o padrão formado pela predominância do ataque de *scanning.*

O mapa x03 tem como característica a mistura entre os tipos de ataques. O curioso é que o mapa x18 também possui como característica a mistura entre os ataques, porém com proporções completamente diferentes. Mesmo o tamanho do arquivo era diferente. Os mapas resultantes foram os mesmos.

Os mapas x15 e x16, apesar da predominância clara dos ataques de força bruta, não seguiram o mesmo padrão dos demais e nem são iguais entre si. Analisando particularmente os dois tráfegos, a única diferença para os demais é que os ataques de força bruta existentes pertencem todos a uma mesma origem. Este ataque começa no mapa x15 (e inclusive não há nenhum outro pacote neste mapa que não seja este ataque) e só termina no mapa x17 (o mapa x16 apresenta alguns poucos pacotes referentes a outro ataque).

Os demais mapas apresentaram todos o mesmo padrão. Quer dizer, predominância dos ataques *brute force*.

Conclui-se, então, que a presença dos mapas de Kohonen numa análise rápida dos tipos de ataque presentes em um determinado tráfego de rede pode ser de extrema importância. Entretanto, a técnica não é infalível. Houveram, nesta prova de conceito, dois ataques que seguiam claramente o mesmo padrão dos demais e não se encaixaram em seus mapas (x15 e x16).

7.3.2 MLP

Conforme visto no tópico anterior, chegou-se a três tipos de redes especialistas a serem treinadas: a rede “bf22” (*brute force ssh*), a rede “scan” (*scanners*) e a rede “ataut” (ataques automáticos). A partir da filtragem destes ataques no tráfego total da *honeynet*, realizada através de *scripts bash*, foi possível coletar as amostras de tráfego de ataque para treinamento da rede MLP. O tráfego normal foi coletado a partir de um *tcpdump* de alguns minutos em uma rede corporativa. Enquanto a coleta ocorria, um *snort* capturava possíveis ataques para garantir que o tráfego era normal. A tabela 7.5 ilustra os parâmetros utilizados para treinamento e os parâmetros de resultado das redes especialistas no *software easyNN*. Chegou-se a alguns desses parâmetros empiricamente após diversos treinamentos e a outros através de sugestões da bibliografia estudada e do próprio *software* utilizado.

Parâmetros	**Bf22**	**Scan**	**Ataut**
Camada de entrada	7	7	7
Camada oculta	9	9	9
Camada de saída	1	1	1
Taxa de aprendizagem	0,3	0,3	0,3
Momentum	0,15	0,15	0,15
Erro máximo	0,05	0,05	0,05
Amostras de treinamento	9499	8999	10900
Amostras de validação	500	999	1200
Amostras de teste	100	100	100
Erro médio de treinamento	0,000269	0,000342	0,000249
Ciclos de aprendizagem	33431	21063	18371

Tabela 7.5 – Parâmetros de treinamento da rede MLP

Os 7 neurônios de entrada representam os mesmos parâmetros utilizados para treinamento da rede SOM, ou seja, os parâmetros da tabela 7.1. O número de neurônios da camada oculta foram determinados por tentativa e erro, assim como a taxa de aprendizagem e o momentum. A camada de saída possui dois resultados possíveis: 0 para tráfego normal e 1 para tráfego de ataque. O critério de escolha de amostras de ataque foi o de separar todas as amostras de determinado ataque e rodar um *script* de coleta aleatória. A exceção fica por conta dos ataques automáticos. Todas as amostras foram utilizadas no treinamento, dentre amostras de treinamento, validação e teste. O mesmo *script* é utilizado para coleta de amostras normais.

A figura 7.5 ilustra a topologia de cada rede especialista, assim como os pesos de cada sinapse.

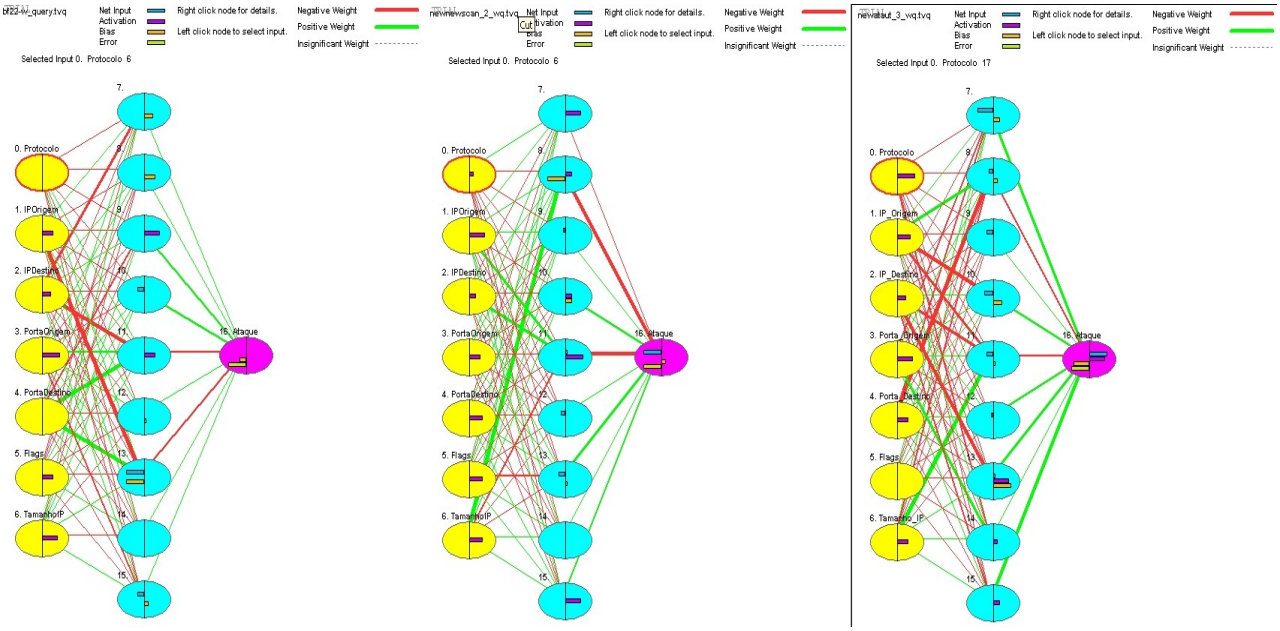
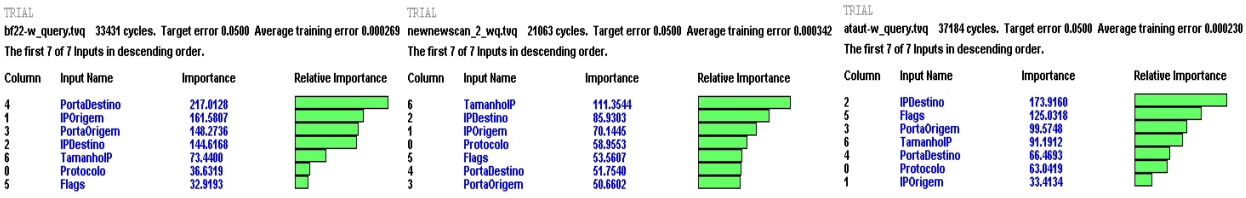


Figura 7.5 – Topologia das redes *bf22*, *scan* e *ataut* respectivamente

Todas as redes foram treinadas com 7 camadas de entrada, 9 camadas ocultas e uma camada de saída. Cada neurônio está conectado com todos os neurônios da camada posterior. Os valores de ativação, erro e bias, após o treinamento, estão representados dentro de cada neurônio, conforme legenda. Os valores dos pesos de cada neurônio estão representados pela sinapse entre eles. Os traços mais grossos representam os neurônios com maiores valores de peso, ou seja, os que exercem maior influência sobre a decisão final de detecção de um pacote, sejam eles negativos ou positivos. A figura 7.6 ilustra a importância relativa de cada neurônio da camada de entrada (cada parâmetro do pacote) na decisão final de detecção.



TRIAL
bf22-w_query.tvq 33431 cycles. Target error 0.0500 Average training error 0.000269
The first 7 of 7 Inputs in descending order.

Column	Input Name	Importance	Relative Importance
4	PortaDestino	217.0128	
1	IPOrigem	161.5807	
3	PortaOrigem	148.2736	
2	IPDestino	144.6168	
6	TamanhoIP	73.4400	
0	Protocolo	36.6319	
5	Flags	32.9193	

TRIAL
newnewscan_2_wq.tvq 21063 cycles. Target error 0.0500 Average training error 0.000342
The first 7 of 7 Inputs in descending order.

Column	Input Name	Importance	Relative Importance
6	TamanhoIP	111.3544	
2	IPDestino	85.9303	
1	IPOrigem	70.1445	
0	Protocolo	58.9553	
5	Flags	53.5687	
4	PortaDestino	51.7540	
3	PortaOrigem	50.6602	

TRIAL
ataut-w_query.tvq 37184 cycles. Target error 0.0500 Average training error 0.000230
The first 7 of 7 Inputs in descending order.

Column	Input Name	Importance	Relative Importance
2	IPDestino	173.9160	
5	Flags	125.0318	
3	PortaOrigem	99.5748	
6	TamanhoIP	91.1912	
4	PortaDestino	66.4693	
0	Protocolo	63.0419	
1	IPOrigem	33.4134	

Figura 7.6 – Importância relativa dos parâmetros de entrada em cada rede.

A figura 7.7 ilustra o gráfico de treinamento das redes especialistas, bem como as principais informações como número de ciclos, erros e exemplos de validação (vide tabela 7.5).

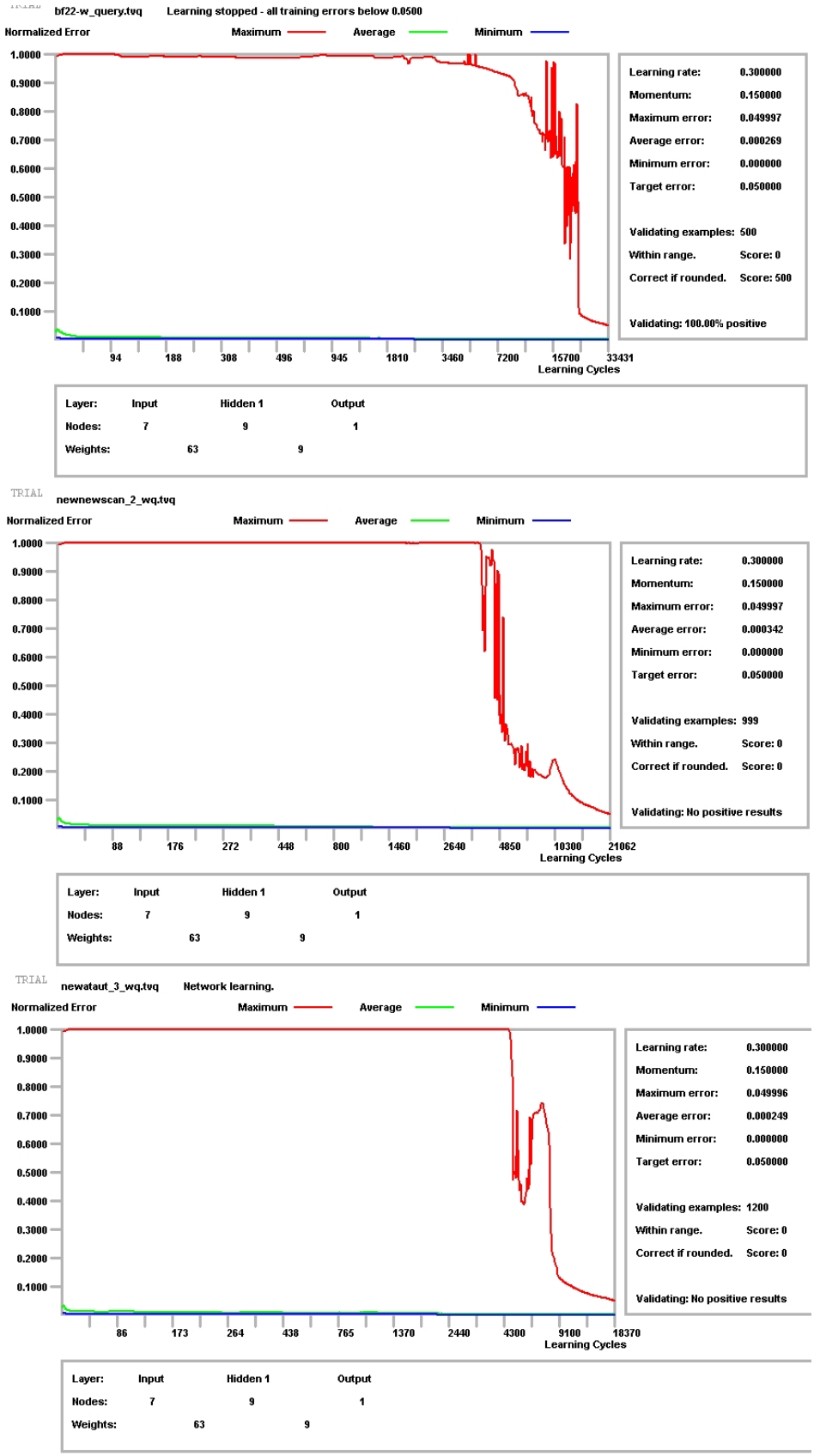


Figura 7.7 – Gráfico de treinamento das 3 redes especialistas

Após o treinamento, foram realizados alguns testes de detecção com padrões não

treinados pelas redes. A tabela 7.6 resume os testes.

	Bf22	***Scan***	***Ataut***
Acertos total	100,0%	74,0%	92,0%
Acertos tráfego normal	100,0%	88,0%	84,0%
Acertos tráfego ataque	100,0%	60,0%	100,0%
Falsos Positivos	0,0%	6,0%	8,0%
Falsos Negativos	0,0%	20,0%	0,0%
Acertos ataques manuais	100,0%	0,0%	------

Tabela 7.6 – Testes das redes neurais especialistas

Cada rede especialista foi testada com 100 novos pacotes de rede (não incluídos no treinamento ou validação). Destes 100 pacotes, 50 eram de tráfego normal e 50 de tráfego de ataque. Todas as amostras de tráfego normal foram coletadas em uma nova situação e nada tinham a ver com as amostras treinadas e validadas pelas redes. Das amostras de ataque, 60% (30) foram testadas com pacotes capturados da *honeynet*, mas que não foram treinados nem validados e 40% foram testadas com pacotes de ataque lançados manualmente. No caso do ataque de *bf22*, uma estação *slackware linux* com servidor de *ssh*, em uma rede interna, foi atacado utilizando uma série de usuários e senhas falsos. No caso do ataque de *scan*, um ataque de *scanning tcp* e *scanning udp* foi realizado utilizando a ferramenta *nmap* de uma estação *slackware linux* para outra estação *slackware linux* na mesma rede interna. No caso do ataque *ataut*, não foi possível simular um ataque manualmente e todas as amostras de ataque testadas foram colhidas da *honeynet* e não incluídas no treinamento ou validação.

A tabela mostra que o ataque de força bruta foi o que melhor se saiu nos testes. Acertou 100% das amostras, sendo que no tráfego normal, quase metade do tráfego era de *ssh* legítimo na porta 22. Os ataques manuais foram realizados em faixas de endereços diferentes das treinadas.

O ataque de *scan* foi o que obteve os piores resultados. Uma taxa de 6% de falsos positivos, ou seja, classificou erroneamente 6 amostras normais. E uma taxa de falsos negativos de 20%, ou seja, classificou erroneamente 20 amostras de ataque. Curiosamente, essas 20 amostras erradas foram os ataques manuais realizados. Duas hipóteses principais para este fracasso na classificação dos ataques manuais: o segundo e o terceiro parâmetros mais importante na decisão de um ataque de *scan*, pelo treinamento (figura 7.6), são IPs de

destino e IPs de origem. A rede utilizada para os ataques manuais tinham como origem e destino IPs de uma rede interna, ou seja, completamente diferentes dos IPs treinados na rede MLP; a ferramenta utilizada para o *scanning* foi o *nmap*. Apesar de ser uma ferramenta bastante popular, não é garantido que ela tenha sido utilizada pelos atacantes da *honeynet.* Pode-se dizer que a rede pode ter sofrido a chamada super especialização, ou seja, a grosso modo ela pode ter "decorado" as amostras de treinamento, não sendo capaz de generalizar suas decisões.

Os ataques automáticos tiveram uma taxa de acerto bem razoável. Todavia, não foram realizados os testes de ataque manual e a capacidade de generalização da rede não foi testada. No treinamento deste ataque, o IPDestino foi considerado o parâmetro com a maior importância.

O número de amostras utilizadas para teste foi muito reduzida devido a uma dificuldade de operação da versão *trial* do *software EasyNN*. Cada parâmetro de cada amostra tem que ser digitado separadamente. O esforço de testar os 300 novos pacotes demandou horas de trabalho manual. Seria desejável em um próximo momento a versão completa do *software* ou um outro *software* de teste para que mais amostras possam ser testadas e a rede possa ser melhor avaliada quanto a sua eficiência capacidade de generalização.

7.4 CONSIDERAÇÕES FINAIS DA PROVA DE CONCEITO

A solução de detecção de intrusão através de redes neurais artificiais, utilizando dados colhidos em uma *honeynet* real, que foi construída durante todo o trabalho, foi testada e apresentada neste capítulo. A utilização de redes neurais não supervisionadas, SOM, foi muito importante para indicar a distribuição do tráfego de ataque durante o período de captura. Ficou claro, após a leitura dos mapas, que um mesmo padrão se repetia em quase todo o período. Com a análise detalhada posterior veio a confirmação. A idéia é que, em um universo muito grande de tráfego, o analista poderá olhar os mapas resultantes e, baseado em sua experiência, separar os tipos de ataques para análise e treinamento.

A proposta inicial de treinamento de redes neurais especialistas para cada tipo de ataque foi realizada. As amostras de tráfego para treinamento foram colhidas aleatoriamente do total de cada tipo de ataque e o tráfego normal aleatoriamente de uma rede em produção.

Entretanto, nem todos os ataques capturados puderam ser treinados e mesmo os que foram treinados não tiveram uma eficiência total. Isto porque alguns parâmetros essenciais do tráfego de rede não puderam ser levados em consideração neste primeiro momento, são eles o *payload* do pacote, a frequência dos pacotes e o estado da conexão. Além disso, alguns tipos importantes de ataque não foram capturados, devido à própria natureza da *honeynet* construída. Por exemplo, o ataque de DoS. A rede especialista que obteve o maior sucesso foi a de ataques *brute force*, com 100% de detecção nos testes. É sabido que ataques deste tipo não necessitam de uma rede neural artificial para serem detectados ou barrados. Um simples sistema de contagem e registro das tentativas de *login* é suficiente. Porém, além deste ataque ter sido o mais realizado na *honeynet*, os resultados provam a capacidade de uma rede neural, se bem treinada, de detectar com total precisão determinado ataque.

8 CONCLUSÃO

Estar conectado na Internet, nos dias de hoje, não é apenas ter um computador com sistema operacional instalado e um modem com acesso à grande rede. Se o sistema estiver desatualizado ou desprotegido, em poucos minutos ele poderá ser comprometido por ataques automáticos ou manuais existentes na rede. A preocupação com a segurança é cada vez mais presente nos usuários e, principalmente, nas corporações representadas na Internet. A quantidade de atacantes virtuais é cada vez maior e as ferramentas utilizadas por eles, cada vez mais sofisticadas. Conseqüentemente, as soluções de segurança são as mais diversificadas possíveis e tentam cobrir as mais diversas falhas existentes em protocolos e *softwares*.

O presente trabalho procurou demonstrar, através de capítulos teóricos, o funcionamento básico da Internet, com sua arquitetura, seus protocolos e seus pontos vulneráveis. A discussão mais aprofundada sobre os diversos protocolos existentes, bem como os detalhes de todos os ataques e falhas nos sistemas existentes vai muito além do trabalho. A idéia básica foi fornecer susbsídios para que o leitor compreenda o conteúdo da proposta e a importância de buscar alternativas às proteções já existentes contra ataques cibernéticos.

A proposta prática consiste no treinamento de redes neurais artificiais MLP com algoritmo *backpropagation* para detecção de intrusão. Diversos outros trabalhos sobre o assunto já foram publicados em Universidades de todo o Mundo. Entretanto, este apresenta três importantes contribuições em relação aos anteriores.

A primeira contribuição diz respeito à base de dados utilizada para coleta dos dados de ataque. Enquanto que a maioria dos estudos e experimentos sobre o assunto utiliza uma base de ataque defasada e inserida em um contexto específico: a base DARPA, construída em 1998 emulando servidores da força aérea americana; ou uma base construída através de ataques simulados pelo próprio autor; este trabalho utiliza uma base recente, de uma rede real construída à imagem e semelhança da rede de produção que se deseja trabalhar. Esta rede real é a *honeynet*, uma rede construída exclusivamente para ser invadida e que possui a garantia que todo e qualquer tráfego recebido é de origem maliciosa e todo e qualquer tráfego iniciado a partir dela é realizado pelo invasor. É bem provável que os ataques recebidos por esta rede serão os mesmos ataques direcionados à produção. Ou seja, uma rede neural artificial treinada com amostras de ataque da *honeynet*, além de amostras de tráfego normal da própria produção estará apta a decidir se os pacotes recebidos na rede de produção são pacotes normais ou de

ataque.

Antes de serem treinados, os ataques recebidos na *honeynet* têm que ser devidamente identificados. Isso porque as características de um ataque diferem muito das características de outro e uma rede neural não pode ser construída simplesmente juntando todos no mesmo treinamento. A identificação tem que ser realizada através da análise humana e uma rede neural artificial MLP *backpropagation* especializada em cada tipo de ataque tem que ser construída. A construção dessas redes especialistas constitui a segunda contribuição importante do trabalho.

A análise humana é realizada através de *logs* dos mecanismos de captura da *honeynet* (IDS, firewall, sistema operacional, sniffer). Entretanto, uma inovação pode ser acrescentada para uma análise anterior aos *logs*, mais rápida e visual. É a última importante contribuição deste trabalho: a rede neural não supervisionada SOM, ou mapas de Kohonen. Através do treinamento destas redes neurais são criados mapas bidimensionais do tráfego e, de acordo com a experiência do analista, é possível determinar o ataque que foi predominante em determinado período de tempo, através da análise do mapa correspondente.

Uma lição importante aprendida no decorrer do trabalho foi que os parâmetros de treinamento, tanto da rede neural não supervisionada SOM, quanto da rede supervisionada MLP não são encontrados com facilidade e necessitam ser definidos empiricamente através de diversos treinamentos anteriores ao definitivo. Esses parâmetros são essenciais para o sucesso da solução.

Outra lição importante é que não importa o quão criteriosa e precisa seja a análise dos ataques recebidos em uma *honeynet*, sempre haverão ataques desconhecidos e novos que não serão classificados corretamente. Por isso, é importante que apenas ataques estritamente conhecidos sejam disponibilizados para treinamento.

Uma outra lição aprendida é que a fase de testes de uma rede neural MLP resultante é tão, ou mais importante que o treinamento propriamente dito. Os resultados demonstraram que o risco da rede neural especialista resultante se tornar super especializada, ou seja, não possuir a capacidade de generalização é grande. Neste caso, o treinamento precisaria ser refeito.

A última lição aprendida diz respeito à construção da *honeynet*. É essencial para o sucesso da solução que a *honeynet* construída seja praticamente uma cópia da rede de produção no que diz respeito a serviços, endereçamento e sistemas operacionais. Apenas desta

maneira, o treinamento da rede neural especialista resultante levará em consideração as reais diferenças entre o tráfego normal e o tráfego de ataque treinados. Lembrando que, para o treinamento, o tráfego normal é coletado diretamente da produção e o tráfego de ataque é coletado da *honeynet*.

O surgimento do mecanismo de reconhecimento de intrusões mais popular hoje em dia, chamado IDS (*Intrusion Detection System*) constituiu um grande avanço na batalha contra os atacantes. Entretanto, com o passar do tempo, mostrou ser um mecanismo bastante limitado e fadado ao desaparecimento. Tanto que, em 2003 o grupo de estudos *Gartner Group* decretou que, em poucos anos, o IDS seria substituído por outro mecanismo mais inteligente, o IPS (*Intrusion Prevention System*).

Assim que uma vulnerabilidade é descoberta, surgem um ou mais *exploits* com o objetivo de comprometer sistemas que possuem aquele furo. O IDS é um mecanismo baseado na assinatura do ataque *(exploit*), ou seja, o mecanismo apenas detecta o ataque se ele estiver idêntico ao *exploit* conhecido. O IPS é baseado na assinatura da vulnerabilidade, uma simples mudança no pacote de ataque não afetará na detecção do mesmo. Além disso, o IPS é um elemento extremamente ativo, atuando como uma barreira na frente da rede, não deixando, caso reconheça um ataque, que ele passe para o outro lado. O IDS, por outro lado, apesar de enviar pacotes de *RESET* para a conexão, só o faz depois que o ataque já está acontecendo.

Outra vantagem do IPS, é sua capacidade de realizar outras análises que não seja a análise baseada em assinaturas. São realizadas análises estatísticas de anomalia de tráfego e anomalia de protocolos. Quanto mais mecanismos que possam indicar a presença de um ataque, tanto melhor. Neste ponto, deve-se acrescentar um novo e promissor mecanismo: os algoritmos não-lineares, como redes neurais artificiais. Obviamente, as soluções deste tipo existentes hoje precisam evoluir muito para fazer parte de um sistema tão complexo e preciso em descobrir ataques. Ainda quando fizerem parte, não serão suficientes para detectar todo e qualquer tipo de ataque. As outras análises ainda serão necessárias e um sistema de pesos diferentes para cada tipo de diagnóstico poderá ser utilizado para definir se deve-se barrar ou não um tráfego de rede por ser ataque.

A solução apresentada no trabalho está completamente inserida no contexto desta nova solução, na busca de minar as limitações de um IDS comum. Foi intensamente discutido, durante o trabalho, as vantagens de generalização, de detecção de ataques novos e desconhecidos e a velocidade extremamente maior de detecção de uma rede neural, em

relação a um mecanismo baseado em assinatura. Entretanto, uma solução apenas baseada no algoritmo não-linear ainda não é possível. Utilizá-lo como um parâmetro na decisão de uma intrusão parece ser uma proposta sensata para um futuro próximo.

Da proposta desenvolvida no trabalho, alguns ajustes poderiam ser feitos para aumentar a efetividade e precisão da solução. Um deles, já discutido, seria inserir nos parâmetros de treinamento uma análise de *payload* do pacote, uma análise da frequência de chegada dos pacotes, assim como uma análise de estado das conexões. Outra melhora seria aplicar uma rotina de treinamento periódica, visto que os ataques não são estáticos e estão sempre mudando ao longo do tempo, além poderem ressurgir de tempos em tempos.

Uma possível rotina de treinamento periódico seria:

a) identifica-se um ataque na *honeynet* (configurada o mais próximo possível da rede de produção);

b) treina-se a rede específica do ataque;

c) coloca-se em produção (não como um decisor absoluto de intrusão, mas como um peso em um sistema mais complexo). Caso o ataque já tenha sido treinado alguma vez, substituí-lo, pois o padrão pode ter mudado.

Enfim, espera-se que este trabalho possa fornecer uma contribuição acadêmica que auxilie na construção futura de um poderoso sistema de detecção e prevenção de intrusões.

9 BIBLIOGRAFIA

[1] COMER, D., *Internetworking with TCP/IP – Volume I principles, protocols, and architeture* – third edition - Prentice Hall, 1995.

[2] STEVENS, W., R., *TCP/IP Illustrated, Volume 1 – The Protocols* – ADDISON-WESLEY – Tucson, Arizona – 1993.

[3] TANENBAUM, A., S., *Redes de Computadores* – tradução [da 3. ed. original] Insight Serviços de Informática – Rio de Janeiro - Campus, 1997.

[4] PETERSON, L., L.; DAVIE, B., S., *Computer Networks, A System Approach* – third edition - Morgan Kaufmann – 2003.

[5] CHESWICK, W., R.; BELLOVIN, S., M.; RUBIN, A., D., *Firewalls and Internet Security, Repelling the Wily Hacker* – second edition - ADDISON-WESLEY – 2003.

[6] KOZIOL, J., *Intrusion Detection with Snort* – Sams Publishing – 2003.

[7] The MIS Corporate Defence Solution Ltd., Network Security Team, *An Overview of Network Security Analysis and Penetration Testing, A guide to Computer Hacking and Preventative Measures* – August 2000.

[8] BREILING, S.; PLATO, A.; WINTER, K., *Blackice Guide to Computer Security* – Version 2.5 – Network ICE Corporation – 2001.

[9] PARMAR, S., K., *Information Resource Guide, Computer, Internet and Network Systems Security, An Introduction to Security.*

[10] TULLOCH, M., *Microsoft Encyclopedia of Security* – Microsoft Press – 2003.

[11] STANGER, J.; LANE, P., T., *Hack Proofing, Linux: A Guide to Open Source Security* – Syngress Publishing, Inc - 2001

[12] The Hackers Layer Group, *The Hackers Layer Handbook* – Version 1.0

[13] REHMAN, R., U., *Intrusion Detection Systems with Snort, Advanced IDS Techniques Using Snort, Apache, MySQL, PHP and ACID* – Prentice Hall – 2003.

[14] STALLINGS, W., *Cryptography and Network Security, Principles and Practice* – Second Edition – Prentice Hall – 1999.

[15] NORTHCUTT, S.; NOVACK, J.; MCLACHLAN, D., *Segurança e Prevenção em Redes* – trad. Marcos Vieira – ed. Berkley – 2001.

[16] WIDROW, B.; LEHR, M., *30 Years of Adaptative Neural Networks: Perceptron, Madaline and Backpropagations* – IEEE – 1990.

[17] KROSE, B.; SMAGT, P., V., D., *An Introduction to Neural Networks* – eight edition – University of Amsterdam – 1996.

[18] KOVACS, Z., L., *Redes Neurais Artificiais, Fundamentos e Aplicações* – 3 ed. - Livraria Física – 1996.

[19] PRINCIPE, J., C., *Artificial Neural Networks, The Electrical Engineering Handbook* – CRC Press LLC – 2000.

[20] JAIN, A., K.; MAO, J., *Artificial Neural Networks: A Tutorial* – IEEE – 1996.

[21] ANDERSON, D.; MCNEILL, G., *Artificial Neural Networks Technology* – Rome Laboratory – 1992.

[22] BAUCHSPIESS, A., *Introdução aos Sistemas Inteligentes, Aplicações em Engenharia de Redes Neurais Artificiais, Lógica Fuzzy e Sistemas Neuro-Fuzzy* – Departamento de Engenharia Elétrica, UnB – 2004.

[23] FAUSETT, L., *Fundamentals of Neural Networks, Architectures, Algorithms and Applications* – Prentice Hall – 1994.
[24] YOUSEFIZADEH, H.; ZILOUCHIAN, A., *Neural Network Architectures* – CRC Press LLC – 2001.
[25] VEELENTURF, L., P., J., *Analysis and Applications of Artificial Neural Networks* – Prentice Hall - 1995.
[26] NEELAKANTA, P., S.; DEGROFF, D., *Neural Network Modeling:Statistical Mechanics and Cybernetic Perspectives* – CRC Press LLC – 1994.
[27] LAZAREVIC, A.; ERTOZ, L.; KUMAR, V.; OZGUR, A.; SRIVASTAVA, J., *A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection* – Computer Science Department, University of Minnesota – 2003.
[28] SANTOS, R., B.; CAMINHAS, W., M.; ERRICO, L., *Detecção de intrusos: Uma abordagem usando redes neurais* – CPDEE, UFMG.
[29] GHOSH, A., K.; MICHAEL, C.; SCHATZ, M., *A Real-Time Intrusion Detection System Based on Learning Program Behavior* – Reliable Software Technology.
[30] CANNADY, J., *Artificial Neural Networks for Misuse Detection* – School of Computer and Information Sciences, Nova Southeastern University – 1998.
[31] RYAN, J.; LIN, M.; MIIKKULAINEN, R., *Intrusion Detection with Neural Networks* – University of Texas at Austin – 1998.
[32] RHODES, B., C.; MAHAFFEY, J., A.; CANNADY, J., D., *Multiple Self-Organizing Maps for Intrusion Detection* – Georgia Institute of Technology – 2000.
[33] CANNADY, J., *The Application of Artificial Neural Networks to Misuse Detection: Inicial Results* – Georgia Institute of Technology – 1998.
[34] LIPPMANN, R., P.; CUNNINGHAM, R., K., *Improving Intrusion Detection Performance Using Keyword Selection and Neural Networks* – MIT Lincoln Laboratory.
[35] MUKKAMALA, S.; JANOSKI, G.; SUNG, A., *Intrusion Detection: Support Vector Machines and Neural Networks* – New mexico Institute of Mining and Technology – 2002.
[36] SILVA, R., M.; MAIA, M., A., G., M., *Redes Neurais Artificiais Aplicadas à Detecção de Intrusos em Redes TCP/IP* – PUC-RJ.
[37] BOMBONATO, F.; COELHO, F., E., S., *Beholder – Utilizando Redes Neurais MPL na Detecção de Intrusos* – Universidade Católica de Brasília.
[38] PLANQUART, J., P., *Application of Neural Networks to Intrusion Detection* – SANS Institute – 2001.
[39] GIRARDIN, L., *An eye on Network Intruder-Administrator Shootouts* – UBS, Ubilab – 1999.
[40] RAMADAS, M.; OSTERMANN, S.; TJADEN, B., *Detecting Anomalous Network Traffic with Self-Organizing Maps* – Ohio University – 2003.
[41] LEI, J., Z.; GHORBANI, A., *Network Intrusion Detection Using an Improved Competitive learning Neural Network* – University of New Brunswick, Canada.
[42] JIRAPUMMIN, C.; WATTANAPONGSAKORN, N.; KANTHAMANON, P., *Hybrid Neural Networks for Intrusion Detection System* – King Mongkut's University of Technology Thonburi – 2003.
[43] LIPMANN, R.; HAINES, J., W.; FRIED, D., J.; KORBA, J.; DAS, K., *The 1999 DARPA Off-Line Intrusion Detection Evaluation* – MIT Lincoln Laboratory – 1999.

[44] HONEYNET PROJECT, *Know Your Enemy* – Addison-Wesley – 2002.
[45] MELO, L., P., *Honeynets* – Universidade Católica de Brasília – 2004.
[46] PERENS, B.; RUDOLPH, S.; GROBMAN, I.; TREACY, J.; DI CARLO, A., *Installing Debian GNU/Linux 3.0 For SPARC* – 2002.
[47] PEIXOTO, J., F., *Honeynet, um Ambiente para Análise de Intrusão* – IBILCE, Universidade Paulista.
[48] www.mhhe.com/socscience/intro/ibank/set1.htm
[49] www.answers.com/topic/action-potential
[50] http://www.ccwu.edu/Thesis_Moynihan/Chapter3_files/image006.gif
[51] http://www.ii.metu.edu.tr/~ion526/demo/chapter1/section1.2/images/fig18.gif
[52] http://www.kovan.ceng.metu.edu.tr/~erol/publications/html/Sah94MS/l2-img10.gif
[53] http://www.ph.tn.tudelft.nl/Research/neural/feature_extraction/papers/thesis/img215.gif
[54] http://www.ph.tn.tudelft.nl/Research/neural/feature_extraction/papers/thesis/img218.gif
[55] http://www.ift.uib.no/~antonych/field.jpg
[56] http://www.lohninger.com/helpcsuite/img/kohonen1.gif
[57] http://www.honeynet.org/
[58] http://www.honeynet.org.br/
[59] http://www.honeynet.org/papers/honeynet/
[60] http://www.honeynet.org.es/papers/honeywall/
[61] http://www.linuxsecure.de/index.php?action=90
[62] http://snort-inline.sourceforge.net/
[63] http://www.snort.org
[64] http://ebtables.sourceforge.net/
[65] http://bridge.sourceforge.net/download.html
[66] http://www.tcpdump.org/
[67] http://www.ethereal.com
[68] http://www.debian.org/
[69] http://www.sunfreeware.com/
[70] http://www.netfilter.org/
[71] http://aurelio.net/shell/canivete.html
[72] http://www.net-security.org/article.php?id=568
[73] http://www.phrack.org/fakes/p62/p62-0x07.txt
[74] http://www.linuxsecurity.com.br/sections.php?op=imprime&artid=5
[75] http://www.insecure.org/tools.html
[76] http://www.ssh.com/
[77] http://www.sans.org/
[78] http://www.dnsstuff.com/
[79] http://www.cert.br/
[80] http://www.iss.net/
[81] http://www.cis.hut.fi/research/som_pak/som_doc.txt
[82] http://www.cis.hut.fi/research/som-research/nnrc-programs.shtml
[83] http://www.easynn.com/
[84] http://www.cet.nau.edu/~mc8/Socket/Tutorials/section1.html
[85] http://reactor-core.org/libpcap-tutorial.html
[86] http://www.tcpdump.org/pcap.htm

[87] http://www.securityfocus.com/
[88] http://pt.wikipedia.org/
[89] http://www.rfc-editor.org/
[90] http://www.cert.org/
[91] http://www.ll.mit.edu/IST/ideval/
[92] http://www.subnet-calculator.com/