MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-9625-TDI/845

ANÁLISE DE ESTADO DE TRÁFEGO DE REDES TCP/IP PARA APLICAÇÃO EM DETECÇÃO DE INTRUSÃO

Marcelo Henrique Peixoto Caetano Chaves

Dissertação de Mestrado em Computação Aplicada, orientada pelo Prof. Dr. Antonio Montes Filho, aprovada em 23/09/2002.

INPE
São José dos Campos
2003

681.3.24

CHAVES, M. H. P. C.
Análise de estado de tráfego de redes TCP/IP para aplicação em detecção de intrusão / M. H. P. C. Chaves.- São José dos Campos: INPE, 2002.
172p. – (INPE-9625-TDI/845).

1.Redes TCP/IP. 2.Segurança de Rede. 3.Sistemas de detecção de intrusão. I.Título.

Aprovado pela Banca Examinadora em cumprimento a requisito exigido para a obtenção do Título de **Mestre** em **Computação Aplicada.**

Dr. Ulisses Thadeu Vieira Guedes

Presidente/INPE/SJCampos-SP

Dr. Antonio Montes Filho

Orientador/INPE/SJCampos-SP

Dr. Adriano Mauro Cansian

Membro da Banca
Convidado UNESP/S.J. Rio Preto-SP

Candidato: Marcelo Henrique Peixoto Caetano Chaves

São José dos Campos, 23 de setembro de 2002.

"Se você é responsável pela segurança de uma organização,
mas não tem autoridade para estabelecer regras ou punir infratores,
seu papel é levar a culpa quando algo grande der errado".

EUGENE SPAFFORD

Aos meus pais, por tudo.

AGRADECIMENTOS

Agradeço a DEUS, por estar sempre presente e por ter iluminado o meu caminho.

Aos meus pais Carlos e Terezinha, pelo amor incondicional, por todo o carinho, dedicação, apoio e pelo maior presente, a vida.

À minha eterna e amada Janaína, por todo o seu amor, lealdade, companheirismo, carinho, dedicação, por estar ao meu lado, mesmo em tempos de *chuva*, e por todo o apoio na fase mais crítica deste trabalho.

Ao meu irmão Carlos, por estar sempre ao meu lado e pela amizade cada vez mais forte.

Ao meu orientador Prof. Dr. Antonio Montes, pelos ensinamentos e experiências compartilhadas, pela orientação e apoio e, principalmente, pelo imensurável auxílio no momento mais crítico.

Aos amigos da República *Heróis da Resistência*, Gilberto, Claudio, Bruno e Glauco, por todo o companheirismo e pela grande amizade, pelas experiências adquiridas e discussões infindáveis mas proveitosas, pelos bons momentos e também pelos momentos difíceis, e aos três primeiros pelo grande auxílio durante todo o desenvolvimento deste trabalho.

Ao grande amigo Ivan, por seu companheirismo, lealdade e amizade incondicional, mesmo nos momentos em que estive mais distante.

Aos inseparáveis amigos Reinaldo e Adriano, que apesar de às vezes distantes, sempre participaram de momentos importantes da minha vida.

À amiga Alessandra, por todo o auxílio enquanto trabalhava na biblioteca do INPE.

Aos amigos Klaus e Cristine, pelos ensinamentos, experiências compartilhadas e, principalmente, pelos bons conselhos e contribuições dadas a este trabalho.

Ao amigo Amândio, por toda a compreensão e apoio nos momentos mais difíceis.

Aos novos amigos Lúcio e Gustavo, pelas contribuições dadas a este trabalho.

Ao Bruno, pelo estilo do LaTeX, desenvolvido segundo as normas de publicação do INPE.

Ao Instituto Nacional de Pesquisas Espaciais (INPE) e, em especial, ao Laboratório de Matemática e Computação Aplicada (LAC), pela oportunidade de estudo, trabalho e utilização de suas instalações, e aos professores e colegas do INPE, pelos ensinamentos e conhecimento compartilhado.

À Fundação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo auxílio financeiro em quase dois anos de bolsa de mestrado.

E finalmente, a todas as pessoas que de alguma forma contribuíram para o cumprimento de mais esta etapa da minha vida.

RESUMO

Este trabalho apresenta o desenvolvimento de uma metodologia de reconstrução de sessões para o tráfego de redes TCP/IP. Esta metodologia baseia-se em um modelo, gerado a partir de dados extraídos do tráfego de rede, que permite reconstruir e rastrear o estado das sessões, utilizando apenas o cabeçalho dos pacotes. Através da extrapolação do conceito de "sessão", esta modelagem permite não só reconstruir e rastrear o estado das sessões TCP, mas também reconstruir sessões ICMP e UDP. O modelo é, então, utilizado como base para o desenvolvimento do *Sistema de Reconstrução de Sessões TCP/IP - RECON* - para ser usado em atividades associadas à detecção de intrusão. Esta abordagem possibilita a redução do número de falso-positivos e falso-negativos, visto que o estado e todo o histórico de pacotes que compõem uma sessão podem ser utilizados na tomada de decisões, em contrapartida àquelas que tomam decisões avaliando pacote a pacote, isoladamente. Ela também permite correlacionar informações de um conjunto de sessões na identificação de atividades hostis, que não podem ser observadas em uma única sessão. O sistema faz uso de um sensor, posicionado adequadamente para coleta de pacotes do tráfego, que armazena os dados em arquivos periodicamente. Estes arquivos são transferidos para uma estação de análise, onde o *RECON* é executado. Uma característica desta metodologia é que ela permite tratar uma quantidade relativamente grande de informações, pois utiliza um número reduzido de dados por pacote, correspondentes apenas aos seus cabeçalhos. O sistema desenvolvido pode também funcionar como uma ferramenta de suporte, aplicado não só em atividades de detecção de intrusões, mas também a outras atividades, tais como o gerenciamento, monitoramento e estudo do comportamento do tráfego de redes TCP/IP. Finalmente, são relatados os resultados obtidos com o sistema desenvolvido, mostrando a eficiência, capacidade e possibilidades de sua aplicação.

STATEFUL ANALYSIS OF TCP/IP NETWORK TRAFFIC FOR INTRUSION DETECTION APPLICATION

ABSTRACT

In this work, the development of a session reconstruction methodology for TCP/IP network traffic is presented. The methodology is model-based and uses netwok traffic extracted data for the reconstruction and tracking of sessions state using only packet headers. By extrapolating the concept of session this modeling allows not just the reconstruction and tracking of TCP sessions states, but also the reconstruction of ICMP and UDP sessions. The model has been designed to support the development of the TCP/IP Session's Reconstruction System - RECON - for use in intrusion detection. Since the state and packets history associated with a session can be used to decide if the traffic is part of an attack, differently from other methods that based decisions on a packet by packet exam, the use of this methodology can reduce the number of false-positives and false-negatives. It also possible to correlate informations from a set of sessions for the identification of hostile activities that can not be observed in an isolated session. The system makes use of a sensor that is appropriatedly located to capture packets from network traffic and to store captured data in files regularly. This files are transfered to an analysis station, where RECON runs. One feature of this methodology is the ability to treat large amount of traffic, due to the use of a reduced amount of data associated with the packet headers. The developed system can also operate as a support tool, applied not just to intrusion detection but also to the management, monitoring and testing of TCP/IP network traffic. Finally, the results obtained with the developed system are reported and showing the efficiency, capability and possible applications.

SUMÁRIO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE SIGLAS E ABREVIATURAS

API	–	*Application Programming Interface*
ARP	–	*Address Resolution Protocol*
BPF	–	*BSD Packet Filtering*
BSD	–	*Berkeley Systems Development*
CVE	–	*Common Vulnerabilities and Exposures*
CERT	–	*Computer Emergency Response Team*
CERT/CC	–	*Computer Emergency Response Team/Coordinate Center*
DNS	–	*Domain Name System*
FTP	–	*File Transfer Protocol*
HTTP	–	*Hypertext Transfer Protocol*
IANA	–	*Internet Assigned Numbers Authority*
IDS	–	*Intrusion Detection System*
IDSs	–	*Intrusion Detection Systems*
ICMP	–	*Internet Control Message Protocol*
IMAP	–	*Internet Message Access Protocol*
IP	–	*Internet Protocol*
I/O	–	*Input/Output*
NFS	–	*Network File System*
NTP	–	*Network Time Protocol*
POP	–	*Post Office Protocol*
RARP	–	*Reverse Address Resolution Protocol*
RFC	–	*Request for Comments*
RPC	–	*Remote Procedure Call*
SDI	–	Sistema de Detecção de Intrusão
SDIs	–	Sistemas de Detecção de Intrusão
SDIR	–	Sistema de Detecção de Intrusão de Rede
SDIRs	–	Sistemas de Detecção de Intrusão de Rede
SMTP	–	*Simple Mail Transfer Protocol*
SNMP	–	*Simple Network Management Protocol*
SSH	–	*Secure Shell*
TCP	–	*Transmission Control Protocol*
TCP/IP	–	*Transmission Control Protocol/Internet Protocol*
UDP	–	*User Datagram Protocol*

CAPÍTULO 1

INTRODUÇÃO

Este capítulo apresenta uma breve discussão sobre os riscos associados às redes de computadores, com relação à segurança, e sobre algumas características associadas à sistemas de informações seguros. Uma breve discussão sobre a evolução dos sistemas de detecção de intrusão é apresentada e, ao final, são abordados os propósitos e motivações para a realização desta pesquisa.

1.1 RISCOS RELACIONADOS À REDE MUNDIAL E À REDE BR

O alto nível de conectividade das redes de computadores que compõem a Internet e o avanço no desenvolvimento de tecnologias relacionadas a redes de computadores, vêm proporcionando a sensação de que acessar um computador do outro lado do mundo não é mais difícil do que acessar um computador na sala ao lado. Com isso, as organizações passaram a confiar e utilizar tecnologias de Internet como meio de comunicação de dados e para o comércio eletrônico, o que despertou a preocupação com a segurança de seus sistemas de informação. Pessoas mal intencionadas poderiam estar utilizando a "grande rede" para desferir ataques contra os computadores e sistemas destas organizações.

Estes ataques poderiam causar grandes danos às organizações, portanto algumas destas passaram a investigar a possibilidade de estarem sendo atacadas e quais vulnerabilidades estavam sendo utilizadas para este propósito. Tem-se observado a crescente preocupação destas organizações para com as questões relacionadas à segurança de seus sistemas, de modo que estas têm dispendido grandes esforços na construção e implantação de sistemas de monitoramento e análise, na busca de uma melhor capacitação para reagir a possíveis incidentes de segurança.

A questão é que o aperfeiçoamento da segurança dos sistemas de informação de uma organização, normalmente, é uma tarefa árdua e requer comprometimento e responsabilidade por parte dos profissionais responsáveis. Além da constante e crescente preocupação com ameaças externas, administradores de sistemas e analistas de segurança estão saturados com uma série de problemas relacionados às redes e sistemas internos à organização. Estes problemas envolvem o constante esforço para o cumprimento da política de segurança da organização, quando esta existe, usuários utilizando senhas "fracas" e disponibilizando suas senhas para pessoas não-autorizadas, abusos de privilégios e acesso não-autorizado a redes e sistemas da organização, e invasões partindo da rede interna. Este último impacta diretamente em um dos principais prejuízos sofridos por uma organização: as perdas financeiras.

Mesmo quando nenhum destes problemas está ocorrendo, administradores dispendem horas ou dias de trabalho atualizando sistemas operacionais e aplicações associadas a serviços de rede disponibilizados na organização. Apesar dos desenvolvedores terem conhecimento dos problemas gerados por falhas na implementação de sistemas computacionais, cuidados para evitar estes tipos de problemas ainda não foram incorporados nos processos de desenvolvimento de *software*. Em grande parte dos casos, estas falhas são diretamente responsáveis e correspondem a vulnerabilidades de segurança.

Em 1999, o MITRE *Corporation*[1] iniciou o desenvolvimento do CVE[2] (*Common Vulnerabilities and Exposures*), um dicionário ou lista padronizada de nomes para todas as vulnerabilidades de segurança publicamente conhecidas. A grande maioria das entradas desta lista refere-se a falhas na implementação de aplicações de rede e sistemas operacionais, que podem, em muitos casos, ser utilizadas no comprometimento de sistemas tanto a partir da rede local quanto a partir de uma rede externa ou remota. Em sua última versão, do dia 25/06/2002, estavam catalogadas 2223 vulnerabilidades oficiais e 2613 vulnerabilidades candidatas.

No ano 2000, o *Computer Security Institute* (CSI[3]) anunciou os resultados do quinto levantamento anual de crimes de computadores e segurança. Este levantamento é conduzido pelo CSI, em conjunto com o *San Francisco Federal Bureau of Investigation's* (FBI) *Computer Intrusion Squad*. O principal objetivo é elevar o nível de preocupação com segurança, e determinar características dos crimes de computadores nos Estados Unidos. Dentre os resultados, foram apontadas as seguintes questões: 90% dos que participaram do levantamento, tais como grandes corporações e agências governamentais, detectaram violações de segurança dentro dos últimos 12 meses; 70% relataram uma variedade de violações de segurança mais sérias, como roubo de informações proprietárias, fraudes financeiras, intrusão de sistemas (partindo de uma rede externa), ataques de negação de serviço (*denial-of-service*) e sabotagem de dados ou redes; 74% notificaram perdas financeiras, devido a violações de segurança, e 42% (723 inspecionados) estavam aptos a quantificar suas perdas financeiras, que giraram em torno de US$265,589,940 (a média anual total dos três anos anteriores foi US$120,240,180). Outros resultados mostraram que ameaças de crimes de computadores, visando grandes corporações e agências governamentais vêm tanto de dentro quanto de fora dos sistemas, confirmando as tendências apontadas em anos anteriores.

Em 2002, o *AusCERT*, juntamente ao *Deloitte Touche* e a *NSW Police* realizaram o pri-

[1]Página Web em: `http://www.mitre.org`.
[2]Página Web em: `http://cve.mitre.org`
[3]Página Web em: `http://www.gocsi.com`.

meiro levantamento anual de crimes de computadores e segurança da Austrália. Dentre os resultados, os seguintes apontamentos foram realizados: 67% das organizações que participaram do levantamento sofreram um incidente de segurança em 2002 e 35% destas vivenciaram 6 ou mais incidentes; pela primeira vez na Austrália a ameaça crescente de ataques externos superou a de ataques internos; 89% das organizações australianas que sofreram incidentes de segurança foram atacadas remotamente, enquanto que menos de 65% foram atacadas internamente; 75 organizações puderam identificar perdas financeiras, mas apenas 60 puderam quantificá-las, sendo que estas giraram em torno de US$5,781,300.

O *ALLDAS.org*[4], *site* que mantém uma base de dados de páginas Web que tiveram seu conteúdo alterado (*defacement*) como consequência de uma invasão, já mantém em seus arquivos 35390 ocorrências, sendo que 10430 correspondem a páginas de organizações comerciais (*.com*).

Um outro aspecto que deve ser levando em conta é o desenvolvimento de ferramentas, utilizadas para desferir ataques contra estas organizações. Inicialmente, o processo envolvido com a execução de um ataque necessitava de um alto nível de conhecimento técnico, por parte do intruso, e os recursos disponíveis para o desenvolvimento deste processo eram bem limitados. Mas com o passar do tempo, os ataques vêm se tornando mais sofisticados, e exigem um nível de conhecimento técnico cada vez menor, por parte do intrusos. O grande responsável por esta questão é o próprio advento da Internet. As informações estão disponíveis, e desta forma, ferramentas desenvolvidas para a realização de ataques são difundidas pela rede muito rapidamente. O maior trabalho de um intruso é encontrar o repositório, contendo a ferramenta certa, que realiza o ataque desejado.

A Figura (1.1) apresenta um gráfico, que relaciona o nível de conhecimento técnico dos intrusos, com a sofisticação dos ataques, no decorrer dos anos.

No ano de 2000, 21.756 incidentes de segurança foram reportados ao *Carnegie Mellon University* CERT/CC[5] (*Computer Emergency Response Team/Coordination Center*). Este número cresceu mais de duas vezes no ano de 2001, totalizando 52.658 incidentes. E até o presente momento, no ano de 2002, este número já alcança 43.136 incidentes reportados. De 1988 a 2002, o número de incidentes totaliza 143.505 ocorrências.

O *Brazilian* CERT, também conhecido como *NIC BR Security Office* (NBSO[6]), é a organização responsável por receber, analisar e responder a incidentes de segurança em

[4]Página Web em: `http://www.alldas.org`.
[5]Página Web e estatísticas relacionadas encontradas em: `http://www.cert.org`.
[6]Página Web e estatísticas relacionadas encontradas em: `http://www.nbso.nic.br`.

computadores envolvendo redes conectadas a Internet brasileira. No ano de 1999, 3.107 ataques foram reportados ao NBSO, e este número cresceu para 5.997 ocorrências em 2000. No primeiro trimestre de 2001, 3138 ataques foram reportados, onde: 0,22% constituíram tentativas de obter/atualizar mapas de DNS; 15,42% ataques ao usuário final (direcionados a máquinas pessoais); 0.83% negação de serviços (*denial-of-service*); 1.72% invasões; 3,7% ataques a servidores *Web*; 77,66% varreduras, e 0,45% fraudes. Os totais relacionados a cada ano mostram uma tendência crescente no número de ataques, de ano para ano.

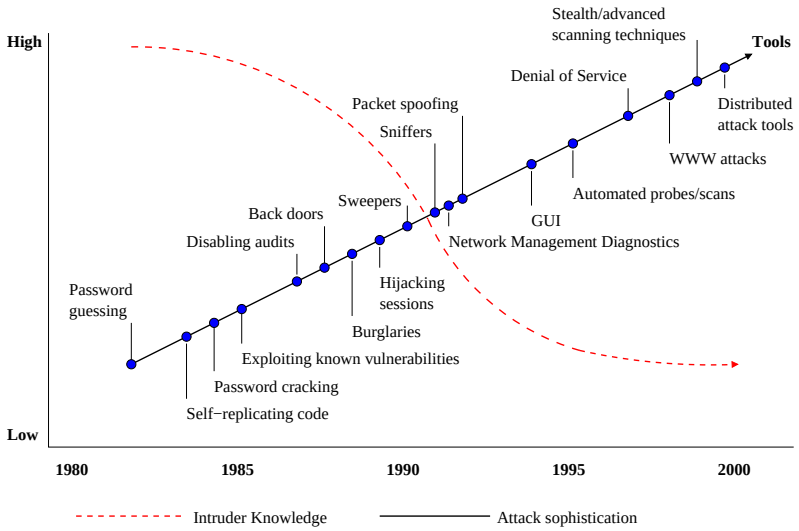

FIGURA 1.1 – Sofisticação dos ataques verso conhecimento técnico dos intrusos.
FONTE: adaptada de McHugh et al. (2000, p. 43).

A intenção aqui é ressaltar que as organizações estão expostas a estes riscos. E também mostrar a importância da segurança de sistemas de informações, e mais especificamente, a importância do uso de sistemas que possam detectar violações de segurança.

1.2 SEGURANÇA DE SISTEMAS DE INFORMAÇÃO

De forma generalizada, a literatura especializada na área diz que a segurança de um sistema de informação e das redes que o compõem não pode ser garantida por um único mecanismo de defesa. Muitos acreditam que a utilização de um *Firewall* é suficiente para proteger uma rede contra quaisquer tipos de ameaças. Mas o nível de segurança de uma rede está na verdade associado à maneira com que mecanismos de segurança são empregados. Não existe um sistema genérico e que concentre todas as soluções para os problemas da área e, portanto, segurança deve ser implementada em camadas e não concentrada em um único ponto, supostamente infalível.

Esta abordagem, utilizada para garantir um bom nível de segurança, está relacionada primeiramente ao estabelecimento de uma política de segurança, onde são definidos comportamentos que são permitidos ou proibidos, ferramentas e procedimentos necessários a organização, e que disseminem o consenso entre gerentes e administradores. Uma boa política de segurança, atrelada a definição de uma política de senhas para os usuários do sistema, a preocupação com a segurança de *hosts*, a utilização de *Firewall*, dividindo e controlando o acesso entre rede interna e externa, e a utilização de SDIs (Sistemas de Detecção de Intrusão) constituem um bom exemplo de segurança em camadas, e que asseguram um bom nível de segurança para a organização.

Dentro desta abordagem, o *Firewall* constitui uma das ferramentas mais utilizadas atualmente, sendo que o tipo mais empregado é conhecido como filtro de pacotes. Basicamente, esses sistemas de filtragem examinam pacotes entrando ou saindo da rede, através do uso de um conjunto de regras fixas, que determinam se os pacotes serão ou não aceitos.

Inicialmente, os sistemas de filtragem de pacotes tomavam decisões, apenas analisando informações, tais como endereços IP de origem e destino, protocolo e portas, pacote por pacote, no momento em que entravam ou saiam do *Firewall*. Mesmo dispositivos um pouco mais sofisticados baseavam-se em informações estáticas. Desta forma, a natureza destes sistemas de filtragem possibilitou que ferramentas fossem desenvolvidas para subverter este mecanismo de defesa.

Para solucionar este problema, foram desenvolvidos filtros de pacotes dinâmicos (*stateful*), que mantém o estado das informações entrando e saindo da rede. Desta forma, estes filtros podem checar se um pacote pertence a uma sessão válida, previamente estabelecida. A grande vantagem dos filtros de pacote dinâmicos é que o tráfego de retorno não precisa ser explicitamente especificado, simplificando assim, o conjunto de regras mantidas pelo *Firewall*. Hartmeier (2002) e Rooij (2002) apresentam boas referências sobre filtragem dinâmica de pacotes.

Sabe-se, entretanto, que *Firewalls* do tipo filtro de pacotes não fornecem uma solução completa para o problema de segurança. Uma de suas limitações consiste no fato de que este não pode bloquear uma tentativa de intrusão ou abuso, que se aproveite de vulnerabilidades de serviços habilitados, para os quais o tráfego é permitido. Daí surge a necessidade de utilização de mecanismos que permitam detectar falhas de segurança, ou seja, detectar intrusões ou tentativas de intrusões. Estes mecanismos são conhecidos como SDIs - Sistemas de Detecção de Intrusão - e desempenham a função de mais uma camada de defesa em um sistema de informação.

1.3 A EVOLUÇÃO DOS SISTEMAS DE DETECÇÃO DE INTRUSÃO

Em 1987, Denning (1987) desenvolveu o primeiro modelo genérico de detecção de intrusão que se tem conhecimento, independente de qualquer sistema em particular, vulnerabilidade de sistema ou tipo de intrusão. Este utilizava dados estatísticos, gerados à partir de registros de auditoria, e baseava-se na hipótese de que violações de segurança poderiam ser detectadas através do monitoramento desses registros, relacionados a padrões anormais de uso do sistema.

A partir daí, métodos de detecção de intrusão foram desenvolvidos e dividiram-se, basicamente, em duas categorias: detecção por anomalia e a detecção por abuso. A primeira baseia-se na idéia de que todo evento intrusivo é necessariamente anômalo, sendo um subconjunto da atividade anômala, e normalmente está associada a técnicas estatísticas. Já a segunda baseia-se na idéia de que ataques podem ser representados na forma de padrões ou assinaturas (sequência de eventos e condições que levam a uma intrusão).

Sistemas de detecção de intrusão foram desenvolvidos (Allen et al., 2000) (Bace e Mell, 2000) baseados nestes dois métodos, e alguns até utilizam os dois métodos simultaneamente. Estes últimos ficaram conhecidos como sistemas híbridos. Estes sistemas serão discutidos em detalhes no Capítulo 4.

Os primeiros SDIs baseavam-se apenas em informações obtidas de um determinado *host*, e em sua linha de evolução, foram rapidamente adaptados para utilizar como base informações coletadas do tráfego de rede. Estes sistemas são conhecidos como SDIs *network-based*.

Nesta nova abordagem, que é amplamente utilizada na atualidade, os SDIs de maior destaque utilizam detecção por reconhecimento de padrões, onde ataques são codificados em assinaturas (padrões), de forma que estas possam ser identificadas pelo mecanismo de detecção. Até há algum tempo, estes sistemas, da mesma forma que os filtros de pacotes estáticos, baseavam-se na análise de informações avaliadas pacote por pacote, à medida que estes eram processados. Caso as informações utilizadas no reconhecimento de uma assinatura de ataque estivessem divididas em mais de um pacote, muitas vezes o SDI falhava na detecção.

A natureza estática destes SDIs permitiu o desenvolvimento de ferramentas, tais como o *stick* (Giovanni, 2002) e o *fragrouter*[7] (Ptacek e Newsham, 1998), cujo principal objetivo era subverter os mecanismos de detecção de intrusão. A base de regras do próprio SDI, contendo as assinaturas de ataques, era utilizada para gerar o tráfego de rede com os padrões intrusivos. Então, o SDI gerava uma infinidade de alarmes que, muitas vezes,

[7] *man page*: `http://www.gnusec.com/resource/security-stuff/IDS/fragrouter.html`

caracterizavam um ataque de negação de serviços (*denial-of-service*) para o próprio SDI. Além disso, um ataque real poderia ser inserido neste tráfego, de modo que este não seria detectado, em meio a tantos alarmes.

Atualmente, desenvolvedores de sistemas de detecção de intrusão têm se preocupado com estas questões e alguns SDIs, na maioria comerciais, já implementam funcionalidades relacionadas a reconstrução e a avaliação do estado das sessões. Estes são conhecidos como SDIs *stateful*. Outros sistemas, um pouco mais sofisticados, já fazem a análise de protocolos de aplicação, mas utilizam como base as funcionalidades mencionadas anteriormente.

O grande problema é que estas soluções são proprietárias, tendo seu código "fechado" e um alto custo atrelado. Outra questão está relacionada a quantidade de recursos necessários a implementação destas soluções. A maioria destes sistemas realizam a detecção de intrusões em tempo real e utilizam o conteúdo dos pacotes nesta tarefa. Portanto, a quantidade de dados a ser avaliada e mantida é normalmente muito grande. Isso acaba tendo uma forte implicação no desempenho dos sistemas de detecção modernos, considerando em particular o grande aumento de banda disponível nas redes atuais (Mueller e Shipley, 2001).

Outro aspecto que soluções proprietárias endereçam e o da captura de pacotes em redes de alta velocidade. Vêm sendo desenvolvidos *drivers* proprietários para interfaces de rede que operam com alta eficiência, evitando, dessa maneira, as limitações das bibliotecas de captura publicamente disponíveis (*libpcap*), que começam a perder pacotes para taxas de transmissão acima de algumas dezenas de megabits por segundo.

1.4 PROPÓSITOS, MOTIVAÇÕES E ESBOÇO GERAL DESTE TRABALHO

O que se pretende com este trabalho é propor, desenvolver e testar uma metodologia de reconstrução de sessões para o tráfego de redes TCP/IP. Esta metodologia baseia-se em um modelo, gerado à partir de dados extraídos do tráfego de rede, que permite reconstruir e rastrear o estado das sessões, utilizando apenas o cabeçalho dos pacotes. Este modelo é, então, utilizado como base para o desenvolvimento do *RECON* - Sistema de Reconstrução de Sessões TCP/IP - a ser utilizado em aplicações associadas à detecção de intrusão, e até mesmo em tarefas de monitoramento e estudo do comportamento do tráfego de redes.

A principal motivação deste trabalho foi o desenvolvimento de recursos para a superação de algumas das fragilidades dos atuais sistemas de detecção de intrusão. Assim, buscou-se fazer uso de uma metodologia de reconstrução de sessões, que possibilita a redução do número de falso-positivos e falso-negativos, visto que o estado e todo o histórico de pacotes

que compõem uma sessão podem ser utilizados na tomada de decisões, em contrapartida àquelas que tomam decisões avaliando pacote a pacote. Além disso, a abordagem onde apenas os cabeçalhos dos pacotes são analisados permite uma considerável redução na quantidade de dados do fluxo de rede a ser capturada e avaliada. Desta forma, espera-se um melhor desempenho e eficiência de aplicações baseadas nestas características. Deve-se considerar também que esta metodologia permite a correlação de sessões, que isoladamente não representariam uma atividade hostil, como por exemplo a utilização de uma máquina intermediária para o lançamento de ataques ou a abertura de uma porta dos fundos em sistemas comprometidos. E por fim, o sistema desenvolvido pode funcionar como uma ferramenta de suporte, de modo que este poderá ser aplicado não só à detecção de intrusões, mas também a outras atividades, tais como o monitoramento e a geração de estatísticas relacionadas ao tráfego de redes.

O restante desta dissertação está organizado em seis capítulos. O **Capítulo 2** faz uma breve revisão teórica sobre a tecnologia TCP/IP, onde é apresentada a pilha de protocolos TCP/IP, com uma descrição de cada camada, seus respectivos protocolos, e algumas aplicações. O **Capítulo 3** apresenta uma revisão sobre captura de pacotes, baseada em uma arquitetura específica. É apresentada também a biblioteca que implementa esta arquitetura, bem como uma aplicação que faz uso dessa biblioteca. O **Capítulo 4** discute conceitos e princípios relacionados a segurança de sistemas de informação e detecção de intrusão. Nele são abordadas as principais categorias e métodos de detecção de intrusão, além de apresentar alguns exemplos de sistemas de detecção de intrusão projetados e implementados. O **Capítulo 5** propõe um modelo para a reconstrução de sessões TCP/IP e apresenta de forma detalhada a implementação do *RECON*, sistema de reconstrução de Sessões TCP/IP baseado na modelagem proposta. O **Capítulo 6** apresenta e discute os resultados obtidos com o sistema desenvolvido. E por último, o **Capítulo 7** apresenta as conclusões e algumas propostas para a continuação desta pesquisa.

CAPÍTULO 2

TECNOLOGIA TCP/IP

Este capítulo mostra uma breve revisão teórica sobre a tecnologia TCP/IP, assunto fundamental para o entendimento do funcionamento de sistemas de detecção de intrusão, atualmente disponíveis e utilizados nas redes de computadores conectadas à Internet. É apresentada a pilha de protocolos TCP/IP, com uma breve descrição de cada camada e de seus respectivos protocolos, bem como algumas aplicações dentre as várias que merecem atenção com relação a segurança de redes de computadores.

2.1 PILHA DE PROTOCOLOS TCP/IP

A pilha de protocolos TCP/IP é organizada em quatro níveis ou camadas conceituais, construídas sobre uma quinta camada (Comer, 2000), correspondente ao nível físico ou de hardware, como mostra a Figura (2.1).

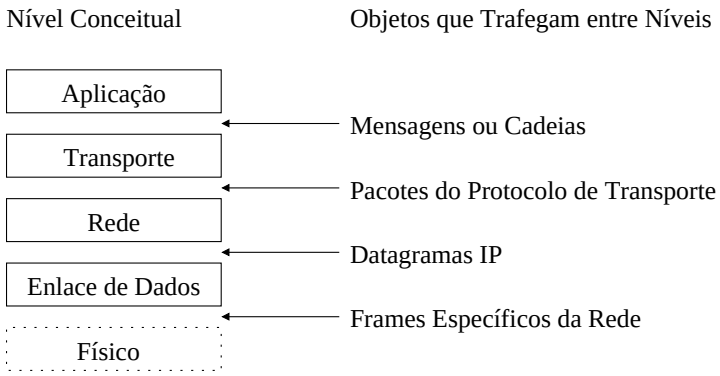

FIGURA 2.1 – Os 4 níveis conceituais da tecnologia TCP/IP e os objetos que trafegam entre níveis.
FONTE: adaptada de Comer (2000, p. 184).

As funções das camada da pilha de protocolos são apresentadas como se segue.

Camada de Aplicação: corresponde ao nível mais alto, onde usuários executam aplicações, através da utilização de serviços disponíveis em uma rede TCP/IP. Uma aplicação interage com os protocolos da camada de transporte para enviar ou receber dados. Cada aplicação escolhe o tipo de transporte, que pode ser uma sequência de mensagens individuais ou uma cadeia contínua de bytes.

Camada de Transporte: a finalidade da camada de transporte é prover comunicação entre aplicações, comumente denominada comunicação fim-a-fim. Essa camada é responsável pelo estabelecimento e controle do fluxo de dados entre dois *hosts*. Além disso, pode

prover transporte confiável, de modo a garantir que as informações sejam entregues sem erros e na sequência correta. A cadeia de dados sendo transmitida é dividida em pacotes, que são passados para camada seguinte (rede).

Camada de Rede: a camada de rede trata da comunicação entre *hosts*. Esta aceita uma requisição de envio de pacote vinda da camada de transporte, com a identificação do *host* para onde o pacote deve ser transmitido. Encapsula o pacote em um datagrama IP e preenche o cabeçalho do datagrama com os endereços lógicos de origem e destino, dentre outros dados. Também utiliza um algoritmo de roteamento para determinar se o datagrama deve ser entregue diretamente, ou enviado para um *gateway*. Finalmente, o datagrama é passado para a interface de rede apropriada, para que este possa ser transmitido.

Camada de Enlace de Dados: é a camada de nível mais baixo na pilha de camadas da tecnologia TCP/IP. É responsável por aceitar os datagramas IP, encapsulá-los em *frames*, preencher o cabeçalho de cada *frame* com os endereços físicos de origem e destino, dentre outros dados, e transmití-los para uma rede específica. Na camada de enlace de dados está o *device-driver* da interface de rede, por onde é feita a comunicação com a camada física.

Camada Física: esta camada corresponde ao nível de hardware, ou meio físico, que trata dos sinais eletrônicos. Esta recebe os *frames* da camada de enlace, convertidos em sinais eletrônicos compatíveis com o meio físico, e os conduz até a próxima interface de rede, que pode ser a do *host* destino ou a do *gateway* da rede, caso esta não pertença a rede local.

Stevens (Stevens, 1994) apresenta essas camadas subdivididas em vários protocolos, ilustrados na Figura (2.2). As descrições dos principais protocolos mostrados nesta figura são apresentadas nas sessões seguintes.

2.2 PROTOCOLOS ARP e RARP

O protocolo ARP (*Address Resolution Protocol*) (Plummer, 2002) foi originalmente projetado para interfaces do tipo 10 Mbit *Ethernet*, mas foi generalizado para outros tipos de hardware. O módulo de resolução de endereços, normalmente parte do driver do dispositivo de hardware, recebe um par <tipo-de-protocolo, endereço-destino> e tenta encontrá-lo em uma tabela. Se o par for encontrado, é retornado o endereço do hardware (físico) correspondente, para que o pacote possa ser transmitido. Caso contrário, é normalmente informado que o pacote será descartado. Um exemplo comum na Internet é a utilização do ARP para converter endereços IP, de 32 bits, em endereços *Ethernet*, de 48 bits.

A função do protocolo RARP (*Reverse Address Resolution Protocol*) (Finlayson et al., 2002) é inversa ao ARP, ou seja, converte um endereço de hardware (físico) em um endereço Internet. É usado principalmente por nós de rede do tipo *diskless*, ou seja, que não possuem um endereço Internet armazenado localmente. No momento da inicialização, o RARP é usado para encontrar o endereço Internet correspondente ao endereço de hardware do nó, que é conhecido.

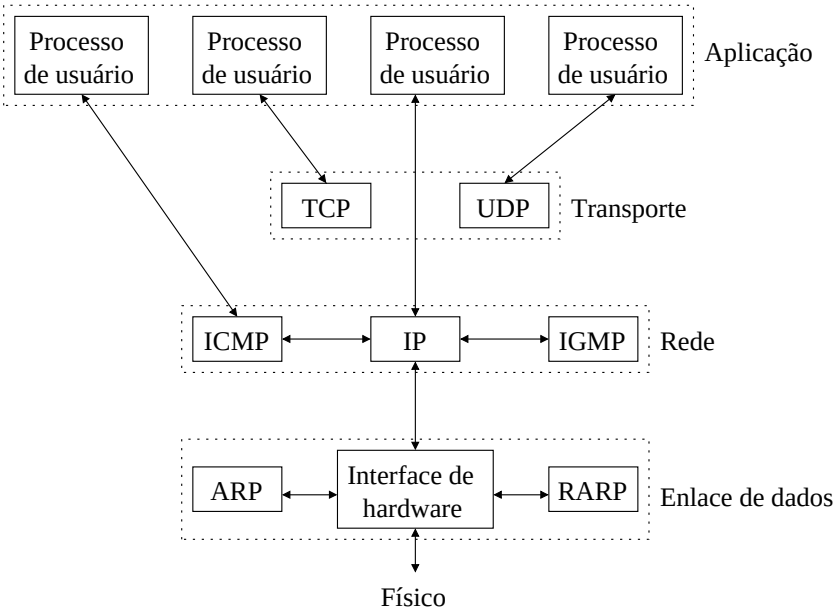

FIGURA 2.2 – Vários protocolos nas diferentes camadas da tecnologia TCP/IP.
FONTE: adaptada de Stevens (1994, p. 6).

2.3 PROTOCOLO IP

O protocolo IP *(Internet Protocol)* (Postel, 2002b) é a base ou suporte para os outros protocolos da pilha TCP/IP, tais como ICMP, UDP e TCP, que são transmitidos em datagramas IP, como mostrado na Figura (2.2).

Datagramas IP podem ser definidos como blocos de dados transmitidos de uma determinada origem para um destino, de forma que origens e destinos são *hosts* identificados por endereços lógicos de tamanho fixo.

Uma característica deste protocolo é a possibilidade de fragmentar e remontar datagramas, de modo que estes possam ser transmitidos entre redes que suportem diferentes tamanhos por bloco de dados. É especificamente projetado para prover as funções necessárias para entregar pacotes de bits (datagramas IP) de uma origem para um destino determinado, em redes interconectadas.

Este é comumente denominado por “serviço de entrega de datagramas não orientado à conexão”, pois não existem mecanismos para aumentar a confiabilidade de dados fim-a-fim, controle de fluxo, sequenciamento, ou outros serviços que normalmente são de responsabilidade dos protocolos de níveis mais altos.

O protocolo IP também incorpora a função de roteamento, isto é, determina se o datagrama deve ser entregue diretamente a seu destino, caso origem e destino pertençam à mesma rede, ou entregue ao *gateway* da rede contendo a origem, para o caso contrário. Se o mecanismo de roteamento decide que o datagrama deve ser enviado ao *gateway*, então este passa a ser responsável por enviar o datagrama para o seu destino.

O IP baseia-se na idéia de entrega de datagramas “sem garantias”, portanto inclui um conjunto de regras que dizem como *hosts* e *gateways* devem processar os datagramas, quando e como uma mensagem de erro deve ser gerada e as condições nas quais datagramas devem ser descartados (Comer, 2000).

A Figura (2.3) mostra o cabeçalho de um datagrama IP, e uma breve descrição, baseada em (Postel, 2002b) e (Stevens, 1994), de alguns de seus campos é apresentada como se segue.

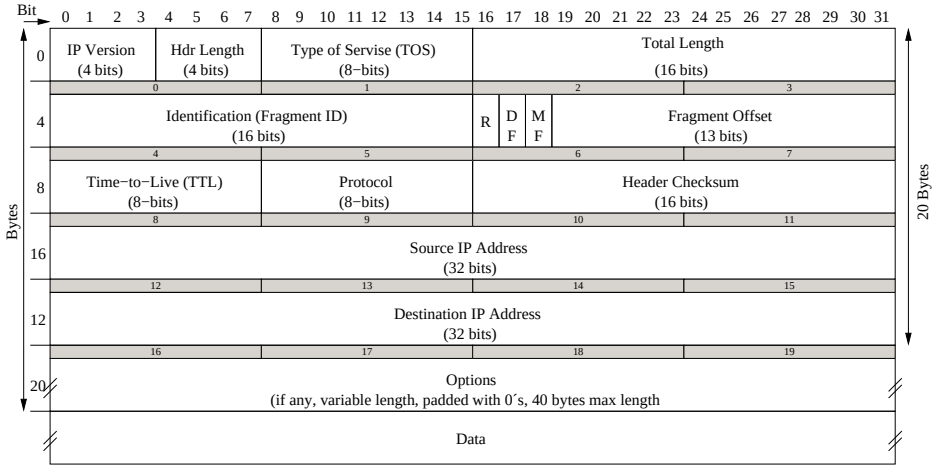

FIGURA 2.3 – Datagrama IP, mostrando os campos no cabeçalho IP.
FONTE: adaptada de Stevens (1994, p. 34).

O campo *version* indica o formato do datagrama, que para este caso corresponde ao IP versão 4 (também conhecido como IPv4). *Hdr length* contém o tamanho do cabeçalho IP, incluindo opções, caso estas ocorram. *Type of Service* provê uma indicação de parâmetros, relacionados a qualidade de serviço desejada, embora a maioria das implementações TCP/IP não suportem esta característica. *Total Length* contém o tamanho total do datagrama, incluindo o cabeçalho IP e os dados propriamente ditos. *Identification* é utilizado

para identificar univocamente cada datagrama enviado por um *host*. O campo *flags* é dividido em três partes: R, DF e MF, sendo R o bit reservado e que normalmente tem o valor 0, DF o bit responsável por dizer que o datagrama não deve ser fragmentado, e MF o bit responsável por informar que o datagrama está fragmentado e existem mais fragmentos. *Offset* indica o posicionamento do fragmento em um dado datagrama a ser remontado. *Time to Live* contém um contador que indica o máximo de roteadores pelos quais um datagrama pode passar. O campo *Protocol* diz qual é o protocolo utilizado pelo próximo nível, ou seja, pelo nível ou camada de transporte. E finalmente, os campos *Source IP Address* e *Destination IP address* contém os endereços lógicos dos *hosts* de origem e destino, respectivamente.

O tamanho do cabeçalho IP é variável, de modo que um datagrama IP sem opções tem 20 bytes. Caso opções estejam presentes no datagrama, este pode ter até 60 bytes. Detalhes sobre o campo *options* podem ser vistos em Postel (2002b).

2.4 PROTOCOLO ICMP

O protocolo ICMP (*Internet Control Message Protocol*) (Postel, 2002a) tem como finalidade relatar erros no processamento de datagramas IP, bem como prover mecanismos de investigação na determinação de características gerais de redes TCP/IP.

Algumas das situações em que mensagens ICMP são utilizadas: datagrama não pode alcançar seu destino, o *buffer* de um *gateway* está cheio e este não pode transmitir o datagrama, o tráfego deve ser direcionado para uma rota mais curta, verificar se um determinado *host* está ativo, dentre outras.

As mensagens ICMP são enviadas em datagramas IP. Embora o ICMP pareça fazer parte do conjunto de protocolos de nível mais alto, este é parte da implementação do protocolo IP. O formato do cabeçalho de uma mensagem ICMP é apresentado na Figura (2.4).

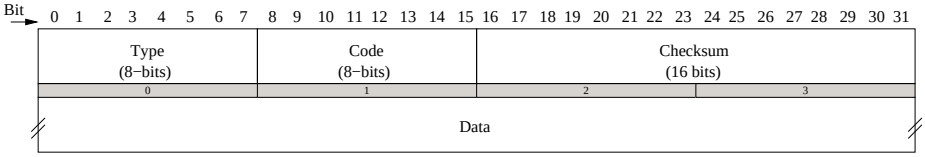

FIGURA 2.4 – Formato da mensagem ICMP.
FONTE: adaptada de Stevens (1994, p. 70).

Mensagens ICMP são identificadas pelo campo *type*, sendo que algumas destas utilizam valores diferentes para o campo *code*. Isto permite que um tipo de mensagem possa especificar diferentes condições, como mostra a tabela A.1.

O ICMP utiliza duas formas de operação, portanto pode ser dividido em duas categorias: mensagens de erro e mensagens de requisição, sendo que a última também é referenciada neste trabalho como mensagens de informação.

2.4.1 Mensagens ICMP de Erro

Mensagens ICMP de erro são utilizadas para relatar problemas referentes a não entrega de um datagrama IP (Arkin, 2002). O formato genérico do cabeçalho de uma mensagem ICMP de erro é apresentado na Figura (2.5).

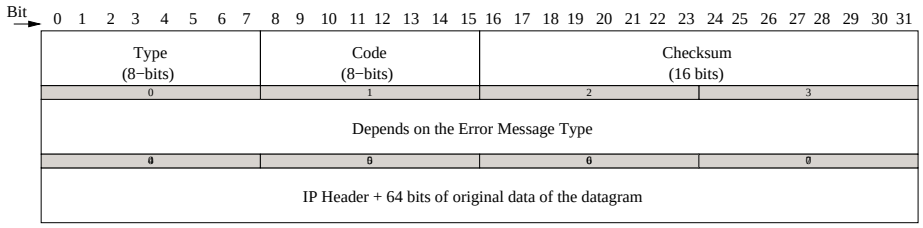

FIGURA 2.5 – Formato da mensagem ICMP de erro.
FONTE: adaptada de Arkin (2002, p. 17).

Os 4 bytes, especificados na Figura (2.5) por "*Depends on the Error Message Type*", são dependentes do tipo de mensagem utilizada. Para cada mensagem de erro, esta área do cabeçalho poderá ser dividida em um ou mais campos.

Toda mensagem ICMP de erro inclui o cabeçalho IP e pelo menos os primeiros 8 bytes de dados do datagrama que ocasionou o erro. Como o tamanho do cabeçalho IP pode variar entre 20 e 60 bytes, o tamanho de uma mensagem ICMP de erro deve estar entre 36 e 72 bytes.

Arkin (2002) apresenta algumas regras de uso do protocolo ICMP para mensagens de erro, onde são enumeradas algumas situações em que estas mensagens não devem ser enviadas. Sendo assim, mensagens ICMP de erro não são enviadas:

- em resposta a outra mensagem de erro, para evitar ciclos intermináveis;
- em resposta a datagramas destinados a endereços de *broadcast* ou *multicast*;
- em resposta a *broadcasts* da camada de enlace;
- em resposta a datagramas, cujo endereço de origem não represente um único *host*.

2.4.2 Mensagens ICMP de Informação

Segundo Arkin (2002), mensagens ICMP de informação (ou requisição) são utilizadas na obtenção de informações gerais de redes TCP/IP, através de requisições e respostas. Estas informações podem variar desde a disponibilidade de um *host* (máquina está ativa e conectada à rede), até a latência da rede. Neste ponto, entende-se por latência de rede o tempo observado, desde a saída da mensagem de requisição partindo do *host* na rede origem, até a chegada da mensagem de resposta, enviada pelo *host* na rede destino.

O formato genérico do cabeçalho de uma mensagem ICMP de informação é apresentado na Figura (2.6).

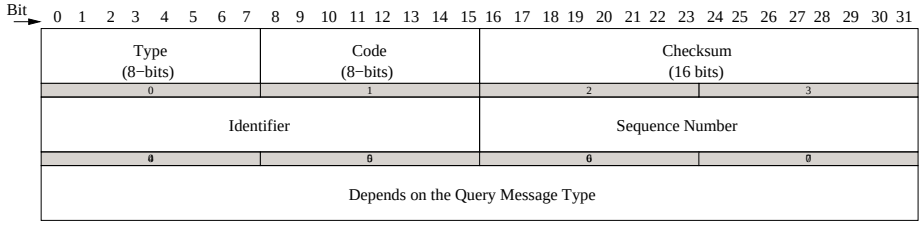

FIGURA 2.6 – Formato da mensagem ICMP de informação.
FONTE: adaptada de Arkin (2002, p. 25).

O campo *identifier* tem como função diferenciar mensagens ICMP enviadas para *hosts* distintos. Já o campo *sequence number* é utilizado para diferenciar mensagens ICMP enviadas para um mesmo *host*.

O tamanho de uma mensagem ICMP de informação varia de acordo com o tipo da mensagem, sendo que seu cabeçalho sempre ocupará 8 bytes. Por exemplo, no caso de mensagens do tipo "*Timestamp Request*" e "*Timestamp Reply*" - vide Tabela (A.1) - além dos 8 bytes, que são fixos para mensagens de informação, tem-se mais 12 bytes, referentes aos campos "*Originate Timestamp*", "*Receive Timestamp*" e "*Transmit Timestamp*", que fazem parte do conteúdo da mensagem e são intrínsecos a este tipo (Arkin, 2002).

2.5 PROTOCOLO UDP

O protocolo UDP (*User Datagram Protocol*) (Postel, 2002e) provê um mecanismo não orientado a conexão[1] e sem garantias, que através da utilização do protocolo IP, envia e recebe datagramas de uma aplicação para a outra. São utilizados números de portas para distinguir entre várias aplicações em um mesmo *host*, ou seja, cada mensagem UDP contém uma porta de origem e uma porta de destino.

[1] O termo "orientado à conexão" é melhor definido na seção 2.6.

Além disso, uma aplicação baseada no protocolo UDP é inteiramente responsável por problemas de confiabilidade (perda e duplicação de pacotes, pacotes entregues fora de ordem, dentre outros) e problemas relacionados à conexão. Programadores, ao desenvolverem aplicações baseadas em UDP, são responsáveis por implementar estas características. Como isto normalmente não ocorre, este protocolo tem grande funcionalidade em ambientes locais e em aplicações que não requerem alta confiabilidade (Comer, 2000).

A mensagem UDP é chamada de "*user datagram*" e seu cabeçalho segue o formato apresentado na Figura 2.7.

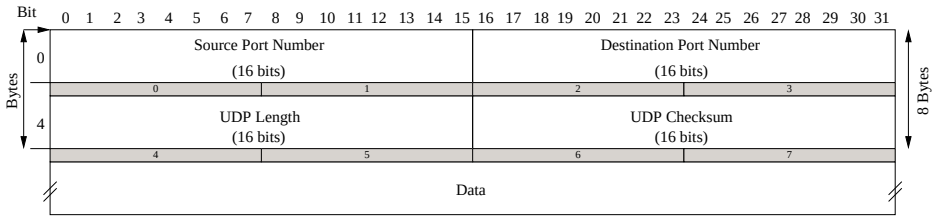

FIGURA 2.7 – Formato da mensagem UDP.
FONTE: adaptada de Stevens (1994, p. 144).

Os campos "*Source Port Number*" e "*Destination Port Number*" são utilizados na identificação dos processos de envio e recebimento de dados, respectivamente. O campo "*UDP Length*" contém o tamanho do cabeçalho UDP mais o tamanho dos dados propriamente ditos. Deve-se observar que o menor valor para este campo é igual a 8 bytes, correspondente ao tamanho do cabeçalho UDP, caso nenhum dado esteja sendo transmitido (Stevens, 1994).

2.6 PROTOCOLO TCP

O TCP (*Transmission Control Protocol*) (Postel, 2002d) é um protocolo de comunicação que provê conexões (circuitos virtuais) entre máquinas, de forma confiável, ou seja, é um protocolo orientado à conexão. Segundo Stevens (1994), o termo "orientado à conexão" significa que duas aplicações, usando um protocolo que detém esta característica, devem estabelecer uma conexão bidirecional, antes de efetuar troca de dados.

O protocolo é confiável porque quando um *host* envia dados a outro, o primeiro requer o reconhecimento relativo à chegada dos dados. Além disso, os dados são sequenciados, de forma que um número de sequência é associado a todo pacote transmitido. Isto permite que dados sejam reordenados caso sejam recebidos fora de ordem, e descartados caso sejam duplicações de dados já recebidos. Outra característica é o controle de fluxo, onde em uma conexão, um *host* sempre informa ao outro quantos bytes poderão ser aceitos,

evitando assim a ocorrência de sobrecargas do *buffer* do *host* que estiver recebendo dados (Stevens, 1998).

O formato do cabeçalho de uma mensagem TCP é apresentado na Figura 2.8 e contém um grande número de campos, utilizados no controle da comunicação e transmissão de dados entre *hosts*. Uma breve descrição, baseada em (Postel, 2002d) e (Stevens, 1994), de alguns de seus campos é apresentada como se segue.

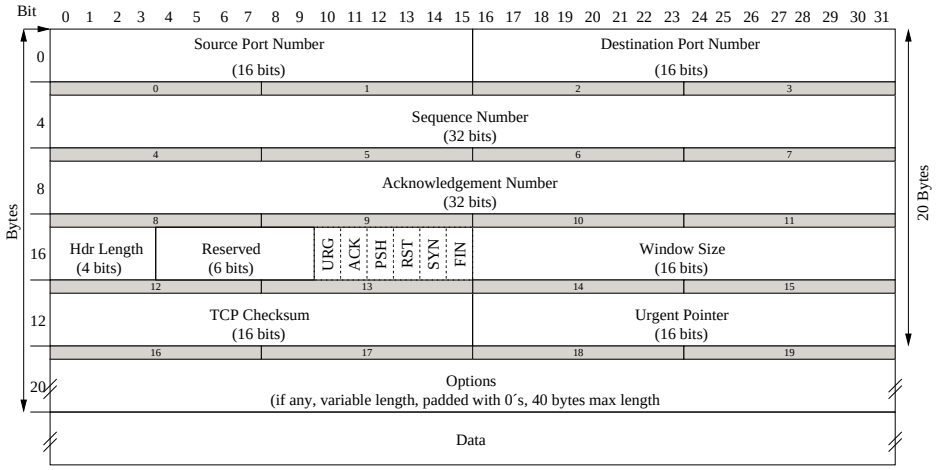

FIGURA 2.8 – Formato da mensagem TCP.
FONTE: adaptada de Stevens (1994, p. 225).

Os campos "*Source Port Number*" e "*Destination Port Number*" são utilizados na identificação das aplicações de envio e recebimento de dados, respectivamente. Quando combinados aos campos "*Source IP Address*" e "*Destination IP address*" do cabeçalho IP (seção 2.3), identificam univocamente cada conexão. "*Sequence Number*" informa o número de sequência do segmento, associado ao primeiro byte da cadeia de dados sendo transmitida. Para o segmento que representa o início de uma conexão TCP, este campo recebe um valor atribuído pelo sistema operacional, conhecido como *Initial Sequence Number* (ISN). Para cada segmento subsequente, este valor é incrementado do número de bytes transmitidos pelo segmento anterior. "*Acknowledgement Number*" contém o próximo número de sequência (ou *sequence number*) esperado. O campo "*Hdr Length*" informa o tamanho do cabeçalho da mensagem TCP, que varia de 20 a 60 bytes. Se o cabeçalho tem 20 bytes, então não há ocorrência de opções[2]. "*Window Size*" é responsável por prover o controle de fluxo implementado no protocolo TCP. Este informa qual é o número máximo de bytes que podem ser recebidos por um *host*. O campo "*Urgent Pointer*" contém um valor positivo a ser adicionado ao número de sequência do segmento, informando que os dados a partir do byte por este apontado tem maior prioridade.

[2]Detalhes sobre o campo "Options" podem ser encontrados em Postel (2002d).

Além dos campos anteriormente descritos, o cabeçalho TCP define seis bits, conhecidos como "*flags*", e suas funções são apresentadas como se segue.

- URG indica que o campo "*Urgent Pointer*" é válido;
- ACK indica que o campo "*Acknowledgement Number*" é válido;
- PSH indica que o *host* recebendo os dados deve repassá-los à aplicação assim que possível;
- RST indica o fim abrupto da conexão;
- SYN indica que os números de sequência estão sendo sincronizados para o início da conexão;
- FIN indica que o *host* terminou o envio de dados e pretende encerrar a conexão.

Conexões TCP são também conhecidas como *full-duplex*, isto é, uma aplicação pode enviar e receber dados, em ambas as direções, a qualquer momento. A idéia é que uma conexão ocorre entre uma aplicação cliente e uma servidora, onde a cliente estabelece a conexão, dados são transmitidos entre cliente/servidora e a conexão é terminada.

A subseção 2.6.1 descreve mais detalhadamente os precedimentos de início e término de uma conexão TCP.

2.6.1 Estabelecimento e Término de uma Conexão TCP

A Figura 2.9 ilustra a troca de mensagens entre dois *hosts*, para o estabelecimento (A) e o término (B) de uma conexão TCP.

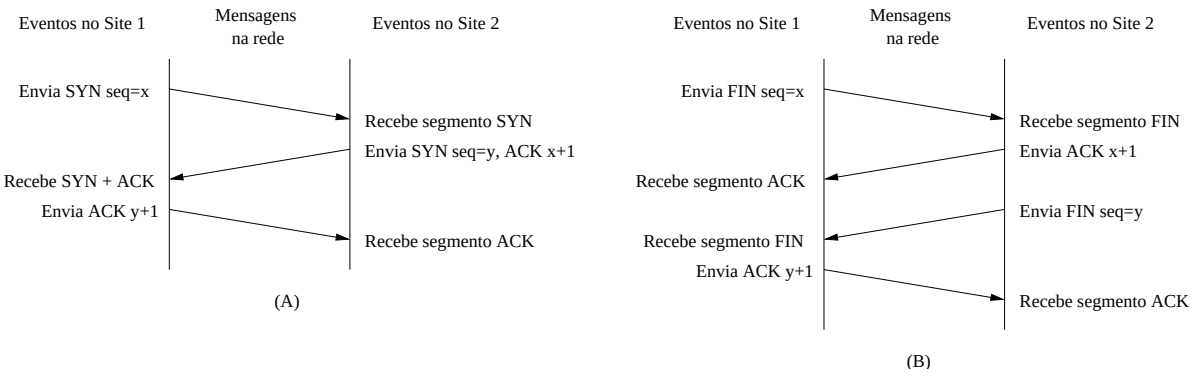

FIGURA 2.9 – (A) Estabelecimento de uma conexão TCP. (B) Encerramento de uma conexão TCP.
FONTE: adaptada de Comer (2000, p. 238,240).

O estabelecimento de uma conexão bidirecional TCP utiliza um processo conhecido como *three-way handshake*, que normalmente utiliza três mensagens. Algumas variações para o seu estabelecimento podem ser encontradas em Postel (2002d).O cliente envia um segmento SYN (*synchronize*), passando ao servidor o número de sequência inicial para os dados que serão enviados. A aplicação servidora deve reconhecer o SYN enviado pelo cliente e, em seguida, enviar um SYN contendo seu número de sequência inicial e um ACK, normalmente contido no mesmo segmento. Então, a cliente envia um ACK reconhecendo o SYN enviado pelo servidor.

Já para o término de uma conexão TCP, normalmente são utilizados quatro mensagens, pois cada *host* envolvido é responsável por fechar sua conexão, já que esta é bidirecional. Quando um *host* não possui mais dados a enviar, este solicita o encerramento da conexão através do envio de um segmento FIN, que deve ser reconhecido pelo outro *host* através do envio de um segmento ACK para o solicitante. Com isto, estabelece-se que um lado da conexão está fechado. O lado da conexão, que ainda está aberto, pode continuar enviando dados, mesmo depois do recebimento de um FIN. O mesmo procedimento, solicitando o término da conexão, deve ocorrer no sentido oposto.

Além da forma normal de término de uma conexão TCP, como descrita anteriormente, existe um outra maneira de finalização, conhecida como término abrupto. É caracterizada pelo envio de uma mensagem TCP, com o *flag* RST ativo, ao invés do FIN. O *host* que enviou a mensagem contendo o RST finaliza o seu lado da conexão, e o *host* que recebeu faz com que a conexão do seu lado seja interrompida. Mensagens de reconhecimento (ACK) não são enviadas quando do recebimento uma mensagem de término abrupto de conexão.

2.7 APLICAÇÕES

As aplicações, muitas vezes chamadas de serviços, constituem um novo conjunto de protocolos, correspondendo ao mais alto nível na pilha de protocolos TCP/IP. Estas aplicações/serviços estão associadas a uma série de parâmetros (número do protocolo, número da porta, identificadores, dentre outros), que são nomeados por um coordenador central, conhecido como *Internet Assigned Numbers Authority* (IANA, 2002), sendo este responsável por atribuir valores únicos a estes parâmetros e manter um registro atualizado dos valores catalogados.

Os números das portas (um dos parâmetros identificadores das aplicações) são, normalmente, divididos em três categorias (Stevens, 1998):

- portas privilegiadas (0-1023): são atribuídas pela IANA e na maioria dos sis-

temas só podem ser utilizadas por processos do sistema, ou por aplicações executadas por usuários privilegiados;

- portas registradas (1024-49151): são listadas e catalogadas pela IANA e podem ser usadas, na maioria dos sistemas, por processos de usuários não privilegiados, ou por aplicações executadas por usuários não privilegiados;
- portas dinâmicas e/ou privadas (49152-65535): também chamadas de portas efêmeras, não sofrem qualquer controle da IANA.

A Tabela (2.1), gerada à partir de dados coletados em IANA (2002), apresenta algumas das aplicações/serviços comumente utilizados na Internet, bem como os números das portas e protocolos associados.

TABELA 2.1: Alguns serviços/aplicações de Rede

Aplicação/Serviço	*Porta*
FTP (*file transfer*) [data]	20/tcp
FTP (*file transfer*) [control]	21/tcp
SSH (*secure shell*)	22/tcp
TELNET (*remote login*)	23/tcp
SMTP (*eletronic mail*)	25/tcp
DNS (*domain name system*)	53/udp
FINGER (*user information*)	79/tcp
HTTP (*the World Wide Web*)	80/tcp
POP3 (*post office protocol - version 3*)	110/tcp
Sun RPC (*remote procedure Call*)	111/tcp
NTP (*network time protocol*)	123/udp
IMAP (*Internet message access protocol*)	143/tcp
SNMP (*simple network management protocol*)	161/udp
NFS (*network file system*)	2049/udp

Breves descrições de algumas destas aplicações são apresentadas como se segue.

FTP (File Transfer Protocol): esta aplicação tem como objetivos promover o compartilhamento de arquivos, proteger o usuário de variações nos sistemas de armazenamento de arquivos entre *hosts* e transferir dados de forma confiável. Diferente da maioria das aplicações, o FTP usa duas conexões de rede separadas, onde a maioria de suas implementações contém dois modos de operação: o ativo e o passivo. Ao iniciar um conexão

FTP com um servidor remoto operando em modo ativo, o cliente abre, a partir de uma porta acima de 1023/tcp, um canal de controle na porta 21/tcp do servidor. Para transferir arquivos ou outras informações, o servidor remoto abre um canal de dados a partir da porta 20/tcp para uma porta acima de 1023/tcp no cliente. A conexão do canal de dados opera ao contrário da maioria dos protocolos, que abrem conexões do cliente para o servidor. Já no modo passivo, tanto a conexão de controle, quanto a conexão para a transferência de dados são iniciadas do cliente para o servidor (Postel e Reynolds, 2002a).

SSH (*Secure Shell*): O SSH é um protocolo da camada de aplicação, que provê um conjunto de mecanismos de comunicação, em forma de canais, multiplexados em um único "túnel" criptografado. Estes mecanismos são: sessões interativas de *login*; execução remota de comandos; tunelamento de conexões TCP/IP, e transferência de dados (Ylonen et al., 2002).

SMTP (*Simple Mail Transfer Protocol*): o objetivo desta aplicação é realizar o serviço de correio eletrônico (transferir mail) de forma eficiente, através da especificação da forma como as mensagens serão transferidas de um *host* para a outro. Este é independente de um sistema particular de transmissão e requer apenas um canal confiável para enviar/receber dados ordenados. O SMTP também permite que se efetue *mail relay* através de ambientes que forneçam serviços de transporte de dados (Postel, 2002c) (Crocker, 2002).

TELNET: esta aplicação tem como finalidade prover uma facilidade de comunicação bidirecional, onde o principal objetivo é proporcionar um método padronizado para o uso de uma interface de dispositivos de terminais e/ou processos orientados a terminal. O protocolo que define esta aplicação utiliza conexões do tipo TCP e é construído sobre três idéias principais: o conceito de "Terminal Virtual de Rede"; o princípio de negociação de opções; e uma visualização simétrica dos terminais e processos. O estabelecimento de uma conexão depende de uma validação, que combina nome e senha do usuário (ou cliente) da aplicação (Postel e Reynolds, 2002b).

DNS (*Domain Name System*): o DNS é um sistema de gerenciamento hierárquico e distribuído de nomes. Este contém a especificação da sintaxe dos nomes usados na Internet, regras para delegação de autoridade na definição de nomes, banco de dados distribuído para associar nomes a atributos (por exemplo, endereço IP), além de um algoritmo distribuído que mapeia nomes em endereços (Mockapetris, 2002a) (Mockapetris, 2002b).

FINGER (*Finger User Information Protocol*): esta aplicação provê a troca de informações de usuário, efetuada através de uma interface para um programa de infor-

mações de usuários remotos (RUIP - *Remote User Information Program*) (Zimmerman, 2002).

HTTP (*Hypertext Transfer Protocol*): o HTTP é um protocolo da camada de aplicação, utilizado em sistemas de informações hipermídia distribuídos e colaborativos. Também conhecido como WWW (*World-Wide Web*), é utilizado como um protocolo genérico, proporcionando a realização de várias tarefas além da convencional (transferência de hipertexto), tais como servidores de nomes e sistemas de gerenciamento de objetos distribuídos, através da extensão de seus métodos de requisição, códigos de erro e cabeçalhos. Uma das características do HTTP é a representação do tipo e negociação de dados, de forma a permitir que sistemas sejam construídos independentemente dos dados a serem transferidos (Fielding et al., 2002).

POP (*Post Office Protocol*): esta aplicação tem com função permitir que um cliente acesse e obtenha mensagens de correio eletrônico, armazenadas em um dado servidor. Esta não provê mecanismos extensivos de manipulação de mensagens, de modo que, normalmente, mensagens são obtidas pelo cliente e então apagadas do servidor (Myers e Rose, 2002).

Sun RPC (*Remote Procedure Call*): o serviço RPC foi criado para facilitar o desenvolvimento de aplicações distribuídas, baseadas no paradigma cliente/servidor. Este é um protocolo de mensagens, especificado por uma linguagem, conhecida como XDR (*eXternal Data Representation*), e é baseado no modelo de chamada de procedimentos remotos, que é similar ao modelo de chamada de procedimentos locais. O processo cliente envia uma mensagem de chamada para o processo servidor, contendo parâmetros de procedimentos, e espera pela mensagem de resposta. O processo servidor, que estava em estado de espera, extrai os parâmetros da mensagem de chamada, computa os resultados e envia uma mensagem de resposta, contendo os resultados da execução do procedimento. Uma vez que a mensagem de resposta é recebida, os resultados são extraídos e o processo cliente é terminado (Sun, 2002b).

NTP (*Network Time Protocol*): o NTP é um protocolo da camada de aplicação que provê um mecanismo para a sincronização e distribuição coordenada de tempo numa rede grande e heterogênea, com taxas de transmissão variando do corriqueiro até altíssimas. Ele usa uma arquitetura na qual servidores de tempo distribuídos numa subrede, operando com uma configuração tipo cliente-servidor hierárquico auto-organizável, sincronizam relógios lógicos dentro da subrede e com padrões de tempo nacionais. Os servidores podem distribuir referências de tempo por meio de algoritmos de roteamento local ou processos específicos de difusão pela rede (Mills, 2002a) (Mills, 2002b) (Mills, 2002c).

IMAP (*Internet Message Access Protocol*): o objetivo deste protocolo da camada de aplicação é permitir que um cliente acesse e manipule mensagens de correio eletrônico em um dado servidor. Diferente do protocolo de aplicação POP3, provê mecanismos que permitem várias formas de manipulação de mensagens no servidor. Destacam-se os seguintes mecanismos: manipulação remota de pastas (*mailboxes*) contendo mensagens; operações de criação, exclusão e troca de nomes para as *mailboxes*; remoção permanente de mensagens, dentre outros (Crispin, 2002).

SNMP (*Simple Network Management Protocol*): o SNMP é um protocolo da camada de aplicação, utilizado no gerenciamento de redes TCP/IP. Seus processos, que atuam como gerentes ou agentes, coletam de objetos gerenciados informações úteis para o gerenciamento da rede. Estes objetos representam recursos, tais como sistema (estação de trabalho), *gateway*, ou equipamentos de transmissão (*modem*, *bridge*, concentrador). Cada objeto gerenciado é mapeado como uma coleção de variáveis, cujos valores podem ser lidos e alterados. O gerente processa as informações recolhidas pelos agentes, com o objetivo de detectar a presença de falhas no funcionamento dos componentes da rede (*hosts*, *gateways*, processos executando protocolos de comunicação, dentre outros), de forma que providências possam ser tomadas para contornar os problemas gerados pelas falhas (Case et al., 2002).

NFS (*Network File System*): o serviço de NFS foi projetado para prover acesso transparente a arquivos/diretórios compartilhados através da rede, e para ser portável entre diferentes *hosts*, sistemas operacionais, arquiteturas de rede e protocolos de transporte. Esta portabilidade é alcançada através do uso de RPCs, construídas sobre a linguagem XDR. Do ponto de vista de programação, este sistema não acrescenta nenhum procedimento específico para o acesso a arquivos remotos. Quando o NFS é instalado em um *host*, o acesso a arquivos remotos é efetuado com as mesmas chamadas de procedimentos utilizadas para acessar arquivos locais (Sun, 2002a).

CAPÍTULO 3

CAPTURA DE PACOTES EM REDES

Este capítulo mostra uma breve revisão sobre captura de pacotes. É apresentada a arquitetura utilizada para a captura, a biblioteca que implementa esta arquitetura, bem como uma aplicação que utiliza esta biblioteca para visualizar pacotes coletados.

3.1 ARQUITETURA PARA A CAPTURA DE PACOTES

Na administração de sistemas de informação, uma técnica que vem sendo amplamente utilizada é a "captura de pacotes". Esta veio possibilitar a monitoração de redes, através de aplicações que realizam esta tarefa. Como esses monitores de rede são executados na forma de processos de usuário, pacotes precisam ser copiados do núcleo do sistema operacional (*kernel*) para o espaço do usuário, antes de serem avaliados.

Uma forma de minimizar as operações de cópia de pacotes consistiu no desenvolvimento de um agente de *kernel*, conhecido como "*packet filter*" (ou filtro de pacotes), cuja função é descartar pacotes desnecessários o mais rápido possível. Os *packet filters* iniciais foram desenvolvidos sobre uma arquitetura baseada em pilhas, e apresentaram um baixo desempenho quando executados nos processadores RISC daquela época.

McCanne e Jacobson (1993) apresentaram uma nova arquitetura para a captura de pacotes, conhecida como "*BPF - BSD Packet Filtering*". Esta nova abordagem, baseada no uso de registradores, mostrou-se 20 vezes mais rápida, quando comparada a abordagem inicialmente proposta.

A arquitetura BPF, apresentada na Figura (3.1), é constituída por dois componentes principais: a sonda de rede e o filtro de pacotes. A sonda de rede coleta cópias de pacotes nos *device-drivers* de rede e os entrega para as aplicações de monitoramento de rede. Já o filtro de pacotes decide se um pacote deve ou não ser coletado, e quantos bytes do mesmo devem ser copiados para a aplicação.

Quando o BPF está sendo executado em um sistema e um pacote chega na interface de rede, antes do *device-driver* da interface decidir o que vai ser feito com este pacote, o BPF realiza as seguintes ações:

- envia uma cópia do pacote para cada processo de filtragem;
- estes filtros decidem se o pacote deve ou não ser aceito;
- cada filtro que aceita o pacote, copia a quantidade de dados requisitada para

o *buffer* associado ao filtro.

O controle é então devolvido ao *device-driver* da interface de rede, que envia o pacote para a pilha de protocolos do sistema, caso esteja endereçado para o próprio *host*.

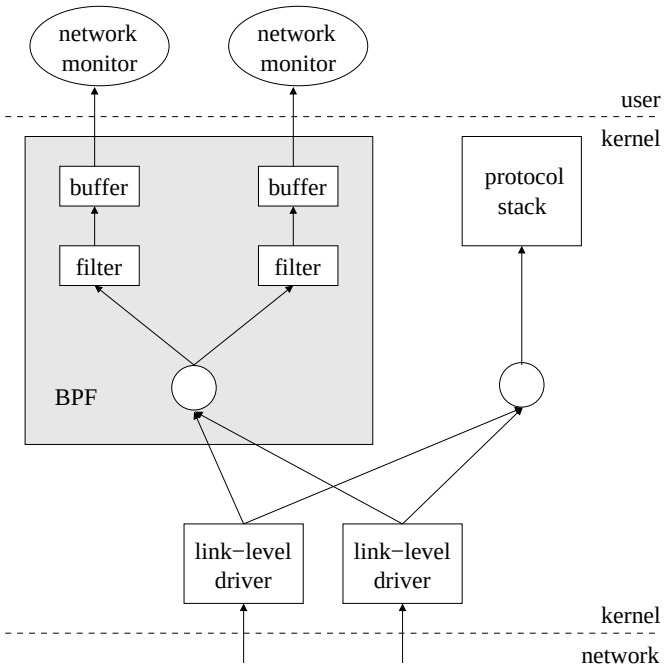

FIGURA 3.1 – Esquema da arquitetura BPF (*BSD Packet Filtering*).
FONTE: adaptada de McCanne e Jacobson (1993, p. 260).

Uma característica fundamental para o bom desempenho desta arquitetura é a capacidade de coletar e armazenar vários pacotes entre uma chamada de leitura do sistema e outra. Seria inviável realizar uma chamada de leitura do sistema para cada pacote coletado, pois o intervalo entre pacotes pode ser de alguns microsegundos.

Portanto, o BPF realiza a coleta e o armazenamento de vários pacotes, de modo que estes são repassados quando da ocorrência de uma chamada de leitura do sistema realizada pelo processo de monitoramento de rede. Além disso, o BPF encapsula os dados capturados para cada pacote com um cabeçalho próprio, que contém o horário de captura, tamanho e deslocamento para o alinhamento dos dados.

A seção 3.2 descreve a biblioteca *libpcap*, implementação atual da arquitetura BPF, proposta por McCanne e Jacobson (1993).

3.2 BIBLIOTECA *LIBPCAP*

McCanne e Jacobson puderam apresentar as vantagens da arquitetura BPF, através da implementação de uma biblioteca de captura de pacotes, conhecida como *libpcap*[1]. Atualmente, o grupo *tcpdump.org*[2] é responsável por dar continuidade ao desenvolvimento dessa biblioteca.

Esta é composta por um conjunto de rotinas, cujas principais funções são descritas como se segue:

- especificar dispositivo de rede onde a coleta deve ser efetuada;
- criar uma sessão de captura e associá-la a um descritor da biblioteca;
- compilar uma expressão de filtragem para o formato que pode ser entendido pela biblioteca;
- aplicar a expressão de filtragem, já compilada, à sessão de captura previamente criada;
- iniciar o processo de obtenção de pacotes da rede;

A rotina de especificação do dispositivo de rede pode ser suprimida. Neste caso, a biblioteca se encarrega de identificar quais dispositivos estão ativos e, portanto a coleta de pacotes é realizada em todas as interfaces válidas. A sessão de captura é associada a um descritor, pois este será utilizado como identificador para posterior aplicação de filtros e início do processo de coleta. A *libpcap* tem uma sintaxe própria para as expressões de filtragem, grande responsável pelo melhor desempenho da biblioteca. Por isso, é necessário compilar os filtros antes de aplicá-los à sessão de coleta de pacotes. O processo de captura pode ser feito para um número determinado de pacotes ou indefinidamente. Para o segundo caso, a coleta é terminada quando da ocorrência de uma interrupção de sistema gerada para este fim.

Além das funções anteriormente descritas, a biblioteca permite que os pacotes coletados sejam armazenados em arquivo e disponibiliza rotinas para recuperar os pacotes previamente armazenados. Para este caso, não há a necessidade de especificação de um dispositivo de rede e a sessão de captura é associada a um arquivo de dados. Filtros podem ser aplicados a este tipo de sessão e o processo de obtenção de pacotes ocorre de forma semelhante.

[1] Disponível em `ftp://ftp.ee.lbl.gov/libpcap.tar.Z`.
[2] Página Web em `http://www.tcpdump.org`.

As rotinas de obtenção de pacotes, seja da rede ou de um arquivo de dados, têm como argumento uma função de retorno, conhecida como "*callback function*", definida pela aplicação que está utilizando a biblioteca. Esta função recebe a sequência de bytes capturados para cada pacote e é responsável por manipular estes dados. Portanto, a *libpcap* não implementa qualquer tipo de tratamento para a sequência de bytes capturados, sendo que esta tarefa é de total responsabilidade da aplicação.

Descrições mais detalhadas das funções de maior relevância para este trabalho, bem como seus argumentos são apresentadas na seção 5.3.6.

A seção 3.3 descreve a primeira aplicação baseada na biblioteca *libpcap*, desenvolvida para o monitoramento de redes.

3.3 *TCPDUMP*

A ferramenta de monitoramento de rede *tcpdump*[3] foi originalmente implementada por McCanne e Jacobson (1993), e da mesma forma que a biblioteca *libpcap*, a continuidade de seu desenvolvimento é de responsabilidade do grupo *tcpdump.org*[4].

Esta tem como finalidade processar pacotes sendo coletados em uma rede ou lidos de um arquivo previamente armazenado. Dispõe de parâmetros que permitem gerar diferentes formas de apresentação dos pacotes sendo impressos e que especificam como as rotinas da *libpcap* devem ser executadas.

A Figura (3.2) apresenta as etapas de processamento de um pacote, realizadas pela função de retorno (*callback function*) definida na implementação do *tcpdump*. Esta função é passada para a biblioteca *libpcap* e é responsável pelo tratamento de cada pacote coletado. Por questões de simplificação e entendimento, o exemplo mostrado na Figura (3.2) apresenta o processamento de um pacote TCP/IP, onde o protocolo da camada de enlace de dados é o *Ethernet*. A descrição de cada uma das etapas é apresentada como se segue.

Na etapa *A* a *libpcap* passa para a função de retorno o "cabeçalho PCAP", definido por uma estrutura contendo o horário da captura do pacote (*timestamp*), a quantidade de bytes capturados (*caplen*) e o deslocamento para o alinhamento dos dados (*offset*), como descrito na seção 3.1. Além disso, recebe um apontador, do tipo *unsigned char* (*ucp*), que aponta para o início da sequência de bytes capturados para o pacote.

Na etapa *B* inicializam-se dois apontadores: um para o início da sequência de bytes (*pktp*) e outro para o fim (*snapend*). O segundo é calculado pela soma do apontador de início

[3]Disponível em `ftp://ftp.ee.lbl.gov/tcpdump.tar.Z`.
[4]Idem 2.

com a quantidade de bytes coletados (*caplen*). Inicializa-se, então, o apontador *ep*, que referencia a estrutura contendo o formato do cabeçalho *Ethernet* (*struct ether_header*). Isto é possível através da utilização de uma característica da linguagem C (Kernighan e Ritchie, 1990), utilizada no desenvolvimento do *tcpdump* e da *libpcap*, que permite fazer conversão (*casting*) de tipos. Por *cast* (ou *casting*) entende-se converter um determinado tipo para outro, respeitando os limites (ou tamanhos) definidos para cada tipo envolvido. Os dados referentes ao cabeçalho *Ethernet* são então processados.

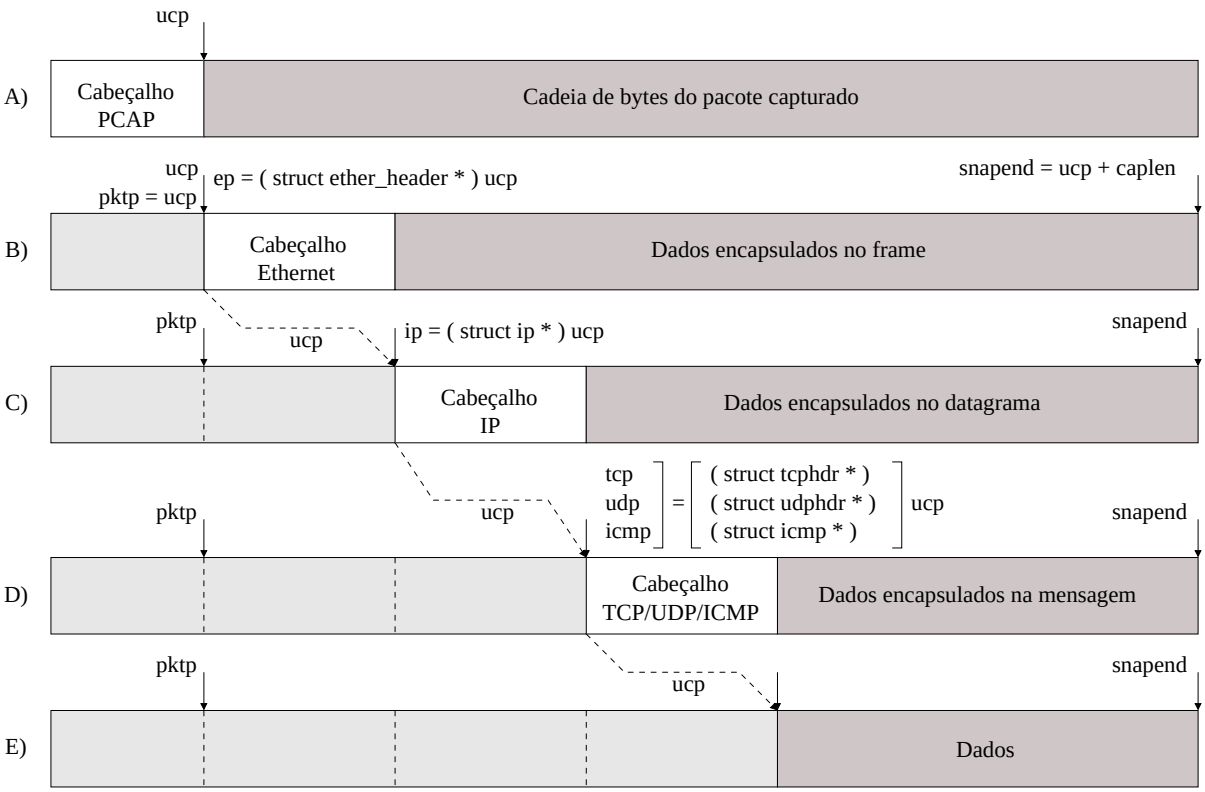

FIGURA 3.2 – Etapas de processamento de um pacote TCP/IP realizadas pelo *tcpdump*.

Na etapa *C* o apontador *ucp* é incrementado do tamanho da estrutura *ether_header*. Então é inicializado o apontador *ip*, que referencia a estrutura contendo o formato do cabeçalho IP (*struct ip*). Os dados referentes ao cabeçalho IP são processados. Neste ponto é feita a verificação do protocolo da camada de transporte utilizado no pacote.

Na etapa *D* o apontador *ucp* é novamente incrementado, mas agora do tamanho da estrutura *ip*. Então é inicializado o apontador *tcp* ou *udp* ou *icmp*, de acordo com o protocolo identificado na etapa anterior. Este apontador referencia a estrutura contendo o formato do cabeçalho TCP (*struct tcphdr*), ou cabeçalho UDP (*struct udphdr*), ou o cabeçalho ICMP (*struct icmp*). Os dados referentes ao cabeçalho em questão são então processados.

Na etapa *E* o apontador *ucp* é incrementado do tamanho da estrutura referente ao cabeçalho da camada de transporte. Então, rotinas de processamento de protocolos da camada de aplicação são executadas.

Para os casos onde o cabeçalho tem tamanho variável, como no IP e TCP, quando da ocorrência de opções, o apontador *ucp* é reajustado, de modo que o processamento a ser executado na próxima etapa aconteça de forma correta.

No decorrer do processamento do pacote, informações de cada cabeçalho são impressas, de acordo com argumentos passados para o *tcpdump* no momento em que este é executado. A descrição detalhada de cada um destes argumentos pode ser encontrada nos manuais (*man pages*) disponibilizados juntamente com a aplicação.

Uma vez aplicados os filtros e processado o pacote, o resultado é armazenado num arquivo ou impresso na tela de um terminal. Um exemplo da síntese de um comando *tcpdump* e a saída[5] correspondente são mostrados a seguir:

```
            argumentos       filtro
          /------------\ /-------------\
# tcpdump -nSr dump_file tcp and port 22

14:38:04.080644 eth0 > xxx.xxx.xxx.xxx.2754 > yyy.yyy.yyy.yyy.22: S 1145744603:1145744603(0) win 32120 \
                <mss 1460,sackOK,timestamp 155682026 0,nop,wscale 0> (DF)
14:38:04.081408 eth0 < yyy.yyy.yyy.yyy.22 > xxx.xxx.xxx.xxx.2754: S 4038512504:4038512504(0) \
                ack 1145744604 win 10136 <nop,nop,timestamp 371616198 155682026, nop,...,mss 1460> (DF)
14:38:04.081529 eth0 > xxx.xxx.xxx.xxx.2754 > yyy.yyy.yyy.yyy.22: . 1145744604:1145744604(0) \
                ack 4038512505 win 32120 <nop,nop,timestamp 155682026 371616198> (DF)
14:38:04.099287 eth0 < yyy.yyy.yyy.yyy.22 > xxx.xxx.xxx.xxx.2754: P 4038512505:4038512554(49) \
                ack 1145744604 win 10136 <nop,nop,timestamp 371616200 155682026> (DF)
14:38:04.099396 eth0 > xxx.xxx.xxx.xxx.2754 > yyy.yyy.yyy.yyy.22: . 1145744604:1145744604(0) \
                ack 4038512554 win 32120 <nop,nop,timestamp 155682027 371616200> (DF)
14:38:04.108752 eth0 > xxx.xxx.xxx.xxx.2754 > yyy.yyy.yyy.yyy.22: P 1145744604:1145744653(49) \
                ack 4038512554 win 32120 <nop,nop,timestamp 155682028 371616200> (DF)
...
```

[5] As linhas da saída *tcpdump* foram editadas e quebradas, por questões de legibilidade, e os endereços IP foram substituídos.

CAPÍTULO 4

DETECÇÃO DE INTRUSÃO

Este capítulo apresenta alguns conceitos e princípios relacionados a segurança de sistemas de informação, bem como a detecção de intrusão. São também abordadas as principais categorias e métodos de detecção relacionados aos sistemas de detecção de intrusões. Ao final, são apresentados alguns dos principais sistemas de detecção de intrusões projetados e implementados, em ordem cronológica. A arquitetura, funcionamento, principais vantagens e desvantagens destes sistemas também são discutidos neste capítulo.

4.1 CONCEITOS DE SEGURANÇA DE SISTEMAS

Entende-se por sistemas de informação os computadores, as redes que os interligam, bem como as informações que estes manipulam. Um sistema de informação seguro está relacionado à confidencialidade, integridade e disponibilidade dos recursos que o compõe. A confidencialidade diz que a informação só está disponível para aqueles devidamente autorizados; a integridade diz que a informação não é destruída ou corrompida e o sistema tem um desempenho correto, e a disponibilidade diz que os serviços/recursos do sistema estão disponíveis sempre que forem necessários.

Outra definição de segurança de sistemas de informação é apresentada por Garfinkel e Spafford (1996), onde um sistema de informação é seguro se este se comporta da forma esperada. Além da confidencialidade, integridade e disponibilidade, há outros requisitos que podem determinar o nível de segurança de um sistema de informação. A consistência diz respeito à certeza de que o sistema se comporta como o esperado pelos usuários autorizados; o controle diz que o acesso ao sistema é restrito, e a auditoria diz respeito à preocupação quanto as atividades realizadas pelos usuários autorizados do sistema.

Apesar de cada um dos requisitos/aspectos de segurança apresentados terem grande importância, diferentes organizações terão diferentes necessidades e, portanto, alguns aspectos/requisitos terão maior importância do que outros. Em um sistema bancário a integridade e a auditoria são, normalmente, os que recebem maior atenção. Já em um instituto ou universidade a integridade e disponibilidade devem ser os requisitos de maior importância.

Um fator determinante e crucial para a segurança de um sistema de informação é o estabelecimento de políticas, que apontem as propriedades de segurança que o sistema deve ter. Estas políticas de segurança são estabelecidas pelas necessidades e particularidades da organização. São definidas através de uma análise de riscos e ameaças aos sistemas de

informação, estabelecendo contra o que a organização deve se defender e de que forma.

Anderson (1980) apresentou alguns conceitos relacionados aos riscos e ameaças que afetam a segurança dos sistemas de informação. Estes são descritos como se segue e serão utilizados nas seções seguintes.

- Ameaça: possibilidade de ocorrência de uma tentativa deliberada e não autorizada para: acessar informações; manipular informação, ou tornar um sistema não confiável ou indisponível;
- Risco: exposição acidental e imprevista de informações, ou violação da integridade das operações, relacionadas ao mal funcionamento do hardware ou projeto incorreto/incompleto de software.
- Vulnerabilidade: Uma suspeita de falha ou uma falha conhecida no projeto do hardware ou software, ou operação de um sistema que expõe informações;
- Ataque: formulação específica ou execução de um plano que explora uma ameaça.
- Intrusão: ataque bem sucedido, habilidade de obter acesso não autorizado (e muitas vezes não detectável) a arquivos e programas, ou controle de um sistema de informação.

Outra definição importante diz respeito ao indivíduo que realiza um ataque/intrusão. São conhecidos como executores/atacantes/intrusos e, normalmente, são usuários legítimos ou não, que exploram ou tentam explorar vulnerabilidades do sistema para: comprometer recursos, fraudar, roubar ou sabotar informações, ou mesmo obter acesso a informações de forma não autorizada. Todas estas ações, que são consideradas violações, resultam na transgressão da política de segurança do sistema de informações.

Os atacantes internos representam a ameaça mais séria, pois suas ações podem resultar em danos onerosos e desastrosos para a organização. Os atacantes externos são normalmente motivados pelos grandes desafios técnicos. Frequentemente passam a atuar por ganhos financeiros.

Sabe-se que a segurança de um sistema de informação é obtida, de acordo com a conveniência e facilidades de uso de recursos que o compõem. Este fato é crítico e complicador na defesa de um sistema, pois sabe-se que quanto mais facilidades e recursos um sistema disponibilizar, mais suscetível estará a falhas e a ocorrência de vulnerabilidade. Daí vem a necessidade de utilização de mecanismos que permitam detectar a exploração de falhas de segurança, ou seja, detectar intrusões. Estes mecanismos são conhecidos como “SDIs -

Sistemas de Detecção de Intrusões", e desempenham a função mais uma linha de defesa dentro de um sistema de informação.

4.2 DETECÇÃO DE INTRUSÃO

Anderson (1980), ao introduzir o conceito de detecção de intrusão, definiu uma tentativa de intrusão ou ameaça como sendo a possibilidade de ocorrência de uma tentativa deliberada e não autorizada para:

- acessar informações;
- manipular informação;
- tornar um sistema não confiável ou indisponível.

De modo geral, intrusões podem ser divididas em 6 tipos principais (Smaha, 1988):

- Tentativa de invasão (*Attempted Break-in*): detectado através de perfis de comportamento atípico ou violações de restrições de segurança;
- Ataque (*Masquerade Attack*): também detectado através de perfis de comportamento atípico ou violações de restrições de segurança;
- Penetração (*Penetration of the Security Control System*): detectado através do monitoramento de padrões específicos de atividade;
- Vazamento (*Leakage*): detectado pelo uso atípico dos recursos do sistema;
- Negação de serviços (*Denial of Service*): também detectado pelo uso atípico dos recursos do sistema;
- Uso mal intencionado (*Malicious Use*): detectado através de perfis de comportamento atípicos, violações de restrições de segurança, ou uso de privilégios especiais.

Dentre as várias definições para detecção de intrusão, Amoroso (1999) diz que: "Detecção de Intrusão é o processo de identificar e responder a atividades mal intencionadas, direcionadas para recursos computacionais e de rede".

É relevante para este trabalho analisar alguns dos termos utilizadas na definição apresentada acima (Amoroso, 1999):

- *Processo*: detecção de intrusão deve ser vista primeiramente como processo envolvendo tecnologia, pessoas e ferramentas;

- *Identificar*: identificação de intrusão pode ocorrer antes, durante ou depois da execução da atividade mal intencionada. Isto é fundamental para determinar o tipo de detecção e ações a serem tomadas;

- *Responder*: está diretamente relacionada e depende da identificação da intrusão. A resposta pode ser a finalização de um serviço, a identificação do intruso, entre outras.

- *Atividades mal intencionadas*: quaisquer atividades que atentem contra a segurança de um sistema de informação.

As técnicas de detecção de intrusão são divididas em duas categorias principais: detecção de intrusão por anomalia e detecção de intrusão por abuso. A discussão apresentada a seguir baseou-se nos trabalhos de Kumar (1994), (Allen et al., 2000), (Bace e Mell, 2000) e Sundaram (2000).

4.2.1 Detecção de Intrusão por Anomalia

A detecção de intrusão por anomalia é baseada na idéia de que todo evento intrusivo é necessariamente anômalo, sendo um subconjunto da atividade anômala. Esta idéia parece bem razoável, uma vez que um intruso, ao invadir um sistema, não conhece o padrão de uso dos recursos, gerando assim, um comportamento anômalo. Normalmente, atividades intrusivas podem ser vistas como um conjunto de atividades individuais, sendo que estas muitas vezes não caracterizam atividades anômalas. Portanto, teoricamente, bastaria encontrar as atividades anômalas para encontrar todas as atividades intrusivas. Mas isto nem sempre acontece, gerando assim os seguintes casos:

- Intrusivo e não anômalo: conhecido como falso-negativo, caracteriza falha na detecção, pois a atividade é intrusiva apesar de não ser anômala;

- Não intrusivo e anômalo: conhecido como falso-positivo, caracteriza falha na detecção, pois a atividade não é intrusiva, mas por ser anômala, é apontada como tal;

- Não intrusivo e não anômalo: caracteriza uma atividade normal;

- Intrusivo e anômalo: caracteriza uma intrusão, pois a atividade é intrusiva por ser também anômala.

Os sistemas de detecção de intrusão por anomalia são, normalmente, baseados em perfis do sistema. Portanto, seu maior problema consiste na mudança dos perfis esperados, que tendem a gerar um grande número de falso-positivos. Outra questão está relacionada ao

custo destes sistemas. Por utilizarem estes perfis e tendo que mantê-los atualizados e sob vigilância constante, estes sistemas normalmente têm o custo bastante elevado. A Figura (4.1) mostra a representação básica de um sistema de detecção baseado neste método.

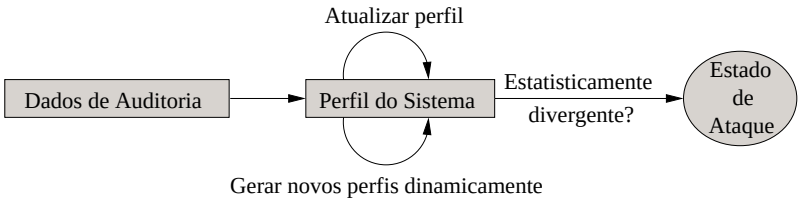

FIGURA 4.1 – Sistema de Detecção de Intrusão por Anomalia (esquema básico).
FONTE: adaptada de Sundaram (2000, p. 3).

Os sistemas que utilizam o método de detecção por anomalia são desenvolvidos segundo algumas abordagens, descritas nas subseções seguintes.

4.2.1.1 Abordagem estatística

Nesta abordagem, o detector de anomalias monitora as atividades dos usuários e processos do sistema e gera perfis de comportamento destes. Estes perfis são atualizados regularmente, sendo que os dados antigos são ajustados de forma apropriada. Logo que os registros de auditoria são processados, é gerado periodicamente um valor indicativo de anormalidade, sendo que este valor é função de medidas de anormalidade individuais, associadas ao perfil do usuário. Existem vários tipos de medidas de atividades, relacionadas a um perfil, tais como tempo de uso da CPU, número de conexões na rede em um certo período, acesso a arquivos, utilização de processos envolvendo I/O (entrada/saída), frequência de *logins* a partir de uma determinada localidade, utilização do servidor de *mail*, compiladores, *shells*, editores do sistema, etc. O comportamento do usuário é armazenado em um perfil corrente. Em intervalos de tempo regulares o perfil corrente é comparado com o perfil armazenado, para determinar se há ocorrência de comportamento anômalo. Alguns destes sistemas concatenam, em intervalos determinados, o perfil corrente do usuário com o seu perfil correspondente armazenado. Outros sistemas são estáticos, e não alteram o perfil armazenado uma vez que este foi determinado.

A grande vantagem desta abordagem é que esta aprende, de forma adaptativa, o comportamento dos usuários do sistema. Mas, existem alguns problemas relacionados a esta abordagem. Os SDIs baseados no método estatístico podem ser gradativamente treinados por intrusos, de forma que eventos intrusivos passem a ser considerados normais; a configuração do limiar (indica se uma anomalia deve ser considerada intrusiva) muito alto ou muito baixo pode gerar um número considerável de falso positivos e falso negativos, e

o relacionamento entre eventos pode não ser considerado, pois as medidas estatísticas de anormalidade não levam em consideração a ordem de ocorrência dos eventos.

4.2.1.2 Geração de prognósticos de padrões

Esta abordagem tenta prever eventos futuros, baseando-se em eventos que já ocorreram. A idéia é que sequências de eventos não são aleatórias, mas seguem um padrão previsível. Uma abordagem desenvolvida por Teng et al. (1990) utiliza regras temporais geradas por indução para caracterizar padrões de comportamento normais dos usuários. As regras são modificadas dinamicamente durante a fase de aprendizado, de forma que somente regras confiáveis permanecem no sistema.

Seja o seguinte exemplo: $E_1 \rightarrow E_2 \rightarrow E_3 \Rightarrow (E_4 = 70\%, E_5 = 25\%, E_6 = 5\%)$.

Isto significa que dadas as ocorrências dos eventos E_1, E_2 e E_3, nesta ordem, existe uma probabilidade de 70% para o evento E_4, ou 25% para o evento E_5, ou 5% para o evento E_6 ocorrer em sequência. Um conjunto de regras geradas por indução, através da observação do comportamento do usuário, então compreende o perfil do usuário. Uma anomalia é detectada se a sequência de eventos observada confere com o lado esquerdo de uma regra, mas os eventos seguintes diferem estatisticamente, e de forma considerável, daqueles previstos pela regra.

Uma desvantagem desta abordagem consiste na existência de padrões de comportamento desconhecidos, que não podem ser identificados como anômalos, apesar de não coincidirem com as regras estabelecidas para o sistema. Algumas das vantagens são: maior facilidade em casos onde usuários possuem comportamentos bem variados, mas que apresentam padrões sequenciais bem significativos, foco em uma quantidade menor de eventos de segurança realmente relevantes, comparado com a análise de todo andamento de uma sessão considerada suspeita, e maior precisão na detecção de violações, ou seja, atacantes tentando treinar o sistema na fase de aprendizado são mais facilmente detectados, devido às características semânticas das regras.

4.2.1.3 Utilização de redes neurais

Uma outra abordagem, que segue a mesma linha da abordagem de geração de prognósticos de padrões, utiliza redes neurais. A rede é treinada sobre um conjunto de informações (comandos), de forma que esta possa prever a próxima ação ou comando de um usuário. Estes comandos ou ações não são necessariamente os registros de auditoria, mas podem estar em um nível de abstração mais elevado. A rede neural toma como entrada o comando atual e os **n** comandos anteriores, onde **n** é o tamanho da janela que constitui os

comandos anteriores, tomados pela rede, para efetuar a previsão do próximo comando. Após a fase de treinamento, a rede constrói os perfis dos usuários sobre sequências de comandos relevantes. Qualquer previsão de eventos incorreta é sinalizada como um desvio, em relação ao perfil do usuário previamente estabelecido. Esta abordagem apresenta algumas vantagens, tais como: tolerante a ruídos no conjunto de informações, o sucesso desta abordagem não depende de hipóteses estatísticas à respeito da natureza dos dados, e é de fácil modificação para novos grupos de usuários. As desvantagens são: uma janela de comandos pequena pode gerar falso positivos, enquanto que janela de comandos grande irá gerar dados irrelevantes e poderá ocasionar em falso negativos, intrusos podem treinar a rede durante a fase de aprendizado, padrões não podem ser induzidos a partir de amostras, e a topologia da rede neural só é determinada depois de vários testes e erros.

4.2.2 Detecção de Intrusão por Abuso

A detecção de intrusão por abuso baseia-se na idéia que ataques podem ser representados na forma de padrões ou assinaturas (sequência de eventos e condições que levam a uma intrusão). Um fato importante é que existe um componente de detecção por abuso na maioria dos sistemas de detecção de intrusão, devido a fato de que técnicas estatísticas isoladas são limitadas na detecção de todos os eventos de segurança, como visto anteriormente. A Figura (4.2) mostra a representação básica de um sistema de detecção de intrusão por abuso.

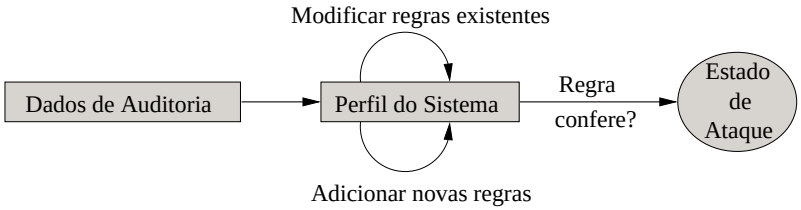

FIGURA 4.2 – Sistema de Detecção de Intrusão por Abuso (esquema básico).
FONTE: adaptada de Sundaram (2000, p. 4).

A grande limitação deste método consiste no fato de que este busca por vulnerabilidades conhecidas, sendo que novas vulnerabilidades não são reconhecidas como intrusões. O principal aspecto relacionado aos sistemas de detecção de intrusão que utilizam o método por abuso, é como preparar uma assinatura, de modo que esta englobe todas as possíveis variações de um ataque determinado. E, além disso, prepará-la de forma que esta não se encaixe em uma atividade não intrusiva.

Os sistemas que utilizam métodos de detecção de intrusão por abuso são desenvolvidos segundo algumas abordagens, descritas nas subseções seguintes.

4.2.2.1 Utilização de sistemas especialistas

Sistemas especialistas são capazes de representar e inferir sobre algum domínio do conhecimento, visando solucionar problemas. Estes são modelados, de forma que haja uma separação entre o centro de inferência e a formulação da solução do problema (base de conhecimento representada por regras e fatos). Quando utilizados na detecção de intrusões, estes sistemas codificam o conhecimento sobre ataques, através da declaração e associação de fatos extraídos de eventos nos processos auditores. Estas codificações geram regras contendo condições e ações a serem realizadas. Quando um conjunto de condições descritas em uma regra é satisfeito, um conjunto de ações é executado. Algumas vantagens deste método são: dedução simbólica da ocorrência de uma intrusão baseada nos dados recebidos e parâmetros podem ser combinados para gerar um quadro coeso acerca de uma intrusão. Algumas desvantagens são: pode apenas detectar vulnerabilidades conhecidas, e o conhecimento relacionado ao sistema especialista depende do profissional que efetua a modelagem deste conhecimento em um conjunto de regras.

4.2.2.2 Detecção por reconhecimento de padrões

Esta abordagem de detecção de intrusão por abuso codifica assinaturas de intrusões conhecidas como padrões, que são então reconhecidos nos dados de auditoria (Kumar, 1995). Este modelo tenta reconhecer os eventos como padrões representando situações de intrusão. A implementação utiliza diagramas de transição de estados, para representar os padrões, sendo que uma transição corresponde à ocorrência de um determinado evento. A Figura (4.3) apresenta um exemplo de codificação, utilizando diagrama de transição de estados.

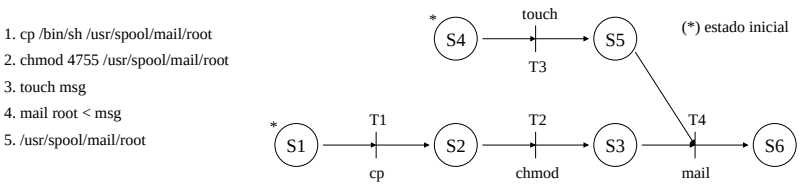

FIGURA 4.3 – Representação de Eventos com Diagrama de Transição de Estados.
FONTE: adaptada de Kumar (1994, p. 19).

Cada transição é rotulada, ou seja, recebe um nome que caracteriza o evento associado (*label*). Uma transição entre estados é disparada após o evento correspondente a esta ocorrer. Além disso, expressões opcionais podem ser associadas a cada transição. Estas expressões, conhecidas como *guards*, são booleanas (assumem valores do tipo *true* ou *false*), e são avaliadas no contexto do evento a que estão associadas. Por exemplo, na Figura (4.3), um *guard* na transição **T4** só permite que esta seja disparada, se os estados

S3 e **S5** foram alcançados e um evento *mail* ocorrer. Quando ocorre um evento reconhecido como um *label* (transição) no diagrama, este dispara a transição e a verificação do padrão passa para um novo estado. Quando um determinado estado é alcançado, é sinalizado um evento de segurança.

Algumas vantagens desta abordagem são: utiliza especificação declarativa, ou seja, precisa apenas especificar que padrões devem ser reconhecidos, e não como reconhecê-los, cadeias de eventos podem ser processadas de forma independente, sendo que seus resultados podem ser analisados em conjunto para evidenciar uma atividade intrusiva, e tem como característica a portabilidade, ou seja, desde que assinaturas de intrusões são escritas em uma linguagem independente de sistema, estas não precisam ser reescritas para processos de auditoria diferentes. Algumas desvantagens são: pode apenas detectar ataques baseados em vulnerabilidades conhecidas, e não é útil na representação de padrões pobremente definidos, visto que a tradução de situações de ataques conhecidas em padrões que possam ser utilizados pelo modelo, não é uma tarefa simples.

4.2.3 Sistemas Híbridos

Basicamente, existem as duas técnicas de detecção de intrusões apresentadas anteriormente: detecção por anomalia e detecção por abuso. Vários sistemas são baseados na detecção por abuso ou na detecção por anomalia, mas ainda existem sistemas que combinam ambas, com o intuito de resolver ou atenuar os deficiências apresentadas por cada uma das técnicas. Um fator importante é que a modelagem destes sistemas, normalmente, é muito complexa, contendo várias considerações, restrições, ou mesmo limitações relacionadas a sua implantação. Geralmente, estes sistemas são implementados de forma parcial, e apresentam bons resultados.

4.2.4 Classificação dos Sistemas de Detecção por Tipo de Análise

Além da duas técnicas principais de detecção de intrusão apresentadas anteriormente, os sistemas de detecção de intrusão podem ser divididos em duas categorias principais: *host-based* - sistemas baseados em *hosts*, e *network-based* - sistemas baseados em rede (Mukherjee e Levitt, 1994). Os sistemas de detecção de intrusão *host-based* utilizam os registros de auditoria como principal conjunto de informações na detecção de intrusões. Aqui, as informações relevantes são tempo de uso da CPU, número de acessos ao sistema via terminal remoto, tentativas de *login*, utilização de recursos do sistema, etc.

Os sistemas de detecção de intrusão *network-based* utilizam as informações da rede como principal fonte de detecção de intrusões. Utilizam informações como tráfego, acesso, fluxo de dados, cabeçalho e conteúdo de pacotes, além de poderem utilizar os registros de

auditoria das máquinas que compõem a rede, para complementar informações ou eliminar casos duvidosos.

4.2.5 O Estado da Arte em Sistemas de Detecção de Intrusões

Denning (1987) desenvolveu um modelo genérico de detecção de intrusão, independente de qualquer sistema em particular, ambiente de aplicações, vulnerabilidade de sistema ou tipo de intrusão. O modelo é baseado na hipótese de que violações de segurança podem ser detectadas através do monitoramento dos registros de auditoria, relacionados a padrões anormais de uso do sistema. A idéia básica do modelo é manter um conjunto de perfis para os *subjects* (sujeitos), que são os iniciadores de atividades no sistema (normalmente usuários). Quando um registro de auditoria é gerado, o modelo o associaria a um determinado perfil e, então, toma decisões a respeito da atualização do perfil, da checagem de comportamento anormal e do relato de anomalias detectadas. Para isto, *logins*, comandos, execução de programas, acesso a arquivos e dispositivos são monitorados em busca de desvios no uso. O modelo não tem conhecimentos específicos, relacionados às vulnerabilidades do sistema.

O protótipo de sistema de detecção de intrusões IDES (*Intrusion-Detection Expert System*) baseou-se neste modelo. Desenvolvido pelo *SRI International's Computer Science Laboratory* (CSL), no final da década de 80, era capaz de prover detecção de violações de segurança, em tempo real, para um único *host*. Este sistema foi um marco no desenvolvimento da tecnologia de detecção de intrusão híbrida (análise de assinaturas e detecção por anomalias), em tempo real (SRI, 2002).

O protótipo consiste de dois componentes principais. O detector estatístico de anomalias (IDES/STAT) utiliza um processo de dedução baseado em estatísticas para determinar se o comportamento, como relatado nos registros de auditoria, é normal em relação a um comportamento aceitável, baseado no perfil histórico de atividades do usuário. Estes perfis históricos são continuamente atualizados, utilizando os dados extraídos dos registros de auditoria, para aprender sobre o comportamento esperado dos usuários de um sistema. Desta forma, provê mecanismos para relatar e concluir sobre violações de segurança, bem como detectar intrusões que não podem ser detectadas pelos controles de acesso do sistema a ser monitorado. O sistema especialista (PBEST), outro componente do protótipo, contém regras que descrevem comportamentos suspeitos, baseadas no conhecimento de intrusões passadas, vulnerabilidades conhecidas do sistema, ou políticas de segurança específicas do domínio. Os registros de auditoria do sistema monitorado são associados a estas regras, para determinar se o comportamento é suspeito. Entretanto, não é razoável esperar que todas as situações de intrusão possam ser codificadas no formato de regras.

Os dois componentes (estatístico e baseado em regras) são independentes, e a combinação destes teria um poder de detecção bem superior ao caso isolado (Lunt et al., 1992).

Em meados dos anos 90, a SRI *International* iniciou trabalhos para aperfeiçoar, otimizar e realizar reengenharia no protótipo IDES, com o intuito de desenvolver um sistema de detecção de intrusão de qualidade, chamado de *Next-Generation Intrusion Detection Expert System* (NIDES). O NIDES introduziu um componente de fusão de resultados, chamado de *Resolver*, para integrar sua lógica de respostas aos resultados produzidos pelo componente estatístico de detecção de anomalias e a ferramenta de análise de assinaturas, baseada em regras (SRI, 2002).

Algumas das mudanças realizadas para o projeto do NIDES incluíram a facilidade de personalizar a análise, de modo que o componente estatístico podia ser ajustado e parâmetros específicos podiam ser habilitados ou desabilitados; a base de regras podia ser personalizada, habilitando ou desabilitando regras, além de permitir a introdução de novas regras no sistema em execução. Um módulo de testes foi adicionado, onde configurações candidatas do NIDES eram executadas "fora" do *software* NIDES rodando em tempo real. Recursos de arquivamento foram adicionados, suportando a armazenagem e recuperação de registros de auditoria, bem como dados de resultados (Anderson et al., 1995).

Os projetos IDES/NIDES marcaram o campo da detecção de intrusão, e tentaram resolver problemas relacionados com uma abordagem genérica e flexível, sem restrições impostas pelos sistemas, tipos de registros de auditoria a serem analisados, ou técnicas a serem utilizadas. Estes projetos estavam focados na necessidade de monitoramento e criação de perfis específicos dos usuários. Estes esforços, entretanto, apresentaram algumas limitações relacionadas a escalabilidade, aplicabilidade em ambientes baseados em rede, por estarem voltados para usuários como alvos de análise, bem como falhas no suporte à interoperabilidade.

Outra linha de pesquisas em detecção de intrusão vem sendo desenvolvida pela *Purdue University*, onde foi proposta uma arquitetura para detecção de intrusão, usando agentes autômonos. Diferente de outros SDIs, que são normalmente "grandes", este é um sistema de detecção distribuído, baseado em várias entidades independentes, trabalhando em conjunto. Estas entidades são conhecidas como agentes autônomos, e são *softwares* que realizam funções de monitoramento de segurança em um *host*. O sistema é conhecido como *Autonomous Agents For Intrusion Detection* (AAFID) e contém três componentes principais: *agents* (ou agentes), *transceivers* e *monitors*. Este pode ser distribuído sobre qualquer número de *hosts* em uma rede. Cada *host* contém um certo número de *agents* monitorando a ocorrência de eventos de interesse. Todos os *agents* em um determinado *host*

enviam relatórios para um único *transceiver*. Os *transceivers* são entidades relacionadas a um conjunto de *hosts*, que supervisionam a operação de todos os *agents* a estes relacionados. Eles exercem controle sobre os agentes, podendo iniciá-los, pará-los ou configurá-los. Estas entidades enviam relatórios para um ou mais *monitors*. Os *monitors* têm acesso aos dados na rede e, portanto, estão aptos a correlacionar e detectar intrusões envolvendo vários *hosts*. *Monitors* podem ser organizados hierarquicamente, de modo que um envie relatórios a outro de nível mais alto (Balasubramaniyan et al., 1998). Esta abordagem apresenta vantagens relacionadas à eficiência, tolerância a falhas, flexibilidade quanto à degradação e escalabilidade. Alguns dos problemas são: *monitors* são pontos críticos de falhas, e caso algum falhe, todos os *transceivers* controlados por ele param de transferir informações; se *monitors* são duplicados para prover redundância, deve haver um esforço para manter os dados de ambos consistentes, e atrasos na detecção de intrusões podem ocorrer, devido ao cascateamento (dividido em níveis) do modelo.

Vários sistemas de detecção de intrusão foram implementados e testados. Alguns destes baseados em detecção por anomalia, outros baseados em detecção por abuso, e ainda outros que combinam as duas técnicas (sistemas híbridos). Estes ainda são classificados de acordo com o tipo de análise que realizam, ou seja, *host-based* ou *network-based*. Em COAST (2002) há uma lista de SDIs, contendo uma breve descrição, técnica utilizada, tipo de análise, dentre outras informações.

4.2.6 Sistemas de Detecção de Intrusão Baseados em Rede

Os modelos de sistemas de detecção de intrusões anteriores foram projetados para operar em *host*. Entretanto, os sistemas de detecção de intrusão atuais, além de utilizarem as técnicas de detecção por anomalia e por abuso, utilizam-nas para efetuar a monitoração de um ambiente de rede, utilizando os dados da rede no processo de detecção. Alguns dos sistemas que incorporam estas características são descritos próximas seções.

4.2.6.1 *Network Security Monitor*

Desenvolvido pela Universidade da Califórnia, o *Netwok Security Monitor* (NSM) analisa o tráfego em uma rede local para detectar comportamentos intrusivos (Mukherjee e Levitt, 1994). Este modela a rede e os *hosts* sendo monitorados em uma estrutura hierárquica, conhecida como *Interconnected Computing Environment Model* (ICEM), composta por seis camadas.

A camada mais baixa é conhecida como camada de pacote, e recebe como entrada um fluxo de bits vindos de uma rede *broadcast*, por exemplo *Ethernet*. O fluxo de bits é dividido em pacotes *Ethernet* completos e um *timestamp* é adicionado ao pacote. A pró-

xima camada, chamada de *thread layer*, recebe o pacote da camada anterior e, então, efetua a correlação entre pacotes para um fluxo de dados unidirecional. Cada fluxo consiste dos dados transferidos de um *host* para o outro, através de um protocolo particular (TCP ou UDP), utilizando um conjunto único de portas. Este fluxo de dados, chamado de *thread*, é mapeado para um vetor, conhecido como *thread vector*. A terceira camada (*connection layer*) recebe os vetores da camada anterior. Cada *thread vector* é combinado com outro para representar um fluxo bidirecional de dados. Estes pares de vetores são representados por um vetor de conexão (*connection vector*). Cada *connection vector* é analisado e reduzido. A quarta camada (*host layer*) recebe os vetores reduzidos gerados pela camada anterior. Estes vetores são utilizados na construção de vetores de *hosts* (*host vectors*). Cada *host vector* representa as atividades de rede de um *host*. A próxima camada (*connected-network layer*) recebe os *host vectors* e os transforma em um grafo G. Se G(*host1, host2, serv1*) não é vazio, há uma conexão do *host1* para o *host2* pelo serviço *serv1*. O valor para a posição G(*host1, host2, serv1*) é não vazia se o *host vector* para o *host1* possui (*host2, serv1*) em seu caminho de conexão. A última camada (*system layer*) utiliza os *connected-network vectors* para construir um único vetor, representando o comportamento do sistema como um todo.

O tráfego na rede é analisado por um sistema especialista simplificado, que trata de alguns tipos de entradas. O primeiro tipo corresponde aos vetores do modelo ICEM. Nesta implementação apenas os *connection vectors* e os *host vectors* são usados. O segundo tipo consiste dos perfis de comportamento esperado do tráfego. A combinação dos perfis com o tráfego atual permite que o NSM detecte comportamento anômalo na rede. O terceiro tipo representa o conhecimento sobre as capacidades oferecidas por cada serviço de rede disponibilizado. O quarto tipo consiste do nível de autenticação requerido por cada serviço. O quinto tipo corresponde ao nível de segurança de cada uma das máquinas e o sexto tipo corresponde às assinaturas de ataques passados.

Os dados destas fontes são usados para identificar se uma conexão em particular representa um comportamento intrusivo, ou se um *host* foi comprometido. A avaliação da segurança de um *host*, relacionado a uma conexão em particular é função de quatro fatores: anormalidade da conexão, nível de segurança do serviço usado, nível de sensibilidade em relação à direção da conexão e as assinaturas de ataques reconhecidas no fluxo de dados daquela conexão (Mukherjee e Levitt, 1994).

Algumas vantagens deste sistema são: o NSM pode monitorar um conjunto heterogêneo de máquinas e sistemas operacionais, por utilizar protocolos de rede padronizados, como UDP e TCP; o processo de detecção é ágil, devido à natureza das redes locais *broadcast*, que permitem acesso aos dados assim que transmitidos pela rede, e o sistema monitora

passivamente a rede e está logicamente protegido, pois é "invisível" e seus dados não podem ser modificados. Algumas desvantagens são: implementação de todo o modelo é complexa; a especificação de limiares de detecção deve ser feito com muita cautela, e só é possível detectar as assinaturas de ataques codificadas.

4.2.6.2 *Distributed Intrusion Detection System*

Outro marco na pesquisa em detecção de intrusão foi o sistema conhecido como *Distributed Intrusion Detection System* (DIDS). Este sistema é um aprimoramento do NSM e foi desenvolvido por Snapp et al. (2000) para suprir as falhas apresentadas naquele sistema. A arquitetura do DIDS combina monitoramento distribuído e redução de dados, sendo que a análise dos dados é centralizada. Os componentes do sistema são *DIDS director*, *host monitor* (por *host*) e *LAN monitor* (por cada segmento da rede local, na rede monitorada). Os componentes *host monitor* e *LAN monitor* são responsáveis pela coleta de evidências relacionadas à atividades suspeitas ou não autorizadas. Por outro lado, o *DIDS director* é responsável pela avaliação dos dados coletados.

O *host monitor* coleta e analisa registros de auditoria do *host*, que representam eventos de interesse. Estes eventos são, então, enviados para o *DIDS director*, para análise posterior. O *LAN monitor* observa todo o tráfego em seu segmento da rede local, para monitorar conexões do tipo *host-a-host*, serviços usados e volume do tráfego. Ele relata atividades de rede, como *rlogin* e *telnet*, o uso de serviços relacionados a segurança e mudanças nos padrões do tráfego da rede. O *DIDS director* obtém os dados de cada *host* e dos *LAN monitors* e os envia para o sistema especialista. O sistema especialista é responsável por avaliar e relatar o estado de segurança do sistema monitorado. Ele recebe relatórios dos *hosts* e *LAN monitors* e, baseado nestes, faz inferências sobre a segurança de cada *host*, bem como do sistema como um todo. O sistema especialista é baseado em regras, sendo que estas regras são derivadas de um modelo de detecção de intrusão, denominado *Intrusion Detection Model* - IDM (Smaha, 1988). A interface do *DIDS director* permite que o *System Security Officer* (SSO), ou analista de segurança, tenha acesso interativo a todo o sistema.

4.2.6.3 *Event Monitoring Enabling Responses to Anomalous Live Disturbances*

Sucessor do NIDES, o *Event Monitoring Enabling Responses to Anomalous Live Disturbances* (EMERALD) é uma ferramenta distribuída e escalável, para rastrear atividades mal intencionadas em redes de grande porte. Sua arquitetura utiliza monitores de vigília e resposta altamente distribuídos, ajustáveis de forma independente, que são empregados em várias camadas abstratas na rede. É composta por um conjunto de uni-

dades interoperáveis de análise e resposta (monitores), que provêem proteção localizada de ativos chaves em uma rede.

Os monitores são computacionalmente independentes, provendo assim um grau de paralelismo na cobertura de análise, enquanto ajuda na distribuição da carga computacional e utilização de espaço em disco. Através do emprego de monitores locais aos alvos de análise, o EMERALD ajuda a reduzir possíveis atrasos na análise e resposta, que podem ocorrer ao longo da rede. Também introduz um esquema de análise hierárquica, onde análises locais são compartilhadas e correlacionadas em camadas de abstração mais altas.

O esquema de análise é iniciado a partir da camada de interface de rede, de domínios administrativos individuais. Os monitores são empregados por todo o domínio e em cada um dos domínios, para analisar a operação dos serviços de rede e outros componentes externamente acessíveis. Cada monitor contém um conjunto específico de manipuladores de respostas, que são invocadas ao detectar um possível abuso. Estes monitores da camada de serviço (*service-layer*) também disseminam suas análises para outros monitores do EMERALD, que realizam a correlação para o domínio. Os monitores de domínio provêem uma perspectiva mais global, em relação a geração de perfis e modelagem de vulnerabilidades, que podem ocorrer de interdependências entre serviços da rede e outros ativos dentro do domínio. Finalmente, o EMERALD implementa uma análise para correlacionar os relatórios de atividades produzidos através do conjunto de domínios monitorados. Os monitores da camada *Enterprise-layer* estão focados em ameaças a rede, tais como *Internet worms*, ataques repetidos contra serviços de redes comuns através dos domínios, ou ataques coordenados a partir de vários domínios contra um único domínio. Através da correlação e compartilhamento de resultados de análise, relatórios de problemas encontrados por um monitor podem propagar para outros monitores e para toda a rede.

O monitor foi projetado para ocupar pouco espaço, ser bem rápido e genérico o bastante para ser empregado em qualquer camada no esquema hierárquico de análise do EMERALD. O monitor opera como um detector de intrusão descentralizado, que combina análise por assinatura com a geração de perfis estatísticos, provendo proteção localizada, em tempo real, da infraestrutura e dos serviços da rede. O monitor incorpora uma API (*application program interface*) que melhora a capacidade de interoperabilidade com o alvo de análise e com outras ferramentas de detecção de intrusão.

O EMERALD representa um grande avanço, em relação às pesquisas anteriores e o desenvolvimento de detecção por abuso e anomalia, para acomodar o monitoramento de grandes sistemas distribuídos e redes. Devido ao fato de análise em tempo real poder

ser distribuída e aplicada onde for mais efetiva, em diferentes camadas de abstração, o EMERALD apresenta vantagens significantes sobre abordagens mais centralizadas, em termos de capacidade de detecção e resposta a eventos. Este pretende detectar não apenas ataque locais, mas também ataques coordenados como *denial-of-service* distribuído ou padrões de ataque repetidos contra vários domínios (Porras e Neumann, 2002).

4.2.6.4 *Advanced Counter-Measures Environment*

(Cansian, 1997) desenvolveu um sistema adaptativo de detecção de intrusão (SADI), que foi a primeira referência para a aplicação do método de detecção de intrusão por abuso, utilizando reconhecimento de padrões associado à redes neurais. O *Advanced Counter-Measures Environment* (ACME!)[1] constitui a continuação deste trabalho.

Este SDI baseia-se em um modelo de detecção, composto por um sistema de captura e tratamento de pacotes, um sistema de rede neural, e um gerenciador de comunicações e interface com o usuário.

O sistema de captura e tratamento de pacotes é modularizado, de forma que seu módulo de mais baixo nível apenas captura o fluxo de dados no tráfego de rede e envia os pacotes ordenados para o módulo de conexão, que por sua vez, está sob controle do módulo de pré-seleção e sistema especialista.

O módulo de pré-seleção e sistema especialista (PSSE) decide se uma conexão é considerada suspeita. Neste caso, um conjunto procedimentos é executado simultaneamente. Este é responsável por mais um nível de filtragem, pois o módulo de conexão só inicia o registro das atividades de *hosts* ou domínios considerados suspeitos, quando indicado pelo módulo de pré-seleção e sistema especialista.

O módulo de conexão (CON) é responsável pela criação e manutenção de vetores de conexão, que constituem a representação contendo a reconstrução do fluxo de dados. Um vetor de conexão armazena os dados que trafegam em uma dada conexão, a partir do momento em que ela foi considerada suspeita. Este é composto por informações de controle, tais como portas, endereços, tipos de *frame*, até os dados efetivamente transportados. Esse módulo é acionado pelo módulo de pré-seleção e sistema especialista, que transmite os dados necessários para a criação do vetor de conexão.

Após a construção do vetor de conexão, é criado o vetor de estímulo, que corresponde à codificação binária do vetor de conexão. Este vetor de estímulo é, então, enviado para o módulo de rede neural. A interface da rede neural recebe o conjunto de bits que compõe

[1]Página Web em: `http://www.acme-ids.org/acme-ids/`.

o vetor de estímulo e, baseado no treinamento ao qual a rede foi submetida, retorna um valor numérico, que indica o grau de suspeita da sessão.

A cada atualização de dados para uma conexão, o sistema efetua uma nova análise do estado alcançado pelo vetor de estímulo, fornecendo um coeficiente de nível de intrusão, através do módulo de rede neural. Este novo valor substitui o antigo coeficiente no arquivo de saída de dados. Este é utilizado pela interface do usuário e contém a relação de todas as conexões monitoradas e suas respectivas informações. A seguir, verifica-se se o coeficiente relacionado a cada conexão ultrapassou o limiar de intrusão permitido. Caso positivo, o sistema executa um módulo para a realização de ações, que podem ser pré-programadas pelo administrador do sistema.

4.2.6.5 *SANS Heuristic Analysis system for Defensive Online Warfare*

O *System Administration Networking and Security* (SANS[2]) *Institute* anunciou, em julho de 1998, uma ferramenta de monitoramento de rede, de domínio público: o *SANS Heuristic Analysis system for Defensive Online Warfare* (SHADOW). Parte do projeto, intitulado *Cooperative Intrusion Detection Evalution and Response* (CIDER), é um sistema de detecção de intrusão, que pode ser construído através da utilização de *softwares* de domínio público disponíveis e *hardware* existente ou de baixo custo. A discussão a seguir baseia-se em Northcutt (1999) e Stocksdale (2002).

A idéia básica do sistema SHADOW é que existe uma história a ser contada pelos pacotes trafegando nas redes. Dentro do fluxo de dados esperado, podem ocorrer pacotes mal intencionados, contendo algumas formas de ataques. Sem um sistema de detecção de intrusão, seria provavelmente impossível detectar estas atividades, para determinar sua origem, observar padrões, ou configurar alarmes quando eventos semelhantes fossem detectados. O sistema é, então, projetado para ter componentes em localizações estratégicas (dentro ou fora de um *Firewall*), coletando dados associados ao tráfego da rede. Estes dados são posteriormente movidos para um módulo de análise, para avaliação.

A arquitetura do SHADOW é composta por dois sistemas: uma máquina de coleta de dados, conhecida como sensor, e uma estação de análise. O sensor é situado na rede de interesse (alvo) e captura parte ou todo o tráfego. Os dados brutos (*raw*) são movidos, através de um canal seguro, para a estação de análise. A implementação é baseada na utilização de duas ferramentas originalmente desenvolvidas pelo *Lawrence Berkeley Laboratory* (LBL[3]): *libpcap* (vista na seção 3.2) e o *tcpdump* (visto na seção 3.3).

[2]Página Web em `http://www.sans.org`.
[3]Página Web em `http://www.lbl.gov`.

Basicamente, o SHADOW é constituído por um conjunto de *scripts* na linguagem de programação PERL (Wall et al., 1996), que realizam as seguintes tarefas:

- Criam arquivos para onde serão copiados os dados coletados;
- Iniciam e terminam processos *tcpdump*, relacionados à coleta do tráfego da rede;
- Transferem, utilizando um processo seguro, os dados coletados para análise;
- Ordenam e agrupam os dados e os disponibilizam em páginas Web;
- Realizam análises diferenciadas sobre os dados coletados.

O sistema sensor é constituído pelos *scripts* que coletam os dados na rede, descritos como se segue:

- *sensor_driver.pl*: este *script* é executado periodicamente na máquina que funciona como sensor. Ele recebe um argumento, apontando para um arquivo de configurações. Este arquivo contém informações, como localização do aplicativo *tcpdump* e seus parâmetros, localização da aplicação utilizada na compactação dos dados coletados (*gzip*), entre outras. Além disso, utiliza um arquivo contendo os filtros (expressões booleanas) a serem aplicados pelo processo de coleta de dados. O *script* faz chamadas para outros dois *scripts*. Um inicia e o outro termina o processo *tcpdump*. Como o *sensor_driver.pl* é periódico, ao ser chamado novamente, pára o processo *tcpdump* anterior e inicia um novo processo;
- *start_logger.pl*: este script tem como parâmetros os arquivos de configuração e de filtros, utilizados pelo *script sensor_driver.pl*. Ele cria o nome do arquivo para onde serão enviados os dados brutos, coletados da rede, baseado na data e hora atuais. Então, inicia o processo *tcpdump*, passando como parâmetro o arquivo de filtros, a interface de rede, um arquivo de *log* (para mensagens de status e erros), e direcionando sua saída para o arquivo com nome no formato data-hora. Enquanto este processo é executado, o arquivo de saída é comprimido, visando a redução do espaço ocupado;
- *stop_logger.pl*: este *script* interrompe o processo *tcpdump* e, então, fecha o arquivo contendo os dados brutos coletados.

Como o sistema sensor, o analisador é constituído por um conjunto de *scripts*, mas que realizam atividades diferentes. Deve-se observar que o analisador precisa habilitar um

servidor Web (como o Apache[4]), pois seus resultados são disponibilizados em formato hipertexto, e precisam estar atualizados. Além de uma página que dá acesso as páginas de resultado, o analisador do SHADOW dispõem de outras facilidades, acessíveis também no formato hipertexto, que auxiliam o *System Security Officer* no rastreamento de um determinado domínio, no relato de um incidente de segurança, dentre outras. Os *scripts* são:

- *fetchem.pl*: este script é executado periodicamente e realiza um sequência de atividades. Ele copia o último arquivo compactado de dados brutos gerado no sensor, utilizando um canal seguro. Este canal seguro é provido pelo Secure Shell - SSH[5] . O arquivo é, então, descompactado e inicia-se o processo *tcpdump* para a leitura dos dados brutos. Este processo utiliza arquivos, contendo filtros ou padrões, a serem reconhecidos nos dados brutos. O resultado desta filtragem é enviado para um arquivo texto temporário. O próximo passo é ordenar e agrupar os dados, de acordo com o endereço de origem, que pode ser obtido do cabeçalho dos pacotes. Além disso, neste mesmo passo, tenta-se resolver os endereços IP dos pacotes em nomes. Deve-se observar que este processo é demorado e pode ser excluído. Finalmente, o resultado é armazenado em uma página, no formato hipertexto, e disponibilizada pelo servidor Web;

- *cleanup.pl*: este *script* acessa o sistema sensor através de um canal seguro e remove os arquivos de dados brutos armazenados a mais de **n** dias, sendo que n pode ser configurado. Esta medida procura evitar que o sensor fique sem espaço de armazenamento.

Para contornar o problema de detecção de ataques pequenos e lentos, como determinados tipos de varreduras de rede, foi desenvolvido um conjunto de *scripts* para este tipo de análise, descritos a seguir:

- *one_day_pat.pl*: procura por um padrão (no formato *tcpdump*) específico, nos arquivos correspondentes a um dia, utilizando a linha de comandos;

- *one_day_script.pl*: procura por padrões nos arquivos correspondentes a um dia, utilizando um filtro pré-definido, no diretório que contém os arquivos de filtragem (especificam as expressões booleanas no formato *tcpdump*);

- *pat_match.pl*: procura em um arquivo de dados brutos específico, por um padrão especificado na linha de comandos.

[4]Página Web em `http://www.apache.org`.
[5]Página Web em `http://www.openssh.org`.

O sistema de detecção de intrusão SHADOW reconhece padrões no tráfego de rede, baseado em filtros construídos por um analista, para identificar atividades anômalas, tais como *Land attacks*, *Ping of Death*, dentre outros. Os analistas devem modificar e adicionar novos filtros que correspondam às necessidades de seu domínio. Finalizando, analistas com conhecimento no desenvolvimento dos filtros são a chave das implementações bem sucedidas deste sistema de detecção de intrusão.

4.2.6.6 *Snort*

O *Snort*, desenvolvido por (Roesch, 1999), é um sistema de detecção de intrusão de domínio público, originalmente projetado para ambientes de rede com tráfego moderado. Ele se baseia na biblioteca *libpcap* para realizar a captura de pacotes de rede e dispõe de um conjunto de regras, utilizado no reconhecimento de padrões no conteúdo dos pacotes capturados. As regras disponíveis permitem a detecção de uma grande variedade de ataques, varreduras e atividades relacionadas à exploração de vulnerabilidades de aplicações, além de ser capaz de gerar alertas em tempo real.

A arquitetura do *Snort* é dividida em três componentes principais: o decodificador de pacotes, o mecanismo de detecção e o sistema de registro de eventos (*logging*) e geração de alertas.

O decodificador de pacotes é composto por um conjunto de subrotinas, que atuam sobre os dados capturados para cada pacote. Cada subrotina corresponde a uma camada da pilha de protocolos TCP/IP e cuida da decodificação da parte relacionada do pacote. A principal função do decodificador é, então, estabelecer apontadores para segmentos do pacote, de modo que estes segmentos possam ser posteriormente analisados pelo mecanismo de detecção.

O *Snort* mantém as regras de detecção em listas encadeadas, conhecidas como *Chain Headers* e *Chain Options*. Cada elemento da *Chain Headers* é constituído por atributos comuns a um constituído por regras, e mantém uma lista do tipo *Chain Options*, onde cada elemento, por sua vez, é constituído por opções de detecção diferenciadas para cada regra do conjunto correspondente. Esta modelagem tem como finalidade melhorar o desempenho do processo de detecção. As listas encadeadas são, então, checadas recursivamente para cada pacote, de modo que a primeira regra que confere com um pacote decodificado no mecanismo de detecção dispara uma ação especificada na definição da regra.

O sistema de registro de eventos e geração de alertas permite que *logs* e alertas sejam gerados em diferentes formatos, que são selecionados no momento em que o *Snort* é

inicializado, através de argumentos da linha de comandos. Pacotes podem ser registrados em seu formato decodificado e legível, e armazenados em uma estrutura de diretórios baseada em endereços IP, no formato binário do *tcpdump* em um único arquivo, ou até em uma base de dados. O primeiro formato de armazenamento permite uma análise mais rápida dos dados coletados pelo sistema, ao passo que o segundo apresenta um melhor desempenho no processo de armazenagem em disco e, portanto, é indicado em situações onde alta performance é necessária. Já os alertas podem ser enviados para o processo *syslog*, para arquivo-texto em diferentes formatos, ou para a tela, através de janelas de diálogo.

A arquitetura modularizada do *Snort* permite que outros módulos (*plugins*) sejam desenvolvidos e acoplados ao sistema, no intuito de adicionar funcionalidades de detecção ou personalizá-lo para realização de tarefas específicas.

Quando um módulo é caracterizado como um preprocessador, este atua entre o decodificador de pacotes e o mecanismo de detecção. Desta forma, uma de suas principais funções é preparar um ou mais pacotes decodificados para serem analisados pelo mecanismo de detecção. Dentre os preprocessadores atualmente disponíveis destacam-se o *Frag2* e o *Stream4*. O primeiro permite remontar datagramas IP fragmentados. Já o segundo provê a capacidade de remontagem de fluxo de dados (*stream reassembly*) e de rastreio e análise de estados para o TCP (Roesch e Green, 2002).

CAPÍTULO 5

O SISTEMA DE RECONSTRUÇÃO DE SESSÕES TCP/IP

Este capítulo apresenta o sistema de reconstrução de sessões TCP/IP, conhecido como *RECON*. São apresentadas algumas considerações, relacionadas ao seu desenvolvimento, e é apresentada e discutida a modelagem criada para representar as sessões TCP/IP, baseada em Chaves e Montes (2001). Também é apresentada e discutida a implementação detalhada de todo o sistema.

5.1 CONSIDERAÇÕES GERAIS SOBRE O DESENVOLVIMENTO DO SISTEMA

O desenvolvimento deste sistema parte de algumas considerações, referentes à obtenção dos pacotes de rede. As próximas seções apresentam uma discussão sobre estratégias de captura de pacotes e suas implicações, bem como a metodologia adotada para a análise dos dados obtidos do tráfego de redes TCP/IP.

5.1.1 Estratégias de Captura de Pacotes

A captura de dados para um sistema de monitoramento de redes, ou mesmo para sistemas de detecção de intrusão, é realizada por um elemento de rede normalmente conhecido como *sensor*.

Existem várias possibilidades para o posicionamento deste dispositivo na rede sendo monitorada e que influenciam diretamente na eficiência e utilidade dos resultados sendo gerados pelo monitor de rede ou detector de intrusões, a partir da análise dos dados coletados. A Figura (5.1) apresenta as formas mais utilizadas no posicionamento de sensores de rede.

Na primeira forma de posicionamento, apresentada na ilustração **(a)** da Figura (5.1), o sensor é colocado fora dos limites de proteção do *Firewall*, de modo que todo o tráfego destinado à rede monitorada é selecionado para a coleta de pacotes. Esta configuração permite a observação de ataques direcionados ao *Firewall* e a recursos protegidos por este. Em contrapartida, a quantidade de dados a serem tratados é consideravelmente maior.

Na ilustração **(b)**, o sensor, posicionado dentro dos limites de proteção do *Firewall*, observa apenas o tráfego permitido e destinado à rede monitorada, eliminando assim a coleta de todos os pacotes bloqueados. Desta forma, a quantidade de dados a serem tratados reduz razoavelmente.

E finalmente, na ilustração **(c)** são colocados dois sensores: um dentro dos limites de proteção do *Firewall* e outro fora. Além das características apresentadas para as ilustrações anteriores, esta abordagem permite, através da comparação do tráfego coletado pelos sensores em ambos os lados do *Firewall*, validar as regras de controle deste dispositivo (McHugh et al., 2000). A quantidade de dados a serem tratados para este caso é semelhante ao **(a)**.

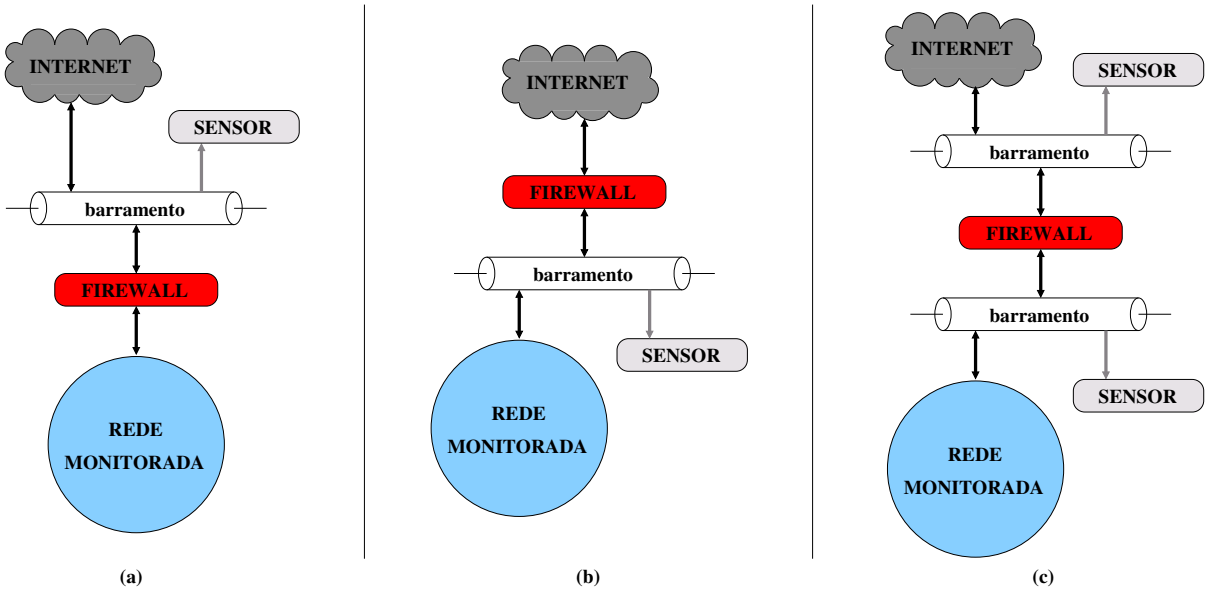

FIGURA 5.1 – Posicionamento do sensor de rede.

Deve-se observar que para todas a ilustrações apresentadas, o sensor de rede está conectado a um barramento, o que viabiliza a observação de todo o tráfego passando pelo ponto de monitoramento em questão. Outra observação importante refere-se ao sentido unidirecional da seta, que representa o tráfego indo para o sensor. Isto quer dizer que o sensor é passivo e não interage com outros dispositivos, a não ser para eventuais transferências de dados coletados.

5.1.2 Metodologia Adotada para Captura e Análise de Dados

Algumas considerações e procedimentos foram adotados no desenvolvimento do sistema de reconstrução de sessões *RECON*, e dizem respeito a forma de captura dos pacotes no tráfego de rede e de sua posterior análise.

O *RECON* foi projetado para analisar apenas os cabeçalhos dos pacotes da pilha de protocolos TCP/IP, onde são considerados:

- da camada de enlace de dados, o protocolo *Ethernet*;
- da camada de rede, os protocolos IP e ICMP;
- e da camada de transporte, os protocolos UDP e TCP.

Desta forma, o componente sensor do sistema de detecção de intrusão *SHADOW*, descrito na seção 4.2.6.5, foi utilizado como dispositivo de captura de pacotes, pois este coleta apenas 68 bytes por pacote do tráfego de rede. Este padrão para a quantidade de dados sendo capturados normalmente engloba todos os cabeçalhos descritos anteriormente.

Foi adotada, então, a metodologia de análise *offline* (ou *pos-mortem*) dos dados. O tráfego de rede é coletado pelo sensor e armazenado em arquivos periodicamente. Estes arquivos são transferidos para uma estação de análise, onde o *RECON* é executado, analisando assim os pacotes TCP/IP obtidos dos arquivos em questão.

Sabe-se que a análise de dados em tempo real é amplamente utilizada na detecção de padrões de ataques conhecidos. Em contrapartida, a análise *offline* permite identificar novas tendências, por meio da detecção de anomalias, e avaliar como os ataques estão se comportando e evoluindo, pois pode-se englobar na análise intervalos de tempo consideravelmente maiores, fazendo uso de pacotes previamente coletados e armazenados. Estas características reduzem a preocupação com problemas relacionados à exaustão de recursos do sistema.

Algumas extensões da metodologia adotada para a captura de pacotes de rede são propostas no Capítulo 7.

5.2 MODELAGEM DAS SESSÕES TCP/IP

Um modelo pode ser visto como a representação de algo que se quer reproduzir ou imitar. Especificamente, um modelo de dados pode ser visto como a representação de um conjunto de informações de forma estruturada, em um sistema computacional, baseado em um conjunto de premissas. Estas premissas, bem como a forma de representação do modelo criado para representar as sessões TCP/IP, são descritas a seguir.

Faz-se necessário estabelecer uma definição concisa para sessão TCP/IP, já que o o objetivo do sistema *RECON* é obter informações dos cabeçalhos dos pacotes e a partir destas informações, reconstruir e inferir sobre o estado das "sessões TCP/IP" contidas no tráfego de rede sendo analisado.

"Uma sessão TCP/IP é definida por qualquer sequência de pacotes, que caracterize a troca de informações entre dois endereços IP, e que tenha início, meio e fim (mesmo que

toda a comunicação esteja contida em um único pacote)."

Na literatura, é comum encontrar referências onde é feita a analogia entre o conceito de sessão e conexão utilizando o protocolo TCP, por ser este um protocolo orientado à conexão. Esta analogia é natural, pois no protocolo TCP o início, meio e fim de uma conexão são bem definidos e facilmente identificáveis. Já para protocolos não orientados à conexão, como o UDP e o ICMP, faz-se necessário extrapolar o conceito normalmente utilizado para a definição de sessão. Portanto, nessa modelagem o conceito de sessão foi dividido em três categorias:

Sessão TCP corresponde a conexão TCP (seção 2.6). Cada sessão TCP é identificada pelo seguinte conjunto de informações: endereço IP de origem e porta de origem, endereço IP de destino e porta de destino. Como podem haver recorrências deste conjunto de informações no tráfego de rede, os pacotes que caracterizam o início e o término de uma conexão TCP (seção 2.6.1) são utilizados para delimitar as sessões.

Sessão ICMP existem dois formatos de mensagens ICMP: mensagens de informação e mensagens de erro. A comunicação utilizando mensagens ICMP de informação ocorre através de requisições e suas respectivas respostas. Portanto, uma sessão ICMP de informação é definida pelo conjunto de pacotes de requisição e de respostas, caracterizadas pelas seguintes informações: endereço IP de origem, endereço IP de destino e o campo "*identifier*" do cabeçalho da mensagem ICMP de informação. Já as mensagens ICMP de erro normalmente estão associadas a alguma falha ou problema na comunicação entre *hosts* ou dispositivos da rede. Como visto na seção 2.4.1, uma mensagem ICMP de erro não deve ser gerada como resposta a outra mensagem de erro, portanto não é definido uma sessão para este formato de mensagem. Cada mensagem ICMP de erro é apenas associada ao pacote gerador do erro, que pode ser um pacote TCP, UDP ou ICMP de informação.

Sessão UDP corresponde ao conjunto de pacotes, identificados pelo seguinte conjunto de informações: endereço IP de origem e porta de origem, endereço IP de destino e porta de destino. Apesar de haver uma correspondência com as informações utilizadas na caracterização da sessão TCP, não existe uma forma de delimitar as sessões. O cabeçalho UDP não fornece informações suficientes para identificar o início e o término de uma comunicação. Normalmente estas informações só podem ser encontradas no conteúdo dos pacotes. Portanto, todos os pacotes no tráfego de rede, caracterizados pelo conjunto de informações inicialmente descrito, são associados a uma mesma sessão UDP.

Para definir a metodologia adotada na representação do modelo de reconstrução de sessões, é preciso inicialmente especificar uma forma de representação para o tráfego de rede. Esta representação é apresentada na próxima seção (5.2.1).

5.2.1 Representação utilizando Grafos

O tráfego de rede constitui a matéria-prima, ou seja, os dados lidos e modelados (ou representados) na forma de sessões TCP/IP. Este pode ser visto como um conjunto de endereços IP (representando os *hosts* ou dispositivos conectados à rede)[1] e a comunicação (ou transferência de dados) entre cada par de endereços deste conjunto. Daí vem a facilidade em estabelecer uma relação entre os elementos que constituem o tráfego de rede e o conceito de grafos.

Segundo Gibbons (1985), um grafo consiste de um conjunto de vértices interconectados por um conjunto de arestas. Para um grafo G, denota-se o conjunto de vértices por V e o conjunto de arestas por E, sendo $G = (V, E)$. Uma aresta no grafo G pode ser especificada por dois vértices que esta conecta. Se a aresta e conecta os vértices v_i e v_j, então denota-se $e = (v_i, v_j)$ ou $e = (v_j, v_i)$. Além disso, se $(v_i, v_j) \in E$, então v_i é adjacente a v_j e v_j é adjacente a v_i.

Portanto, o tráfego de redes é representado pelo grafo $G = (V, E)$, onde o conjunto de vértices V é constituído por todos os endereços IP do tráfego de rede, e cada aresta do conjunto de arestas E representa a comunicação entre dois endereços IP (vértices) do conjunto V, justificando assim a utilização de grafos para a modelagem inicial dos dados.

Cada vértice $v_i \in V$ é associado a um único IP do tráfego de rede, sendo que o número inteiro que representa o IP em questão, obtido do cabeçalho do datagrama IP, é utilizado como identificador para o vértice. Uma aresta $e = (v_i, v_j)$ é inserida no conjunto de arestas E, no momento em que ocorre a primeira transferência de dados (envio de um pacote) entre os IPs representados pelos vértices v_i e v_j. Quaisquer transferências subsequentes referem-se à mesma aresta.

Dadas as definições citadas anteriormente, os seguintes termos serão utilizados com significados equivalentes, no decorrer desta dissertação:

- vértice - IP - endereço IP - número inteiro representando o IP - *host* - dispositivo conectado à rede;
- aresta - transferência de dados entre dois IPs - comunicação entre dois IPs -

[1] Parte-se do princípio que um *host* ou dispositivo conectado à rede é representado por um endereço IP, a não ser para casos explicitamente especificados.

envio de pacote de um IP para outro.

No processo de geração do grafo G, inicialmente os conjuntos V (vértices) e E (arestas) são vazios ($V = \emptyset$ e $E = \emptyset$). À medida que cada pacote é lido do tráfego de rede e utilizado para alimentar o processo de geração do grafo, três casos podem ocorrer. Para apresentar os casos, seja v_i um dos IPs do pacote, v_j o outro IP e $e = (v_i, v_j)$ a comunicação entre estes IPs (representada pelos dados contidos no pacote).

Caso 1 Os vértices v_i e v_j não existem no conjunto V do grafo $G = (V, E)$. Ocorre na leitura do primeiro pacote entre dois IPs, onde ambos ainda não apareceram nos dados coletados. São adicionados ao conjunto V os vértices v_i e v_j [$V \leftarrow V \cup v_i$ e $V \leftarrow V \cup v_j$] e é adicionada ao conjunto E a aresta $e = (v_i, v_j)$ [$E \leftarrow E \cup e$ ou $E \leftarrow E \cup (v_i, v_j)$].

Caso 2 O vértice v_i não existe no conjunto V do grafo $G = (V, E)$. Ocorre na leitura do primeiro pacote entre dois IPs, onde um dos IPs ainda não apareceu nos dados coletados. É adicionado ao conjunto V o vértice v_i ($V \leftarrow V \cup v_i$) e é adicionada ao conjunto E a aresta $e = (v_i, v_j)$ [$E \leftarrow E \cup e$ ou $E \leftarrow E \cup (v_i, v_j)$].

Caso 3 Os vértices v_i e v_j existem no conjunto V, mas a aresta $e = (v_i, v_j)$ não existe no conjunto E do grafo $G = (V, E)$. Ocorre na leitura do primeiro pacote entre dois IPs, onde ambos já apareceram nos dados coletados, mas ainda não houve comunicação entre eles. É adicionada ao conjunto E a aresta $e = (v_i, v_j)$ [$E \leftarrow E \cup e$ ou $E \leftarrow E \cup (v_i, v_j)$]).

A geração do grafo G consiste na criação de uma lista de adjacência (Gibbons, 1985), onde cada vértice tem uma lista associada de vértices adjacentes. A Figura 5.2 apresenta o exemplo de um grafo G, gerado para um tráfego de rede hipotético, e sua respectiva lista de adjacência.

Para cada vértice contido nas listas de adjacência dos vértices A, B, C, D e E, pode-se observar a ocorrência do símbolo $\uparrow$. Este é utilizado para denotar que o elemento representante do vértice é um apontador. Por exemplo, a lista de adjacência do vértice A contém um apontador para o vértice B e outro para o vértice C, significando que o IP A se comunicou como o IP B e com o IP C. Isto evita que identificadores de IPs sejam replicados, reduzindo assim o espaço ocupado no armazenamento do grafo.

O grande problema de uma representação baseada em grafos é a determinação de um método para a localização de vértices. A metodologia empregada para solucionar este problema é apresentada na próxima seção (5.2.2).

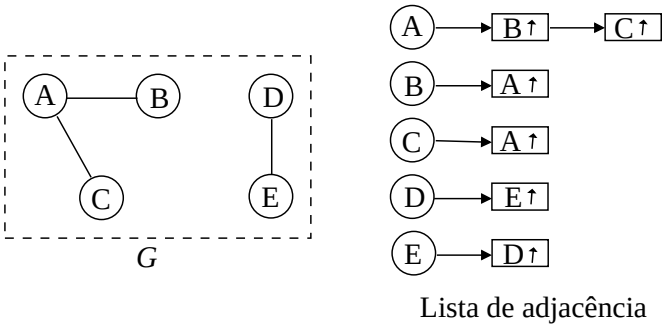

FIGURA 5.2 – Grafo G representando o tráfego de rede e sua listas de adjacências.

5.2.2 Solução para o Problema de Localização de Vértices

A maioria das implementações de grafos necessitam de um método sistemático para localização de vértices. Foram vistos três casos na seção 5.2.1, que envolvem a identificação de vértices, de forma que ações são tomadas de acordo com a existência ou não de um determinado vértice no conjunto V.

Normalmente, métodos de identificação (localização) de vértices em um grafo, como *depth-first search* (Gibbons, 1985), têm complexidade linear - $O(max(n, |E|))$ - onde n é o número de vértices e $|E|$ é o número de arestas. Neste caso a complexidade está associada ao tempo de execução do método ou ao número de comparações realizadas para encontrar o vértice.

Para minimizar o número de comparações feitas na busca de um determinado vértice, foi analisado um outro método baseado na utilização de uma árvore binária balanceada (Gonnet e Baeza-Yates, 1991). Neste novo método, os vértices do grafo G correspondem aos nós da árvore. A localização (ou identificação) de um nó em uma árvore binária balanceada tem complexidade logarítmica - $O(\log_2 n)$ - onde n é o número de nós (equivalente ao número de vértices no grafo G). Portanto, esta nova abordagem melhora o desempenho na localização em pelo menos uma ordem de magnitude.

A Figura 5.3 apresenta a árvore binária balanceada, já associada às listas de adjacências, para o grafo G da Figura 5.2.

São utilizados os números inteiros representando os endereços IP, como chaves (ou identificadores) para os nós da árvore. Para o exemplo da Figura 5.3 tem-se $A < B < C < D < E$. À medida que vértices são inseridos no conjunto de vértices V do grafo $G = (V, E)$, os apontadores para os nós da árvore são atualizados, mantendo-a balanceada.

Depois de especificada a representação básica para o tráfego de rede, onde números IP são inseridos em uma árvore binária balanceada, e a comunicação entre cada par de IPs

constitui um elemento da lista de adjacências de cada nó da árvore, faz-se necessário apresentar a visão geral do modelo criado para a reconstrução de sessões TCP/IP, como descrita na próxima sessão (5.2.3).

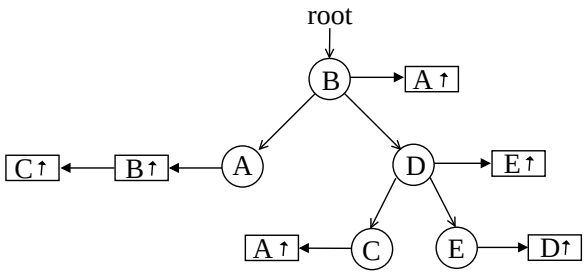

FIGURA 5.3 – Árvore binária balanceada associada a lista de adjacência.

5.2.3 Visão Geral do Modelo

Atualmente, a grande maioria dos fabricantes de sistemas de detecção de intrusão de rede (SDIRs) diz que uma das características fundamentais de seus produtos é a "remontagem".

Apesar deste termo ser amplamente utilizado, sua real finalidade não é bem especificada. Ranum (2001) enumera algumas características, relacionadas ao termo "remontagem", que um SDIR deve ter:

Desfragmentação (*defragmenting*) processo de combinar datagramas IP fragmentados em um único pacote, de modo que este possa ser checado em busca de ataques.

Reordenação (*reordering*) processo de reordenar datagramas IP fragmentados ou mensagens TCP, de forma que estejam na sequência correta.

Remontagem do fluxo de dados (*stream reassembly*) processo de combinar mensagens TCP, de modo que estas representem uma cadeia de dados ordenados da forma como o *host* de destino está interpretando.

Rastreio do estado das sessões (*state tracking*) processo de rastrear os números de sequência TCP e o estado das sessões, de forma que o SDIR possa entender quais dados o *host* destino trata como válidos.

Para metodologia de desenvolvimento do sistema de reconstrução de sessões TCP/IP, algumas destas características foram adaptadas para melhor se adequar à modelagem de suporte do sistema, e são descritas como se segue:

- Desfragmentação constitui o processo de combinar datagramas IP fragmentados em um único pacote, de modo que este possa ser inserido, de forma correta, em sua sessão correspondente;

- Reordenação constitui o processo de reordenar apenas datagramas IP fragmentados, de forma que estejam na sequência correta;
- Remontagem do fluxo de dados não é tratada nesta modelagem, pois esta baseia-se apenas no cabeçalho dos pacotes TCP/IP;
- Rastreio do estado das sessões constitui o processo de rastrear e inferir sobre o estado das sessões TCP/IP, diferenciadas por protocolo da camada de transporte (TCP, UDP ou ICMP).

O modelo criado para representar os dados, fluxo de execução e as funções do sistema de reconstrução de sessões *RECON*, foi subdividido em módulos, como mostra a Figura (5.4).

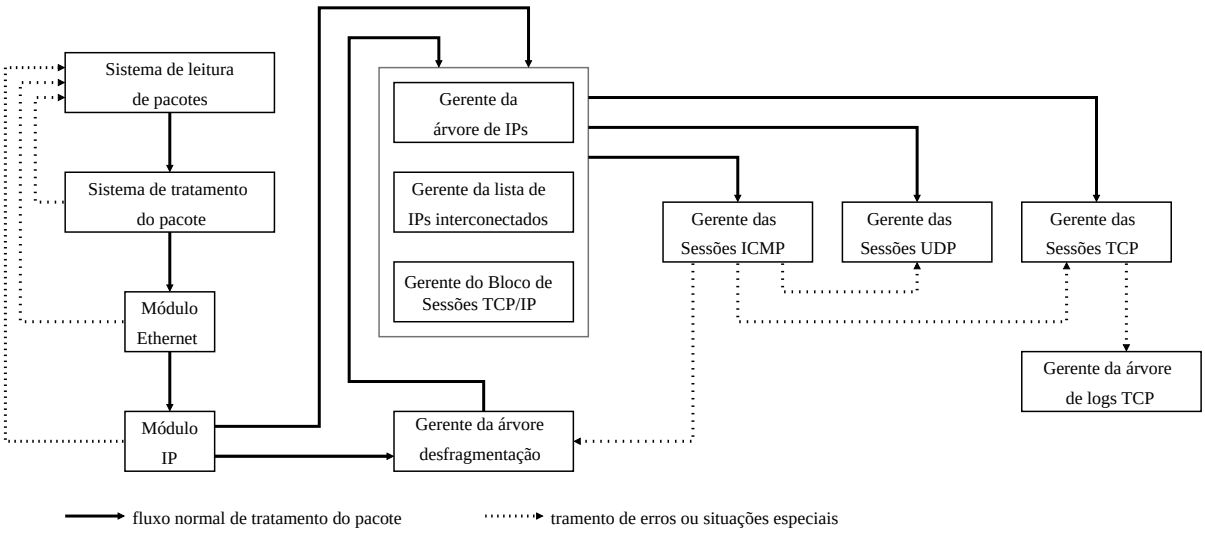

FIGURA 5.4 – Visão geral do modelo para a reconstrução de sessões.

O primeiro módulo a ser descrito é o sistema de leitura de pacotes. Este utiliza a biblioteca de captura de pacotes para obter de um arquivo, contendo o tráfego de rede a ser analisado, cada pacote e o envia para o sistema de tratamento do pacote. Este, por sua vez, é responsável por verificar se o *frame*, contido no pacote, é do tipo *Ethernet*. Caso positivo, envia o *frame* para o módulo *Ethernet*. Caso contrário, devolve o controle para o sistema de leitura de pacotes.

O módulo *Ethernet* checa se o cabeçalho *Ethernet* está completo e, caso positivo, verifica se os dados contidos no *frame* constituem um datagrama IP. Caso contrário, devolve o controle para o sistema de leitura de pacotes. Se o *frame* contiver um datagrama IP, este é enviado para o módulo IP.

O módulo IP verifica se o cabeçalho do datagrama está completo e, caso contrário, devolve

o controle para o sistema de leitura de pacotes. Se o datagrama está correto, então existem duas possibilidades: ele é um datagrama normal ou é um fragmento.

Para o datagrama normal, verifica-se qual protocolo da camada de transporte[2] está sendo usado (TCP ou UDP ou ICMP). Então, de acordo com o protocolo, é feita a verificação do cabeçalho da camada de transporte em questão. Caso este esteja completo, o datagrama IP é enviado para os módulos gerente da árvore de IPs, gerente da lista de IPs interconectados e gerente do bloco de sessões TCP/IP.

Se o datagrama é um fragmento, este é enviado para o gerente da árvore de desfragmentação. Este módulo é responsável por receber cada fragmento IP e realizar as seguintes operações: verifica se existe um nó da árvore de desfragmentação associado ao fragmento; cria o nó se este não existir, ou atualiza as informações associadas a este nó, que dizem se o fragmento ainda está sendo remontado, se o tempo de remontagem expirou, ou se o datagrama já foi completamente desfragmentado. Caso este tenha sido desfragmentado, é tratado da mesma forma que um datagrama normal, como descrito anteriormente.

O gerente da árvore de IPs e o gerente da lista de IPs interconectados são responsáveis por gerar e atualizar o grafo, que representa o tráfego de rede, como descrito nas seções 5.2.1 e 5.2.2. O gerente da árvore de IPs obtém os IPs de origem e destino do datagrama e os insere na árvore de IPs, caso estes não façam parte dessa. E o gerente da lista de IPs interconectados cria a relação entre o par de IPs do datagrama, através da atualização das listas de adjacência de cada um dos IPs em questão. Já o gerente do bloco de sessões TCP/IP cria um bloco de acesso às sessões associadas a esse par de IPs (caso este ainda não tenha sido criado).

Depois de checar o cabeçalho da camada de transporte e atualizar a árvore de IPs, as listas de adjacências e o bloco de sessões TCP/IP, é extraída do datagrama IP a mensagem, correspondente ao protocolo da camada de transporte, sendo utilizada pelo pacote. Esta é, então, enviada ao gerente das sessões ICMP ou ao gerente das Sessões UDP ou gerente das Sessões TCP, de acordo com o protocolo de transporte.

O gerente das sessões ICMP divide-se em duas partes: o gerente das sessões ICMP de informação e o gerente dos pacotes ICMP de erro. Caso a mensagem ICMP seja de informação, verifica-se se o pacote faz parte de uma sessão previamente criada, de acordo com o tipo da mensagem ICMP e com seu campo "*identifier*". Caso positivo, o pacote é adicionado à sua respectiva sessão. Caso contrário, uma nova sessão é criada e adicionada a lista de sessões ICMP de informação, e o pacote é associado a esta. Já para a mensagem

[2]Para facilitar o entendimento, assume-se que o ICMP é um protocolo da camada de transporte.

ICMP de erro, o pacote que a contém é adicionado a lista de pacotes ICMP de erro, e é associado ao pacote gerador do erro, caso seja possível.

Na Figura (5.4), as setas tracejadas, saindo do gerente das Sessões ICMP para o gerente das Sessões UDP e para o gerente das Sessões TCP, indicam a associação de pacotes ICMP de erro com pacotes que fazem parte das sessões UDP ou TCP, e que geraram cada pacote de erro. E a seta para o gerente da árvore de desfragmentação indica a ocorrência do pacote ICMP de erro *time exceeded in fragmentation reassembly* - tipo 11, código 1 - cuja função é relatar que o tempo estipulado pelo *host* recebendo os fragmentos expirou e o pacote não foi remontado. Quando da ocorrência deste tipo de pacote de erro, o gerente das sessões ICMP interage com o gerente da árvore de desfragmentação, para localizar o pacote sendo remontado. Caso este seja localizado, é associado ao pacote um atributo dizendo que o tempo de remontagem expirou.

O gerente das Sessões UDP verifica se o pacote faz parte de uma sessão previamente criada, de acordo com as portas de origem e destino, contidas na mensagem UDP. Caso positivo, o pacote é adicionado à sua respectiva sessão. Caso contrário, uma nova sessão é criada e adicionada a lista de sessões UDP, e o pacote é associado a esta.

O gerente das Sessões TCP verifica se o pacote faz parte de uma sessão previamente criada. Esta verificação ocorre através da checagem das portas de origem e destino contidas na mensagem TCP e do estado atual da sessão. Mesmo que o pacote contenha portas de origem e destino que casem com uma sessão TCP já criada, o estado desta é verificado, pois esta sessão já pode ter sido terminada. Se este pacote pertencer a uma sessão TCP, cujo estado permita sua associação, este é adicionado à respectiva sessão e o estado desta é atualizado. Caso não pertença a uma sessão previamente criada, uma nova sessão TCP é adicionada a lista de sessões TCP, o pacote é associado a esta nova sessão e seu estado é atualizado.

Existem situações em que pacotes TCP podem caracterizar algum comportamento estranho ou não esperado, ou mesmo não estar associados a alguma sessão. Para tratar estas situações foi criado o gerente da árvore de *logs* das sessões TCP, que por sua vez, é controlado pelo gerente das Sessões TCP. Algumas destas situações são: pacotes TCP não esperados durante o processo de início de uma conexão TCP (*three-way handshake*), ou pacotes com uma combinação de *flags* não esperada, muitas vezes utilizados por sistemas de varredura de redes (*scanners*). Quando o gerente das sessões TCP identifica um pacote deste gênero, ele aciona o gerente da árvore de *logs* TCP. Este então atualiza a sua árvore de *logs*, adicionando o pacote TCP ao nó, identificado pelos IPs de origem e destino obtidos do pacote. Caso este nó não exista, ele é criado e o pacote é então adicionado.

O resultado da execução de todo o processo descrito anteriormente, para a modelagem das sessões TCP/IP, é apresentado na Figura (5.5).

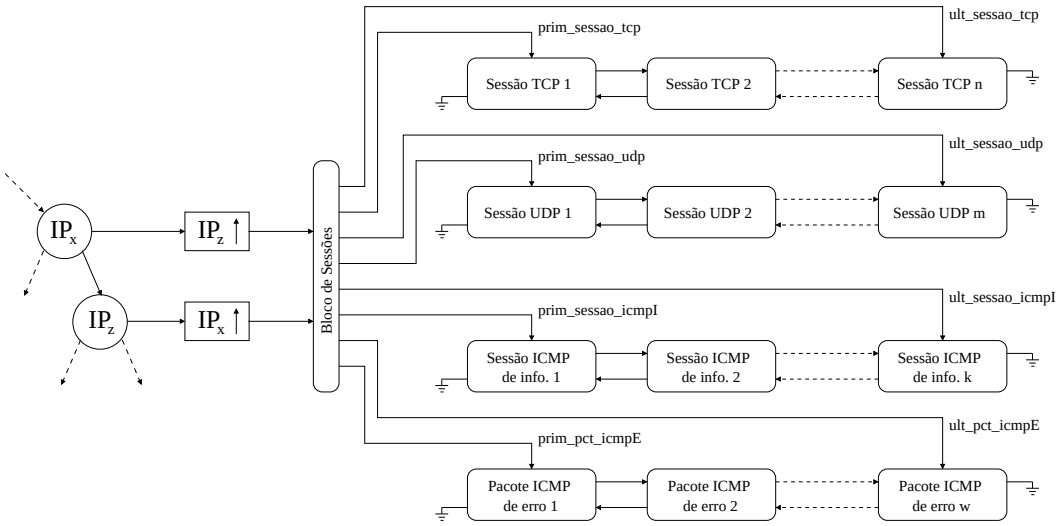

FIGURA 5.5 – Sessões TCP/IP associadas a cada par de IPs interconectados.

Depois de processado todo o arquivo de dados capturados, tem-se uma árvore binária balanceada com todos os IPs do tráfego de rede, onde para cada par de IPs interconectados, seus respectivos nós na árvore terão em suas listas de adjacências um elemento que referencia o outro IP em questão. Estes elementos, pertencentes às listas de adjacências dos dois IPs, referenciam um bloco de sessões TCP/IP comum. Este bloco mantém referências para a primeira e para a última sessão TCP, UDP e ICMP de informação e para o primeiro e último pacote ICMP de erro, de modo que estas sessões e pacotes são elementos de listas duplamente encadeadas.

Justifica-se a utilização de listas duplamente encadeadas na representação das sessões e pacotes, por facilitar a localização de elementos que a compõem, pois neste tipo de estrutura é permitido caminhar tanto para frente quanto para trás, de acordo com a necessidade.

A próxima seção (5.3) apresenta, de forma detalhada, a implementação do *RECON* - Sistema de Reconstrução de Sessões TCP/IP, baseado na modelagem aqui descrita.

5.3 A IMPLEMENTAÇÃO DO SISTEMA DE RECONSTRUÇÃO DE SESSÕES

Nesta seção são descritos os recursos utilizados no desenvolvimento do *RECON* - Sistema de Reconstrução de Sessões TCP/IP, é apresentada a visão geral do sistema e são descritos, de forma detalhada, os módulos funcionais que o compõem.

5.3.1 Recursos Utilizados no Desenvolvimento do Sistema

O desenvolvimento das rotinas do *RECON* - Sistema de Reconstrução de sessões TCP/IP - foi realizado em linguagem C ANSI (Kernighan e Ritchie, 1990) e estas foram compiladas com o compilador GNU C (*gcc*)[3]. O sistema foi compilado e executado nos sistemas operacionais **Linux**, **FreeBSD** e **OpenBSD**, para a plataforma **UNIX**.

O equipamento utilizado para a implantação do *RECON* trata-se de um PC, com processador Pentium III de 550MHz, 256Mb de memória RAM, aproximadamente 40Gb de capacidade em disco rígido, e interface de rede *Ethernet* 100/10 Mbits/s. O sistema FreeBSD foi utilizado neste equipamento, onde algumas medidas de segurança foram tomadas com o intuito de minimizar a possibilidade de ataques. O *kernel* do sistema operacional foi recompilado, de modo que funcionalidades desnecessárias foram removidas. Serviços potencialmente inseguros ou desnecessários também foram excluídos. Além disso, um *Firewall* (filtro de pacotes) foi instalado neste equipamento, permitindo apenas o acesso explicitamente autorizado.

Este equipamento trabalha como estação de análise de dados e recebe, periodicamente, arquivos contendo o tráfego de rede, coletados pelo componente sensor do sistema de detecção de intrusões SHADOW. Estes arquivos são armazenados em disco e são posteriormente utilizados como entrada de dados para o *RECON*.

5.3.2 Visão Geral do Sistema

O *RECON* foi subdividido em módulos funcionais, que executam uma série de procedimentos, cujo objetivo é reconstruir as sessões TCP/IP contidas em um tráfego de rede. A Figura (5.6) apresenta a visão geral da implementação do sistema, e uma breve descrição dos procedimentos executados por cada um de seus módulos é apresentada posteriormente.

No momento em que o *RECON* é executado, os seguintes procedimentos são realizados:

a) São inicializados os contadores e as variáveis globais do sistema. Estes contadores são utilizados posteriormente, na apresentação de estatísticas relacionadas ao tráfego de rede analisado;

b) Obtém-se o arquivo de configuração do RECON, que é então passado para o *parser*. Este tem como função extrair valores do arquivo e armazená-los em variáveis que serão utilizadas no decorrer da execução do sistema.

[3]Página Web em `http://www.gnu.org/software/gcc/gcc.html`.

c) O sistema permite a utilização de vários argumentos. Dentre eles, um argumento indispensável é o arquivo contendo o tráfego de rede a ser analisado. Nesta etapa, o sistema verifica a consistência destes argumentos, além de verificar algumas informações obtidas do arquivo de configuração.

d) Nesta etapa, o arquivo contendo o tráfego de rede, armazenado em formato *tcpdump*, é passado para rotina da biblioteca *libpcap*, que é responsável pela extração dos pacotes. Esta rotina, que constitui o leitor do tráfego de rede, executa um laço (*loop*), passando para o processador de pacote cada pacote do arquivo de dados. Os outros processadores são acionados de acordo com o tratamento e características de cada pacote lido. Este conjunto de rotinas constitui o núcleo do sistema de reconstrução de sessões TCP/IP, e é a implementação da modelagem apresentada na seção (5.2).

e) E finalmente, o gerador de resultados, de acordo com os argumentos passados na execução do *RECON* e com os parâmetros obtidos do arquivo de configuração, lê e formata as informações geradas e estruturadas na etapa anterior, e as apresenta para o usuário.

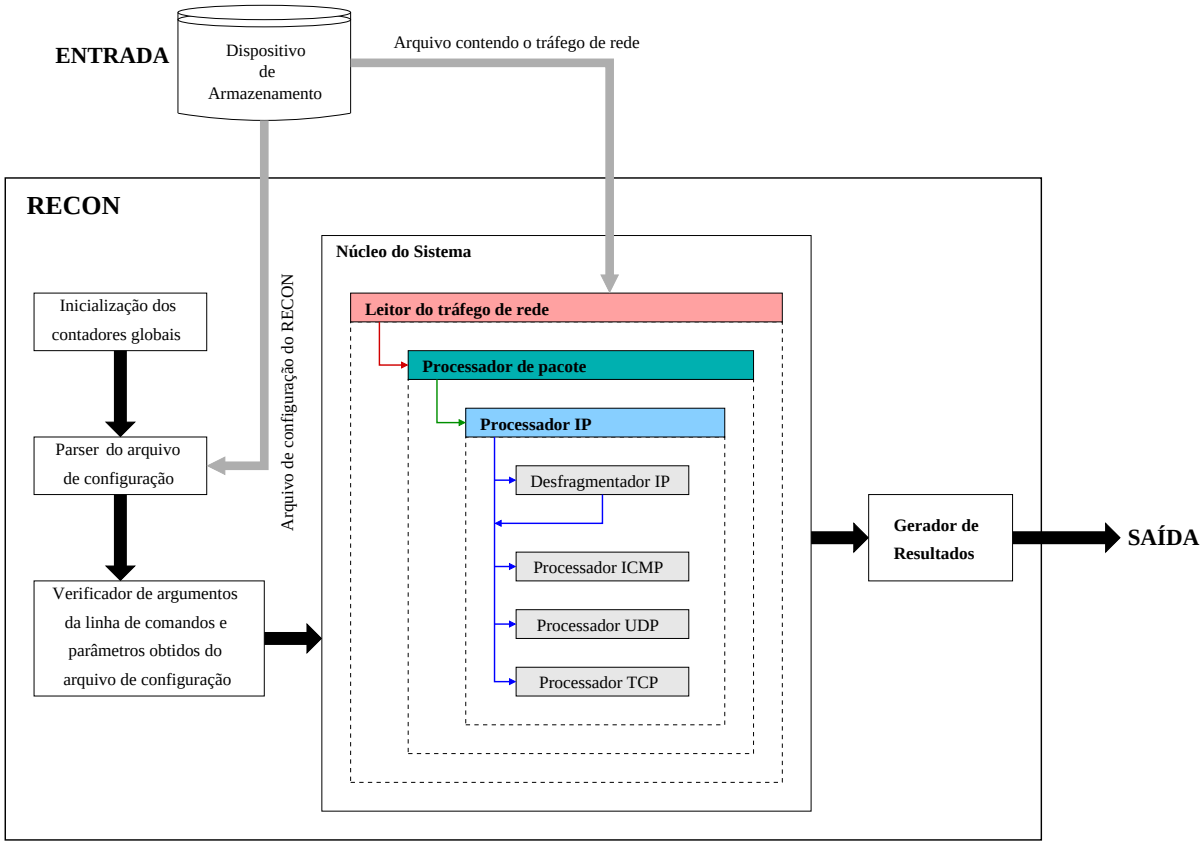

FIGURA 5.6 – Visão geral da implementação do *RECON*.

Pode-se observar na Figura (5.6) que os módulos responsáveis pela reconstrução das sessões TCP/IP estão contidos no sistema *RECON*. A seção 7.1 propõe uma modificação, onde esses módulos passam a constituir uma biblioteca independente. Esta nova biblioteca poderá ser, então, utilizada por aplicações envolvendo detecção de intrusões, ou mesmo aplicações de monitoramento de rede.

As próximas seções apresentam, de forma mais aprofundada, a implementação de cada um dos módulos que compõem o *RECON*.

5.3.3 A Inicialização dos Contadores Globais

Este módulo é responsável por inicializar as variáveis globais, que têm como finalidade armazenar a contagem e a ocorrência de certas características dos pacotes que compõem o tráfego de rede analisado. Todos os contadores globais, definidos no sistema *RECON*, bem como suas descrições, são apresentados na Tabela (B.1).

Alguns dos principais contadores são: total de pacotes lidos; endereços IP distintos; conexões distintas entre pares de endereços IP; datagramas IP com o conteúdo truncado; datagramas IP cujos cabeçalhos contém opções; datagramas IP fragmentados; Mensagens ICMP desconhecidas ou reservadas; Mensagens ICMP do tipo *fragment reassembly time exceeded*; Mensagens ICMP, UDP e TCP inseridas nas estruturas de reconstrução de sessões; Mensagens ICMP de informação, do tipo *request* e do tipo *reply*, inseridas nas estruturas de reconstrução de sessões; Mensagens ICMP de erro inseridas nas estruturas de reconstrução de sessões; Mensagens TCP com o cabeçalho corrompido; Mensagens TCP cujos cabeçalhos contém opções, e Mensagens TCP inseridas na árvore de *logs* TCP.

Além dos contadores, este módulo inicializa também variáveis, que são apontadores (ou ponteiros) para as árvores binárias balanceadas de IPs, de desfragmentação e de *logs* TCP. Estas variáveis apontam, inicialmente, para **NULL** e são atualizadas posteriormente pelos módulos processadores e desfragmentador.

5.3.4 O *Parser* do Arquivo de Configuração

O módulo responsável por realizar o *parsing* do arquivo de configuração do *RECON* é implementado pela função ***parse_configfile***, e tem o seguinte formato:

int ***parse_configfile****(char *file_name)*

file_name é um apontador para um *buffer*, que contém o caminho e o nome do arquivo de configuração. Este arquivo pode ser especificado através de um argumento do executável do sistema *RECON*. Caso não seja especificado, o sistema adota o arquivo

"/etc/local/recon.conf" como padrão.

Inicialmente, a rotina ***parse_configfile*** checa se o arquivo de configuração existe, e caso não exista, termina a execução do sistema, retornando o respectivo erro. Caso exista, este arquivo é aberto no modo leitura e o processo de *parsing* é iniciado.

A seção B.2 apresenta um exemplo deste arquivo, contendo uma série de parâmetros de configuração. A grande maioria dos parâmetros apresentados são utilizados na seleção, formatação e apresentação dos resultados gerados pelo sistema, e serão discutidos posteriormente.

Alguns destes parâmetros são apresentados e descritos como se segue:

- DATA_DIR: especifica o diretório, onde os arquivos contendo o tráfego de rede estão armazenados;
- TEMP_DIR: especifica o diretório a ser utilizado pelo *RECON* como área de armazenamento temporário;
- OUTPUT_DIR: especifica o diretório onde os resultados gerados serão armazenados (depende de um argumento na linha de comandos);
- NETWORK: especifica o endereço de rede, correspondente à rede monitorada;
- NETMASK: especifica a máscara de rede, correspondente à rede monitorada.

Depois de abrir o arquivo, o *parser* realiza um conjunto de procedimentos. Inicia-se um laço (*loop*), que lê cada linha do arquivo de configuração. Um número fixo de bytes (atualmente 1024 bytes) da linha é copiado para um *buffer*. Portanto, os bytes de uma linha que exceda este valor são descartados. Se a linha for um comentário (iniciada pelo caracter '#'), esta é descartada. Caso contrário, espaços e comentários que não iniciam a linha são removidos. Então, o *parser* busca pelo separador '='. Este é utilizado para separar cada parâmetro, no arquivo de configuração, de seu respectivo valor. Uma vez estabelecida a separação, o parâmetro é copiado para um *buffer* e o valor para outro, sendo que estes *buffers* também têm tamanho limitado. O *parser* verifica, então, se o parâmetro no *buffer* confere com algum na lista de parâmetros conhecidos. Caso positivo, o *buffer* contendo o valor é checado. Cada parâmetro conhecido pelo *parser* tem um valor máximo permitido. Se o valor obtido do arquivo exceder o valor permitido, um erro é retornado. Caso o valor esteja dentro dos limites pré-estabelecidos, este é armazenado em uma variável do sistema, para posterior utilização.

5.3.5 O Verificador de Argumentos e Parâmetros Obtidos do Arquivo de Configuração

Este módulo, responsável por verificar os argumentos da linha de comandos do *RECON* e o valor dos parâmetros obtidos do arquivo de configuração, é implementado pela função ***check_config_and_arg_values***, e tem o seguinte formato:

int ***check_config_and_arg_values****(void)*

O retorno da função igual a 0 (zero) indica que algum erro ocorreu. Então, todos os erros identificados são mostrados e a execução do *RECON* é terminada.

A verificação inicia-se com a checagem do conteúdo das variáveis atualizadas pelo módulo de *parsing* (seção 5.3.4). Apesar de checar o tamanho e os valores a serem armazenados nas variáveis, o *parser* não trata de sua consistência.

Alguns conteúdos de variáveis, checados para o arquivo de configuração, são:

- existência do diretório, onde estão os arquivos contendo o tráfego de rede;
- existência do diretório, onde estão armazenados os resultados;
- existência do diretório temporário;
- formato dos endereços de rede e máscara de rede.

Além disso, esta rotina verifica os argumentos passados na linha de comandos, quando da execução do *RECON*. Existem argumentos que são exclusivos, ou seja, não devem ocorrer simultaneamente. Outros dependem de valores e outros são obrigatórios.

A execução do *RECON* sem argumentos, gera uma linha, como a apresentada na Figura (5.7)[4] abaixo, que apresenta todos os argumentos válidos. Aqueles delimitados pelos caracteres '<' e '>' são obrigatórios, os delimitados por '[' e ']' são opcionais, e argumentos divididos por '|' são exclusivos.

```
Usage: recon [-AaFfhIiLlOSTtUuv] [-c config_file] [-C pkts]
             <-r dump_file | -d yyyymmddhh:yyyymmddhh> [ -R filter_file | expression ]
```

FIGURA 5.7 – Linha gerada na execução do *RECON* sem argumentos.

As funções de alguns dos argumentos da linha de comandos são: mostrar todas as sessões reconstruídas; mostrar as sessões TCP ou UDP ou ICMP; mostrar pacotes ICMP de erro;

[4] A linha foi quebrada para facilitar a legibilidade.

apresentar o logs TCP; ler '**n**' pacotes do arquivo contendo o tráfego de rede; filtrar o arquivo contendo o tráfego de rede, através de uma expressão ou arquivo de filtragem; apresentar a tabela de desfragmentação; ativar o modo *verbose*, dentre outras.

Uma listagem descritiva dos argumentos, como ilustrada na seção B.3, é apresentada quando o *RECON* é executado com o argumento '**-h**'.

5.3.6 O Leitor do Tráfego de Rede

O leitor do tráfego de rede utiliza algumas rotinas implementadas pela biblioteca *libpcap* (seção 3.2), que permitem ler de um arquivo de dados previamente armazenado, as informações de cada pacote capturado. Além disso, permitem filtrar o arquivo, de modo que apenas o tráfego desejado seja lido. As descrições das funções utilizadas foram obtidas das páginas de manual (*man pages*) da biblioteca e de Carstens (2002).

- *pcap_t* ∗***pcap_open_offline***(*char* ∗*fname, char* ∗*ebuf*)
  Esta função é utilizada para abrir um arquivo previamente armazenado, que contenha o tráfego de rede capturado, no modo de leitura. *fname* especifica o nome do arquivo a ser aberto. Caso o nome "-" seja utilizado, especifica que os dados serão obtidos da entrada padrão (*stdin*). *ebuf* é usado para retornar um texto de erro, caso a função falhe e retorne **NULL**.

- *int* ***pcap_compile***(*pcap_t* ∗*p, struct bpf_program* ∗*fp, char* ∗*str, int optimize, bpf_u_int32 netmask*)
  Esta função é utilizada para compilar a *string str* em um código de filtragem, que pode ser entendido pela biblioteca. *p* é o ponteiro para o descritor da sessão de leitura dos dados. *program* é o ponteiro para uma estrutura do tipo *bpf_program*, que é preenchida pela própria função com a versão compilada do filtro. *optimize* informa para a função se a otimização do código de filtragem deve ser realizada. *netmask* especifica a máscara da rede local.

- *int* ***pcap_setfilter***(*pcap_t* ∗*p, struct bpf_program* ∗*fp*)
  Esta função é utilizada para aplicar um código de filtragem a uma sessão de leitura de dados. *p* é o ponteiro para o descritor da sessão de leitura dos dados. *fp* é o ponteiro para a estrutura do tipo *bpf_program*, que contém a versão compilada do código de filtragem, e é resultado da chamada para a função ***pcap_compile***.

- *int* ***pcap_loop***(*pcap_t* ∗*p, int cnt, pcap_handler callback, u_char* ∗*user*)
  Esta função é utilizada para, efetivamente, ler os pacotes capturados. *p* é o ponteiro para o descritor da sessão de leitura dos dados. *cnt* especifica o

número de pacotes a serem lidos. Caso *cnt* seja negativo, todos os pacotes são lidos, exceto se um erro ocorrer. *callback* especifica a função de retorno, que será responsável pelo tratamento de cada pacote lido. *user* é utilizada para passar argumentos adicionais para a função de retorno *callback*.

- *void* ***pcap_close****(pcap_t *p)*
  Esta função é responsável por fechar os arquivos associados ao descritor da sessão de leitura dos dados *p* e liberar os recursos alocados.

As estruturas *pcap_t*, *pcap_handler* e *bpf_program* são internas à biblioteca *libpcap* e não serão discutidas neste trabalho.

O valor 0 (zero) é atribuído ao argumento *netmask* da função ***pcap_compile***, pois o *RECON* utiliza, como entrada de dados, arquivos contendo o tráfego de rede previamente coletado. Este argumento deve conter o valor real da máscara da rede monitorada, apenas no momento em que os dados estão sendo capturados.

A definição da função de retorno (*callback function*), utilizada como parâmetro da rotina ***pcap_loop*** e que é responsável pelo tratamento de cada pacote, possui alguns argumentos que devem ser respeitados, pois a biblioteca *libpcap* preenche estes argumentos com informações referentes a cada pacote sendo lido. A descrição desta função, aqui intitulada "Processador de Pacote", é apresentada na próxima seção (5.3.7).

5.3.7 O Processador de Pacote

A rotina ***packet_proc*** corresponde à função de retorno, passada como argumento para a função de leitura de pacotes da biblioteca *libpcap*, descrita na seção anterior. Esta é a implementação do processador de pacote e seu formato é apresentado como se segue:

void ***packet_proc****(u_char *user, const struct pcap_pkthdr *h, const u_char *p)*

user é o ponteiro para uma *string* contendo argumentos adicionais, que podem ser passados da rotina de obtenção de pacotes para a função de retorno. Neste trabalho, este ponteiro é desconsiderado. *h* é o ponteiro para a estrutura do tipo *pcap_pkthdr*. Esta estrutura é adicionada pela biblioteca de captura de pacotes à sequência de bytes capturados para cada pacote. *p* é o ponteiro para o início da cadeia de bytes que representa o pacote.

A Figura (5.8) apresenta a estrutura *pcap_pkthdr*, que é constituída pelos seguintes campos: *timestamp*, que contém o horário de captura do pacote; *caplen*, que contém a quantidade de bytes capturados para o pacote, e *len*, que contém o tamanho efetivo do pacote.

A função ***packet_proc*** mantém dois apontadores, um para o início e outro para o final

da sequência de bytes do pacote. O primeiro, *packetp*, é igual ao ponteiro *p*, parâmetro da própria função. E o segundo, *snapend*, é igual ao ponteiro *p* somado com *caplen*.

```
struct pcap_pkthdr {
  struct timeval ts;
  bpf_u_int32 caplen;
  bpf_u_int32 len;
};
```

FIGURA 5.8 – Estrutura adicionada pela biblioteca *libpcap* a cada pacote.

Os seguintes procedimentos são, então, executados:

a) É declarado o ponteiro *ep*, como apontador para estrutura *ether_header*. Esta estrutura é definida pelo sistema operacional, e contém os campos do cabeçalho *Ethernet*;

b) São armazenados em três variáveis, *cur_pkt_tv*, *caplen* e *length*, o horário de captura, a quantidade de bytes capturados e o tamanho efetivo do pacote, obtidos da estrutura *pcap_pkthdr*;

c) A quantidade de bytes capturados para o pacote é comparada com o tamanho do cabeçalho *Ethernet*. Se a quantidade for menor que o cabeçalho, a função termina e retorna o controle para o leitor do tráfego de rede;

d) Se não for, as variáveis *caplen* e *length* são decrementadas do tamanho do cabeçalho *Ethernet*;

e) O ponteiro *ep* é inicializado, atribuindo-se *p* a *ep*, através de um *casting* de tipos. Isto permite acessar os campos do cabeçalho *Ethernet*;

f) O ponteiro *p* é incrementado do tamanho da estrutura *ether_header*, passando a apontar para o início do datagrama contido no *frame Ethernet*;

g) O cabeçalho *Ethernet* possui um campo que informa qual é o tipo de datagrama que ele está encapsulando. Este campo é checado e, se o datagrama for IP, a rotina que implementa o processador IP é executada, recebendo como parâmetros as variáveis *p*, *length* e *caplen*. Caso contrário, a função termina e retorna o controle para o leitor do tráfego de rede.

5.3.8 O Processador IP

A função ***ip_proc*** corresponde à implementação do processador IP, e seu formato é apresentado como se segue:

void ***ip_proc****(const u_char *bp, const u_int length, u_int cap_len)*

A rotina de processamento do pacote, discutida na seção anterior, preenche os parâmetros da função ***ip_proc***, de modo que *bp* aponta para o início do datagrama IP, *length* contém o tamanho efetivo do pacote e *cap_len* contém a quantidade de bytes capturados para o pacote, decrementada do tamanho do cabeçalho *Ethernet*.

São declarados quatro ponteiros - *ip*, *icmp*, *udp* e *tcp* - que apontam para as estruturas *ip*, *icmp*, *udphdr* e *tcphdr*, respectivamente. Estas estruturas são definidas pelo sistema operacional. Então, o apontador *ip* é inicializado, atribuindo-se *bp* (argumento da função ***ip_proc***) a *ip*, através de um *casting* de tipos. Isto permite acessar os campos do cabeçalho IP.

Inicia-se uma série de checagens acerca do tamanho do datagrama IP. Caso a quantidade de bytes, informada pelo parâmetro *cap_len*, seja menor que o tamanho do cabeçalho IP, a função termina e retorna o controle para o leitor do tráfego de rede.

O ponteiro *cp*, do tipo *u_char*, é declarado e inicializado, atribuindo-se *ip* a *cp*, através de um *casting* de tipos. Este novo ponteiro é incrementado do tamanho do cabeçalho IP. Para isto, é utilizado o campo *header length* da estrutura apontada por *ip*.

As variáveis *cap_len*, que é decrementada do tamanho do cabeçalho IP, e *snapend*, que aponta para o fim da sequência de bytes capturados para o pacote, serão utilizadas nas próximas verificações.

Checa-se, então, se o campo *offset*, obtido da estrutura apontada por *ip*, é igual a 0 (zero). Esta característica indica que o cabeçalho da camada de transporte (ICMP, UDP ou TCP) deve estar presente no datagrama. Caso positivo, verifica-se o campo *protocol*.

Se este apresenta um protocolo diferente do ICMP, UDP ou TCP, a função termina e retorna o controle para o leitor do tráfego de rede. Para as outras possibilidades o seguinte teste é realizado: caso '$cp + 1$' for maior que *snapend*, ou, de acordo com o campo *protocol*, *cap_len* for menor que o tamanho do cabeçalho ICMP ou UDP ou TCP, a função termina e retorna o controle para o leitor do tráfego de rede.

O próximo passo é verificar se o datagrama IP é um fragmento. Para isto, são utilizados o bit MF (*more fragments*) e o campo *offset* da estrutura apontada por *ip*. Caso o datagrama seja um fragmento, dois casos podem ocorrer: o bit MF está ativo; ou o bit MF está desativado, mas o campo *offset* é diferente de 0 (zero).

Então, é chamada a rotina implementada pelo desfragmentador IP, passando para esta

os devidos parâmetros. Os detalhes de sua implementação são apresentados na próxima seção (5.3.9). Quando a execução desta rotina termina, verifica-se se o datagrama está no estado "REMONTADO". Caso positivo, os ponteiros *cp* e *ip* são atualizados, através do nó contendo o pacote remontado na árvore de desfragmentação. À partir daí, o tratamento é semelhante para um datagrama normal ou remontado.

Obtém-se os endereços IP de origem e destino, contidos na estrutura apontada por *ip*, de modo que estes dois endereços são inseridos na árvore de IPs, caso ainda não existam. A Figura (5.9a) apresenta a estrutura de cada nó desta árvore.

ip_addr armazena o endereço IP. *f_adjl_elem* e *l_adjl_elem* são apontadores para o início e fim da lista de adjacências do nó. *adjl_elem_count* armazena o número de elementos na lista de adjacências. *left* e *right* apontam para as sub-árvores esquerda e direita. E *balance* é utilizado no balanceamento da árvore binária de IPs.

Para o nó que contém o endereço IP de origem, verifica-se se sua lista de adjacências contém o elemento correspondente ao endereço IP de destino. Caso não exista, este elemento é inserido. O mesmo é feito para o nó que contém o endereço IP de destino. A Figura (5.9b) apresenta a estrutura de cada elemento da lista de adjacências.

t_node aponta para o nó relacionado na árvore binária de IPs. *session* aponta para o bloco de sessões TCP/IP associado. E *next* aponta para o próximo elemento da lista de adjacências.

É importante ressaltar que o par <nó, elemento da lista de adjacências do nó> identifica a comunicação entre dois IPs e a estes dois está associado um bloco de sessões TCP/IP. O bloco é criado e o ponteiro *session*, dos elementos nas listas de adjacências em questão, passam a apontar para este. A Figura (5.9c) apresenta a estrutura do bloco de sessões TCP/IP.

```
struct avlnode {
  u_int32_t ip_addr;
  struct adj_list_elem *f_adjl_elem;
  struct adj_list_elem *l_adjl_elem;
  u_int32_t adjl_elem_count;
  struct avlnode *left;
  struct avlnode *right;
  char balance;
};
```

a

```
struct adj_list_elem {
  struct avlnode *t_node;
  struct tcpip_session *session;
  struct adj_list_elem *next;
};
```

b

```
struct tcpip_session {
  struct tcp_sessions *f_tcp_session;
  struct tcp_sessions *l_tcp_session;
  struct udp_sessions *f_udp_session;
  struct udp_sessions *l_udp_session;
  struct icmp_Isessions *f_icmp_Isession;
  struct icmp_Isessions *l_icmp_Isession;
  struct icmp_Epkt *f_icmp_Epkt;
  struct icmp_Epkt *l_icmp_Epkt;
  u_int8_t clear;
};
```

c

FIGURA 5.9 – Estruturas de armazenamento da árvore binária de IPs, lista de adjacências e bloco de sessões TCP/IP.

Os campos desta estrutura são apontadores para o início e fim das sessões TCP, UDP,

ICMP de informação e para o início e fim dos pacotes ICMP de erro. *clear* é utilizado apenas no momento em que esta estrutura é desalocada.

O próximo passo na execução processador IP é estipular um código para a direção do pacote. Esta informação é de suma importância, pois toda a implementação *RECON* baseia-se na utilização de árvores binárias e, apesar dos IPs estarem relacionados por listas de adjacências, não há informação sobre a direção da comunicação.

Como cada endereço IP é representado univocamente por um número inteiro de 4 bytes, a seguinte codificação é apresentada na Tabela (5.1).

TABELA 5.1: Codificação da direção do pacote

Comparação	*Código da Direção*
Endereço IP de origem < Endereço IP de destino	0
Endereço IP de origem > Endereço IP de destino	1

Depois de estipulado o código para a direção do pacote, o próximo passo é acionar o processador da mensagem na camada de transporte, de acordo com o conteúdo do campo *protocol*, obtido da estrutura apontada por *ip*. Três casos podem ocorrer, de modo que os seguintes procedimentos são realizados, de acordo com identificação do protocolo, contida neste campo:

- ICMP: inicializa-se o ponteiro *icmp*, atribuindo *cp* a *icmp*. Se o campo *type*, obtido da estrutura apontada por *icmp*, indicar que a mensagem é de informação, a rotina que implementa o Processador ICMP para sessões de informação é executada com os devidos parâmetros. O mesmo acontece caso *type* indique que a mensagem é de erro, só que, nesse caso, a rotina que implementa o Processador ICMP para pacotes de erro é executada;

- UDP: inicializa-se o ponteiro *udp*, atribuindo *cp* a *udp*. A rotina que implementa o processador UDP é executada com os devidos parâmetros;

- TCP: inicializa-se o ponteiro *tcp*, atribuindo *cp* a *tcp*. A rotina que implementa o processador TCP é executada com os devidos parâmetros.

Apesar de não ser manipulada pela função ***ip_proc***, faz-se necessário apresentar a estrutura de armazenamento dos dados obtidos do datagrama IP, pois esta será utilizada pelos processadores ICMP, UDP e TCP, descritos nas próximas seções. Esta estrutura é ilustrada na Figura (5.10).

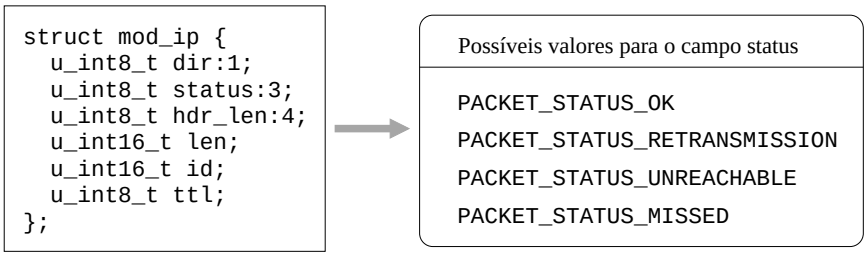

FIGURA 5.10 – Estrutura de armazenamento dos dados obtidos do datagrama IP.

Os campos *hdr_len*, *len*, *id* e *ttl* são análogos aos campos no cabeçalho IP, como visto na seção 2.3. *dir* armazena o código da direção do pacote, como descrito anteriormente. O campo *status* pode assumir os valores apresentados na Figura (5.10).

No momento em que esta estrutura é criada e os dados do datagrama são armazenados, o campo *status* recebe o valor "*PACKET_STATUS_OK*". Os demais valores serão discutidos posteriormente.

5.3.9 O Desfragmentador IP

A função ***frag_avl_insert*** corresponde à implementação do desfragmentador IP, e seu formato é apresentado como se segue:

char ***frag_avl_insert****(struct frag_node **node, struct frag_node **ret_node, struct frag_elem **ret_f_elem, struct ip *ip, u_int32_t ip_src, u_int32_t ip_dst, u_char *message, u_int caplen)*

Quando a rotina de processamento do datagrama IP, discutida na seção anterior, verifica que o datagrama é um fragmento, esta faz a chamada para a função ***frag_avl_insert***, preenchendo seus parâmetros.

node é a referência do apontador para o nó raiz da árvore de desfragmentação. *ret_node* armazena a referência do apontador do nó atual. *ret_f_elem* armazena a referência para o apontador do elemento atual, na lista encadeada de elementos de fragmentação do nó. *ip* é o apontador para a estrutura contendo o cabeçalho IP. *ip_src* e *ip_dst* são os endereços IP de origem e destino do pacote. *message* é o apontador para a sequência de bytes do pacote à partir do cabeçalho IP. E *caplen* contém o tamanho da sequência de bytes do pacote, decrementado do tamanho do cabeçalho *Ethernet* e do cabeçalho IP.

As informações do datagrama fragmentado, obtidas da estrutura apontada por *ip*, são inseridas na árvore de desfragmentação e divididas nas estruturas que a compõem. Os endereços IP de origem e destino são utilizados como identificadores do nó. Cada nó na árvore contém uma lista encadeada de elementos de fragmentação. Os campos *identification*

e *protocol* são utilizados como identificadores de cada elemento. E cada elemento da lista de desfragmentação contém uma lista encadeada de fragmentos contendo *offset* e *length* de cada fragmento.

Suponha que o IP **X** envie um pacote **ICMP** para o IP **Y**, e que este pacote tenha o campo *identification* do cabeçalho IP igual a **1000**. Suponha também que este pacote tenha sido dividido em três fragmentos. Sabe-se que os três fragmentos terão, em seus cabeçalhos IP, os campos *identification* e *protocol* semelhantes.

Então, o desfragmentador IP, ao processar estes fragmentos, realizará os seguintes procedimentos: criará um nó na árvore de desfragmentação, identificado pelo par de IPs <**X**,**Y**>; este nó terá, em sua lista de elementos de desfragmentação, um elemento identificado pelo par <**1000**,**ICMP**>, correspondente aos campos *identification* e *protocol* do cabeçalho IP; e este elemento terá uma outra lista, contendo três elementos, onde cada um destes terá o *offset* e o *length* de um fragmento.

A Figura (5.11) apresenta, de forma simplificada, o esquema de armazenamento de datagramas IP fragmentados na árvore de desfragmentação.

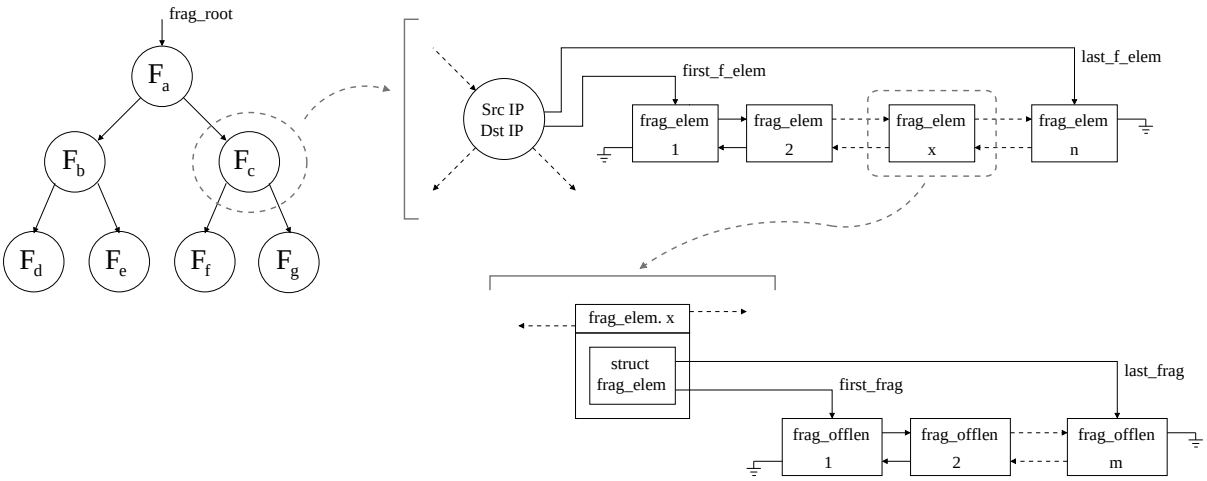

FIGURA 5.11 – Esquema de armazenamento dos fragmentos IP.

Esta abordagem permite que os dados dos fragmentos sejam armazenados, sem que informações comuns dos fragmentos de um dado pacote sejam replicadas, reduzindo assim a quantidade de recursos alocados para seu armazenamento.

A Figura (5.12) apresenta, de forma completa, as estruturas correspondentes ao nó da árvore de desfragmentação (**a**), ao elemento de desfragmentação (**b**) e ao elemento contendo o *offset* e *length* de um fragmento (**c**).

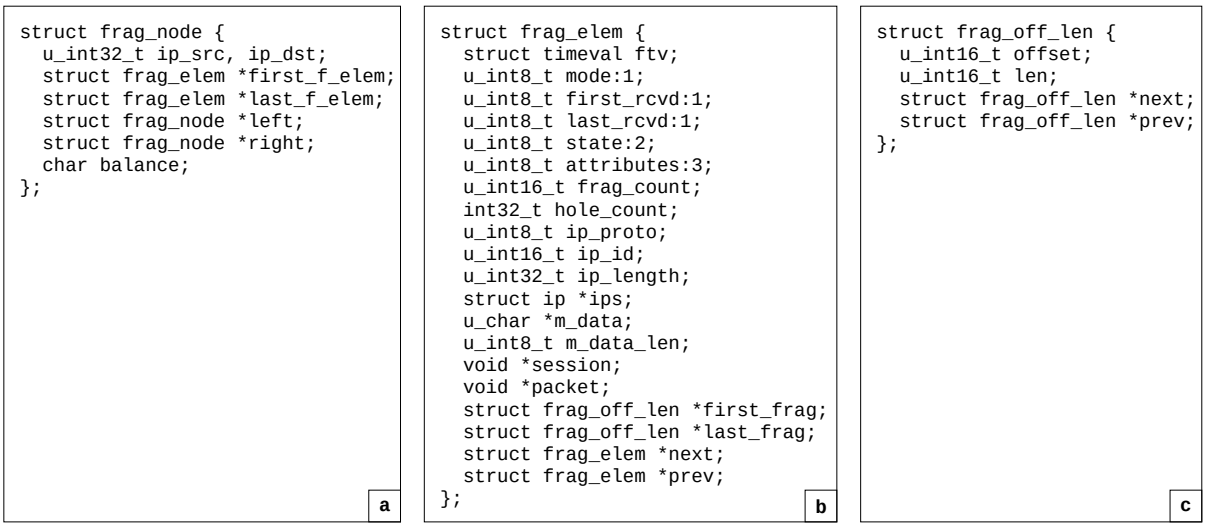

FIGURA 5.12 – Estruturas de armazenamento dos fragmentos IP.

Na estrutura *frag_node*, correspondente a cada nó da árvore de desfragmentação, os campos *ip_src* e *ip_dst* armazenam os endereços IP de origem e destino do pacote fragmentado, e são utilizados na identificação do nó. *first_f_elem* e *last_f_elem* são apontadores para o início e fim da lista encadeada de elementos de fragmentação. *left* e *right* são apontadores para as sub-árvores esquerda e direta do nó. E *balance* é utilizado no balanceamento da árvore binária de desfragmentação.

Na estrutura *frag_elem*, correspondente a cada elemento da lista de desfragmentação de um dado nó da árvore, *ftv* contém o horário de captura do primeiro fragmento. *mode* não é utilizado e está reservado para futuras extensões. *first_rcvd* e *last_rcvd* indicam se o primeiro e último fragmentos já foram recebidos. *state* contém o estado do pacote sendo desfragmentado. *attributes* contém os atributos do pacote sendo desfragmentado. *frag_count* armazena o número de fragmentos recebidos. *hole_count* conta o número de espaços entre fragmentos. *ip_proto* armazena o protocolo encapsulado no datagrama fragmentado. *ip_id* armazena o campo *identification* do cabeçalho IP. *ip_length* contém o tamanho total do datagrama IP desfragmentado. *ips* é um ponteiro para o cabeçalho IP do primeiro fragmento. *m_data* aponta para sequência de bytes capturados para o primeiro fragmento, à partir do cabeçalho IP. *m_data_len* armazena a quantidade de bytes na área apontada por *m_data*. *session* é o apontador para a sessão onde o datagrama remontado foi inserido. *packet* é o apontador para o pacote remontado, já inserido em uma sessão. *first_frag* e *last_frag* são apontadores para o início e fim da lista encadeada de *offsets* e *lengths* de cada fragmento. E *next* e *prev* são apontadores para o próximo elemento de fragmentação e para o anterior.

Na estrutura *frag_off_len*, correspondente a cada fragmento da lista de fragmentos de um

dado elemento de desfragmentação, *offset* armazena o campo *offset* do cabeçalho IP do fragmento. *len* armazena o tamanho total do fragmento IP, decrementado do tamanho do cabeçalho IP. E *prev e next* são apontadores para o próximo elemento da lista encadeada de *offsets* e *lengths* e para o anterior.

Existem três possíveis estados para um pacote fragmentado, quando seus fragmentos estão sendo inseridos na árvore de desfragmentação. Estes estados estão associados a valores que atualizam o campo *state* da estrutura correspondente ao elemento de desfragmentação do pacote, e são descritos como se segue:

- *FRAG_REASSEMBLING*: este estado é associado ao elemento de desfragmentação, quando da ocorrência do primeiro fragmento. Neste momento, o elemento de desfragmentação é criado e permanece neste estado enquanto o pacote não for completamente remontado;

- *FRAG_REASSEMBLED*: este estado é associado ao elemento de desfragmentação, quando todos os fragmentos já foram recebidos e o pacote foi remontado. Indica que mais fragmentos não podem ser associados a este elemento de desfragmentação;

- *FRAG_TIME_EXCEEDED*: este estado é associado ao elemento de desfragmentação, quando da ocorrência de um pacote ICMP do tipo *time exceeded in fragmentation reassembly*, correspondente ao elemento de desfragmentação. Diz que o tempo de remontagem do pacote expirou e indica que mais fragmentos não podem ser associados a este elemento de desfragmentação.

Para determinar se um pacote foi totalmente remontado, três campos da estrutura correspondente ao elemento de desfragmentação são utilizados: *first_rcvd*, *last_rcvd* e *hole_count*. *first_rcvd* recebe o valor "1", quando o cabeçalho IP de um fragmento contém *offset* igual a 0 e o bit MF ativado. *last_rcvd* recebe o valor "1", quando o cabeçalho IP de um fragmento contém *offset* diferente de 0 e o bit MF desativado. Quando estes dois campos têm valor "1", significa que o primeiro e o último fragmento já foram processados. Então basta determinar se os fragmentos intermediários também foram. Para isto, é utilizado o campo *hole_count*.

Quando o elemento de desfragmentação é criado, o campo *hole_count* recebe o valor "-1" e quando o primeiro fragmento, independente de seu *offset*, é inserido na lista de fragmentos, este campo recebe o valor "0". À medida que os fragmentos vão sendo inseridos na lista de fragmentos, e estes são inseridos de forma ordenada, os campos *offset* e *len* do fragmento atual são checados, de modo que o valor de *hole_count* é atualizado da seguinte forma:

- Decremento do campo *hole_count*: *offset* do fragmento atual é igual ou menor que *offset* + *len* do fragmento anterior; ou *offset* + *len* do fragmento atual é igual ou maior que *offset* do próximo fragmento;

- Incremento do campo *hole_count*: *offset* do fragmento atual é maior que *offset* + *len* do fragmento anterior; ou *offset* + *len* do fragmento atual é menor que *offset* do próximo fragmento;

Para o pacote fragmentado estar completamente remontado, os campos *first_rcvd* e *last_rcvd* devem ter o valor "1" e o campo *hole_count* deve ter o valor "0", indicando que o primeiro e o último fragmento foram processados e não há espaços entre eles. Então, o campo *state* do elemento de desfragmentação do pacote remontado recebe o valor "*FRAG_REASEMBLED*".

Quando um pacote remontado é inserido em uma sessão, seja ela ICMP, UDP, ou TCP, os campos *session* e *packet*, da estrutura correspondente ao elemento de desfragmentação do pacote, passam a apontar para a sessão onde o pacote foi inserido e para o pacote propriamente dito. Estes apontadores serão utilizados pelas funções de ajustes de sessões, implementadas no processador ICMP, a ser visto na próxima seção (5.3.10).

Existem, ainda, alguns atributos que podem ser associados ao campo *attributes* da estrutura correspondente a um elemento de desfragmentação, para um pacote sendo remontado. Estes atributos indicam a possível intenção de explorar vulnerabilidades ou fraquezas na implementação da pilha de protocolos TCP/IP, e são descritos como se segue:

- *FRAG_OVERLAP*: indica a sobreposição de fragmentos, ou seja, *offset* + *len* de um fragmento é maior que o *offset* do fragmento consecutivo. Esta técnica é muito utilizada na tentativa de iludir sistemas de detecção de intrusão (Ptacek e Newsham, 1998);

- *FRAG_TEARDROP*: indica que o tamanho do fragmento remontado excedeu o máximo permitido, que é igual 65535 bytes. Esta técnica foi muito utilizada na tentativa de indisponibilizar serviços e até mesmo *hosts* conectados à Internet (Northcutt e Novak, 2001) (Northcutt et al., 2001).

- *FRAG_MULTI*: indica que o tamanho de algum fragmento, com exceção do último, não é múltiplo de 8. Esta técnica também foi muito utilizada na tentativa de indisponibilizar serviços e até mesmo *hosts* conectados à Internet (Northcutt e Novak, 2001) (Northcutt et al., 2001).

A implementação das rotinas que compõem o desfragmentador IP foram baseadas em Clark (2002), Ziemba et al. (2002) e Miller (2002).

5.3.10 O Processador ICMP

O processador ICMP é dividido em duas partes: o processador ICMP de informação e o processador ICMP de erro. As seções 5.3.10.1 e 5.3.10.2 discutem as implementações destas partes.

5.3.10.1 Processador ICMP de Informação

A função ***icmp_Isession_proc*** corresponde à implementação do processador ICMP de informação, e seu formato é apresentado como se segue:

void ***icmp_Isession_proc****(struct icmp_Isessions **f_icmp_Isession, struct icmp_Isessions **l_icmp_Isession, struct ip *ip, struct icmp *icmp, u_int8_t icmp_categ, u_int8_t pkt_dir, u_int length)*

Quando o processador IP verifica que o pacote sendo processado contém uma mensagem ICMP de informação, este executa a chamada da função ***icmp_Isession_proc***, preenchendo devidamente seus parâmetros.

f_icmp_Isession e *l_icmp_Isession* são apontadores para o início e para o fim da lista encadeada de sessões ICMP de informação, associada aos dois IPs contidos no pacote. *ip* é o apontador para a estrutura contendo o cabeçalho IP do pacote. *icmp* é o apontador para a estrutura contendo o cabeçalho ICMP do pacote. *icmp_categ* contém a categoria da mensagem ICMP de informação, que pode ser *request* ou *reply*. *pkt_dir* contém o código da direção do pacote, como visto na seção 5.3.8. E *length* contém o tamanho da mensagem ICMP, calculada pela diferença dos campos *ip_len*, que é o tamanho total do datagrama IP, e *ip_hl*, que é o tamanho do cabeçalho IP do pacote.

As sessões ICMP de informação têm como identificador o campo *identification* do cabeçalho ICMP. Quando ocorre uma mensagem ICMP, onde seu campo *identifier* é diferente das sessões ICMP de informação associadas aos dois IPs no pacote em questão, ou a lista de sessões ICMP está vazia, uma nova sessão ICMP é criada, e todos os pacotes ICMP de informação com o campo *identification* semelhante e da mesma categoria são inseridos nesta sessão, compondo assim sua lista encadeada de pacotes.

A Figura (5.13) apresenta as estruturas correspondentes a: cada sessão ICMP de informação (**a**); cada pacote ICMP, que compõe a lista encadeada de pacotes para uma dada sessão (**b**); e os dados armazenados de cada mensagem ICMP (**c**).

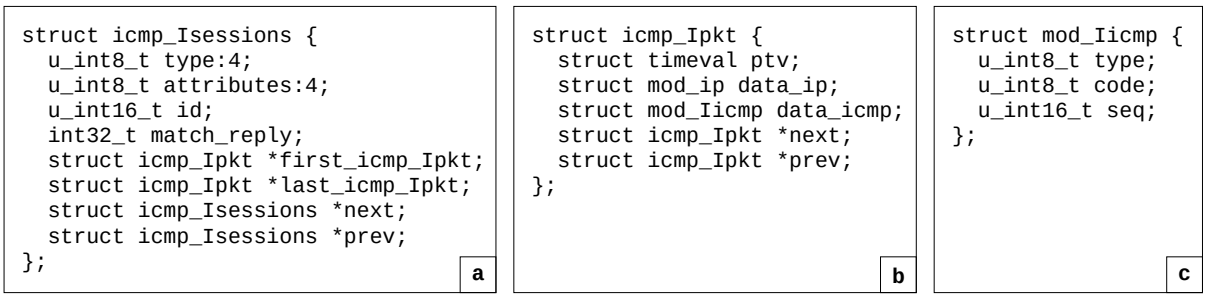

FIGURA 5.13 – Estruturas de armazenamento das sessões ICMP de informação.

Na estrutura *icmp_Isession*, correspondente a cada sessão ICMP, o campo *type* armazena o tipo da sessão, ou seja, se esta é do tipo *echo*, *timestamp*, *address mask* ou *information*. *attributes* contém os atributos associados à sessão. *id* armazena o campo *identification*, obtido do cabeçalho da mensagem ICMP, e que é utilizado como identificador da sessão. *match_reply* é o contador da diferença no número de pacotes de requisição e de resposta, inseridos na sessão. *first_icmp_Ipkt* e last_icmp_Ipkt são apontadores para o primeiro e para o último pacote da sessão. E *next* e *prev* são apontadores para a próxima sessão e para a anterior.

Na estrutura *icmp_Ipkt*, correspondente a cada pacote inserido em uma determinada sessão ICMP de informação, o campo *ptv* armazena o horário de captura do pacote. *data_ip* é uma estrutura que armazena os dados obtidos do cabeçalho IP do pacote, como visto na seção 5.3.8. *data_icmp* é uma estrutura que armazena os dados obtidos do cabeçalho ICMP. E *next* e *prev* são apontadores para o próximo pacote na sessão e para o anterior.

Na estrutura *mod_Iicmp*, correspondente aos dados da mensagem ICMP de informação armazenados para cada pacote, o campo *type* contém o tipo da mensagem ICMP e o campo *code* contém o código. O campo *seq* contém o *sequence number* associado ao pacote ICMP de informação.

Quando um pacote ICMP de informação constitui uma requisição (*request*) e este é inserido em uma sessão, o campo *match_reply* da estrutura correspondente a esta é incrementado de "1". E quando um pacote constitui uma resposta (*reply*), o campo *match_reply* é decrementado de "1".

Ao final do processamento de todos os pacotes do arquivo contendo o tráfego de rede, três situações devem ser consideradas para as sessões ICMP de informação reconstruídas, e dizem respeito ao valor armazenado pelo campo *match_reply* destas sessões. Na primeira, *match_reply* é igual a 0 (zero). Isto indica que, para todos os pacotes de requisição, houve um pacote de resposta correspondente. Na segunda, *match_reply* é maior que 0. Indica que

o número de requisições foi maior que o número de respostas. E finalmente, na terceira situação, *match_reply* é menor que 0. Indica que o número de respostas foi maior que o número de requisições.

Algumas considerações devem ser feitas acerca da última situação apresentada. Se *match_reply* for igual a "-1", uma possibilidade é que, na captura do tráfego de rede, um pacote de requisição tenha sido perdido. Mas se este número for bem menor que "-1", deve-se considerar a possibilidade de ocorrência de algum comportamento anômalo, que deve ser melhor avaliado.

Além do campo *match_reply*, que apresenta algumas informações sobre o que ocorreu com uma dada sessão ICMP de informação, existem atributos que podem ser associados a uma sessão e são descritos como se segue:

- *ICMP_INFO_REPLY_FIRST*: ocorre em uma sessão, quando o primeiro pacote ICMP de informação inserido é um pacote de resposta;
- *ICMP_INFO_NOMATCH_REPLY*: ocorre quando o processador ICMP não consegue associar um pacote de resposta ao respectivo pacote de requisição, dentro de uma mesma sessão. Para realizar esta associação, o campo *sequence number* do pacote de resposta deve ser igual ao mesmo campo de algum dos pacotes de requisição na sessão.

5.3.10.2 Processador ICMP de Erro

A função ***icmp_Epkt_proc*** corresponde à implementação do processador ICMP de erro, e seu formato é apresentado como se segue:

void ***icmp_Epkt_proc****(struct icmp_Epkt **f_icmp_Epkt, struct icmp_Epkt **l_icmp_Epkt, struct ip *ip, struct icmp *icmp, u_int8_t pkt_dir)*

Quando o processador IP verifica que o pacote sendo processado contém uma mensagem ICMP de erro, este executa a chamada da função ***icmp_Epkt_proc***, preenchendo devidamente seus parâmetros.

f_icmp_Epkt e *l_icmp_Epkt* são apontadores para o início e fim dos pacotes ICMP de erro, associados aos dois IPs contidos no pacote. *ip* é o apontador para a estrutura que contém o cabeçalho IP do pacote. *icmp* é o apontador para a estrutura que contém o cabeçalho ICMP do pacote. E *pkt_dir* armazena o código da direção do pacote, como visto na seção 5.3.8.

Os pacotes ICMP de erro contém, além do cabeçalho ICMP propriamente dito, o cabe-

çalho IP do datagrama que ocasionou o erro, como visto na seção 2.4.1. Então, os dados obtidos do cabeçalho IP do pacote gerador do erro são utilizados em duas situações distintas, de acordo com o tipo da mensagem de erro gerada.

Na primeira situação, o pacote ICMP de erro não é do tipo "*time exceeded in fragmentation reassembly*". Então, os seguintes procedimentos são realizados:

a) Insere-se o pacote na lista encadeada de pacotes ICMP de erro, associada aos dois IPs, obtidos do pacote de erro.

b) Os campos correspondentes aos endereços IP de origem e destino do cabeçalho IP gerador do erro são utilizados para encontrar os nós correspondentes na árvore binária de IPs. Caso algum destes nós não seja encontrado, o atributo "*ICMP_ERROR_NOMATCH*" é associado ao pacote de erro e a função termina.

c) Verifica-se o campo *protocol* do cabeçalho IP gerador do erro. De acordo com o valor deste campo, a busca do pacote gerador do erro é direcionada para as sessões ICMP, UDP ou TCP, relacionadas aos nós localizados na árvore binária de IPs.

d) A busca é iniciada na última sessão da lista encadeada de sessões, e é realizada até a primeira sessão, atendendo a algumas condições de parada. Estas condições são apresentadas nos procedimentos seguintes.

e) Para cada sessão, a lista de pacotes nela inseridos é verificada, partindo do último pacote. Caso os campos *ip_hl*, *ip_len* e *ip_id*, do cabeçalho IP gerador do erro, sejam iguais aos respectivos campos na estrutura que armazena os dados do cabeçalho IP do pacote atual, os horários de captura são checados.

f) Se o horário do pacote ICMP de erro for menor que o horário do pacote atual, a função termina e o atributo "*ICMP_ERROR_NOMATCH*" é associado ao pacote de erro. Caso contrário, significa que o pacote gerador do erro foi encontrado. Então, o pacote ICMP de erro, inserido na lista encadeada, é associado ao atributo "*ICMP_ERROR_MATCHED*", à sessão onde o pacote gerador do erro está, e ao pacote gerador do erro propriamente dito.

Quando um pacote ICMP de erro é associado ao pacote gerador do erro, o campo *status*, na estrutura que contém os dados do cabeçalho IP do pacote gerador do erro (visto na seção 5.3.8), é atualizado. Se o pacote ICMP de erro for do tipo "*destination unreachable*", o campo *status* recebe o valor "*PACKET_STATUS_UNREACHABLE*". Caso contrário, recebe o valor "*PACKET_STATUS_MISSED*".

Já na outra situação, onde o pacote ICMP de erro é do tipo "*time exceeded in fragmentation reassembly*", os seguintes procedimentos são realizados:

a) Insere-se o pacote na lista encadeada de pacotes ICMP de erro, associada aos dois IPs, obtidos do pacote de erro. Os próximos procedimentos interagem com a árvore de desfragmentação, vista na seção 5.3.9.

b) Busca, na árvore binária de desfragmentação, o nó que contém os endereços IP de origem e destino iguais aos do cabeçalho IP do pacote gerador do erro, contido no pacote ICMP de erro. Caso encontre, busca pelo elemento de desfragmentação correspondente. Caso não encontre o nó ou o elemento de desfragmentação, a função termina.

c) Se o elemento de desfragmentação estiver no estado "*FRAG_REASSEMBLING*", este é atualizado para "*FRAG_TIME_EXCEEDED*" e a função termina.

d) Se o elemento de desfragmentação estiver no estado "*FRAG_REASSEMBLED*", significa que o pacote foi remontado e faz parte de alguma sessão. Portanto, alguns ajustes devem ser feitos. Para isto, são utilizados os apontadores *session* e *packet* da estrutura que representa o elemento de desfragmentação. O pacote remontado é, então, removido da sessão correspondente. O estado e os atributos desta sessão são atualizados quando necessário e, até mesmo a sessão é removida, se for o caso. Finalmente, o estado do elemento de desfragmentação é atualizado para "*FRAG_TIME_EXCEEDED*".

A Figura (5.14) apresenta a estrutura correspondente a cada pacote ICMP de erro (**a**) e aos dados armazenados para cada um destes pacotes (**b**).

```
struct icmp_Epkt {
  struct timeval ptv;
  u_int8_t attributes;
  struct mod_ip data_ip;
  struct mod_Eicmp data_icmp;
  struct icmp_Epkt *next;
  struct icmp_Epkt *prev;
};
```

a

```
struct mod_Eicmp {
  u_int8_t type;
  u_int8_t code;
  struct avlnode *src_node;
  struct avlnode *dst_node;
  u_int8_t proto;
  void *session;
  void *packet;
};
```

b

FIGURA 5.14 – Estruturas de armazenamento das sessões ICMP de erro.

Na estrutura *icmp_Epkt*, correspondente a cada pacote inserido na lista encadeada de pacotes ICMP de erro, o campo *ptv* armazena o horário de captura do pacote. *attributes*

contém os atributos associados ao pacote de erro. *data_ip* é uma estrutura que armazena os dados obtidos do cabeçalho IP do pacote, como visto na seção 5.3.8. *data_icmp* é uma estrutura que armazena os dados obtidos do cabeçalho ICMP e referências para o pacote gerador do erro. E *next* e *prev* são apontadores para o próximo pacote na lista de pacotes de erro e para o anterior.

Na estrutura *mod_Eicmp*, correspondente aos dados da mensagem ICMP de erro armazenados para cada pacote, o campo *type* contém o tipo da mensagem ICMP e o campo *code* contém o código. *src_node* e *dst_node* são apontadores para os nós da árvore binária de IPs, correspondentes aos IPs de origem e destino do pacote gerador do erro. *proto* contém o protocolo utilizado pelo pacote gerador do erro, e pode ser ICMP, UDP ou TCP. *session* é o apontador para a sessão onde o pacote gerador do erro foi inserido. E *packet* é o apontador para o pacote gerador do erro propriamente dito.

5.3.11 O Processador UDP

A função ***udp_session_proc*** corresponde à implementação do processador UDP, e seu formato é apresentado como se segue:

void ***udp_session_proc****(struct udp_sessions **f_udp_session, struct udp_sessions **l_udp_session, u_char *up, struct ip *ip, struct udphdr *udp, u_int8_t pkt_dir, u_int caplen, u_int length)*

Quando o processador IP verifica que o pacote sendo processado contém uma mensagem UDP, este executa a chamada da função ***udp_session_proc***, preenchendo devidamente seus parâmetros.

f_udp_session e *l_udp_session* apontam, respectivamente, para o início e fim das sessões UDP, associadas aos dois IPs contidos no pacote sendo processado. *up* aponta para o byte de início do cabeçalho UDP, na sequência de bytes capturados para o pacote. *ip* é o apontador para a estrutura contendo o cabeçalho IP. *udp* é o apontador para a estrutura contendo o cabeçalho UDP. *pkt_dir* contém o código da direção do pacote. *caplen* contém a quantidade de bytes do pacote, a partir do início do cabeçalho UDP. E *length* contém o tamanho total do datagrama IP, que contém a mensagem UDP, decrementado do tamanho do cabeçalho IP.

Uma sessão UDP tem como identificadores as duas portas contidas nas mensagens UDP dos pacotes que a compõem. Utiliza-se o par <*portA*,*portB*> para identificar a sessão, onde a primeira está associada ao menor endereço IP dos pacotes, e a segunda está associada ao maior endereço IP. Observe que não há a preocupação com a idéia de porta

de origem e destino, pois existe a codificação da direção do pacote, como visto na seção 5.3.8.

Quando um pacote UDP é processado, a lista encadeada de sessões UDP, associada aos dois endereços IP do pacote, é verificada. Caso exista uma sessão, identificada pelo mesmo par de portas contido no pacote sendo processado, este pacote é inserido na lista encadeada de pacotes desta sessão. Caso contrário, uma nova sessão é criada e o pacote é então inserido em sua lista de pacotes.

O processador UDP propõe e implementa duas extensões para o tratamento das aplicações de DNS (Mockapetris, 2002b) e NTP (Mills, 2002c) (Mills, 2002b) (Mills, 2002c). Sabe-se que estas aplicações baseiam-se em requisições e respostas, mas para identificar se um pacote DNS ou NTP é uma requisição ou uma resposta, é preciso avaliar o conteúdo da mensagem UDP.

Para o DNS, o primeiro byte do conteúdo da mensagem UDP contém o identificador da requisição/resposta, e o primeiro bit do segundo byte diz se o pacote é uma requisição ou uma resposta. Já para o NTP, o primeiro byte do conteúdo informa a versão do protocolo NTP sendo utilizado e o modo de operação do pacote, informando assim se este está operando como uma requisição ou como uma resposta.

Sabe-se que o arquivo que contém o tráfego de rede armazena, no máximo, 68 bytes por pacote. Portanto, pode-se considerar a seguinte divisão para os pacotes UDP: 14 bytes do cabeçalho *Ethernet*, 20 bytes do cabeçalho IP sem opções e 8 bytes do cabeçalho UDP. Então, restam 26 bytes para armazenar o conteúdo da mensagem UDP, que permitem a obtenção dos bytes iniciais para as aplicações de DNS e NTP.

Existem dois critérios, utilizados para definir se um pacote DNS é uma requisição ou uma resposta. É aplicada uma máscara de bits ao segundo byte do conteúdo do pacote, para obter apenas o valor do bit desejado. Se este bit for igual a “0”, então o pacote é uma requisição. Se for “1”, então este é uma resposta.

Já para os pacotes NTP, outros critérios são utilizados para definir se um pacote NTP é uma requisição ou uma resposta. Primeiro é aplicada uma máscara de bits ao primeiro byte do conteúdo do pacote, para obter a versão do protocolo NTP sendo utilizado. E segundo, é aplicada outra máscara de bits sobre o mesmo byte, para identificar o modo de operação do pacote. A Tabela (5.2) apresenta estes critérios:

TABELA 5.2: Identificação do tipo do pacote NTP

Versão	*Modo de Operação*	*Verificação*	*Tipo do pacote*
igual a 1	0 (*unspecified*)	porta de destino igual a 123	requisição (*request*)
		porta de origem igual a 123	resposta (*reply*)
maior que 1	1 (*symmetric active*)	—	requisição (*request*)
	2 (*symmetric passive*)	—	resposta (*reply*)
	3 (*client*)	—	requisição (*request*)
	4 (*server*)	—	resposta (*reply*)

Para as sessões UDP, associadas às aplicações de DNS e NTP, é feita a alocação de um inteiro de 4 bytes, para armazenar a contagem da diferença no número de pacotes de requisição e de resposta. Este contador tem a mesma funcionalidade do campo *match_reply*, associado às sessões ICMP de informação, vista na seção 5.3.10.1, e é associado a cada sessão UDP deste gênero.

Portanto, para cada pacote UDP, onde é caracterizada a utilização das aplicações de DNS e NTP, juntamente com os dados obtidos da mensagem UDP contida no pacote, são armazenados os seguintes campos: para o DNS, o campo *id* de um byte, que contém a identificação do pacote DNS e é utilizado no casamento das respostas com as respectivas requisições, e o campo *flags* também de um byte, que contém o bit que informa se o pacote é uma requisição ou resposta; e para o NTP, o campo *status* de um byte, que contém a versão e o modo de operação do pacote.

Existem, ainda, alguns atributos associados às sessões deste gênero, e são apresentados como se segue:

- *DNS_REPLY_FIRST e NTP_REPLY_FIRST*: associados à sessão UDP de DNS e NTP, respectivamente, quando o primeiro pacote inserido é uma resposta;
- *DNS_REPLY_NOMATCH e NTP_REPLY_NOMATCH*: associados à sessão UDP de DNS e NTP, respectivamente, quando não é encontrado o pacote de requisição relacionado ao pacote de resposta sendo processado;
- *DNS_TRUNC_DATA e NTP_TRUNC_DATA*: associados à sessão UDP de DNS e NTP, respectivamente, quando o processador UDP não consegue obter os bytes de conteúdo, necessários para identificar as requisições e respostas;

- *DNS_EXCEEDED_PAYLOAD*: associado a sessão UDP de DNS, quando o tamanho do conteúdo da mensagem UDP excede o máximo permitido. Segundo Mockapetris (2002b), o tamanho máximo de um pacote DNS, utilizando o protocolo UDP, é de 512 bytes.

A Figura (5.15) apresenta as estruturas utilizadas no armazenamento: de cada sessão na lista encadeada de sessões UDP (**a**); de cada pacote na lista encadeada de pacotes UDP de uma sessão (**b**); dos dados obtidos da mensagem UDP contida no pacote (**c**); dos dados específicos, no caso da aplicação DNS (**d**); e dos dados específicos, no caso da aplicação NTP (**e**).

```
struct udp_sessions {
  u_int8_t attributes;
  u_int16_t portA;
  u_int16_t portB;
  void *spec;
  struct udp_pkt *first_udp_pkt;
  struct udp_pkt *last_udp_pkt;
  struct udp_sessions *next;
  struct udp_sessions *prev;
};
```
a

```
struct udp_pkt {
  struct timeval ptv;
  struct mod_ip data_ip;
  struct mod_udp data_udp;
  struct udp_pkt *next;
  struct udp_pkt *prev;
};
```
b

```
struct mod_udp {
  u_int16_t len;
  void *app_data;
};
```
c

```
struct dns_data {
  u_int16_t id;
  u_int8_t flags;
};
```
d

```
struct ntp_data {
  u_int8_t status;
};
```
e

FIGURA 5.15 – Estruturas de armazenamento das sessões UDP.

Na estrutura *udp_sessions*, correspondente a cada sessão da lista encadeada de sessões UDP, *attributes* contém os atributos da sessão. *portA* e *portB* contém as portas associadas à sessão UDP, onde estas referem-se ao maior e menor endereço IP, respectivamente. *spec* é um apontador para informações específicas, associadas à aplicação utilizando o protocolo UDP, no caso de aplicações tratadas pelo processador UDP. *first_udp_pkt* e *last_udp_pkt* são apontadores para o primeiro e para o último pacote, na lista encadeada de pacotes UDP da sessão. E *next* e *prev* são apontadores para a próxima sessão UDP e para a anterior.

Na estrutura *udp_pkt*, correspondente a cada pacote da lista encadeada de pacotes UDP de uma sessão, *ptv* contém o horário de captura do pacote UDP. *data_ip* é a estrutura que contém os dados obtidos do cabeçalho IP do pacote, como visto na seção 5.3.8. *data_udp* é a estrutura que contém os dados obtidos do cabeçalho UDP do pacote. E *next* e *prev* são apontadores para o próximo pacote e para o anterior.

Na estrutura *mod_udp*, correspondente ao dados armazenados do cabeçalho UDP de cada

pacote, *len* contém o tamanho da mensagem UDP, obtido da estrutura apontada por *udp*. E *app_data* é o apontador para dados específicos da aplicação utilizando o protocolo UDP, no caso de aplicações tratadas pelo processador UDP.

Na estrutura *dns_data*, correspondente aos dados específicos armazenados para a aplicação DNS, *id* contém a identificação da requisição ou resposta DNS. E *flags* contém os bits necessários para identificar se o pacote é uma requisição ou uma resposta DNS.

Na estrutura *ntp_data*, correspondente aos dados específicos armazenados para a aplicação NTP, *status* contém os bits necessários para identificar se o pacote é uma requisição ou uma resposta NTP.

Para o caso das sessões UDP de DNS e NTP, o apontador *spec*, da estrutura *udp_sessions*, referencia o contador da diferença no número de requisições e respostas. E o apontador *app_data*, da estrutura *udp_pkt*, referencia a estrutura *dns_data* ou *ntp_data*, de acordo com a aplicação. Para os demais casos, estes apontadores são iguais a **NULL**.

5.3.12 O Processador TCP

A função ***tcp_session_proc*** corresponde à implementação do processador TCP, e seu formato é apresentado como se segue:

void ***tcp_session_proc****(struct tcp_sessions **f_tcp_session, struct tcp_sessions **l_tcp_session, struct ip *ip, struct tcphdr *tcp, u_int8_t pkt_dir, u_int caplen, u_int length)*

Quando o processador IP verifica que o pacote sendo processado contém uma mensagem TCP, este executa a chamada da função ***tcp_session_proc***, preenchendo devidamente seus parâmetros.

f_tcp_session e *l_tcp_session* são apontadores para o início e fim das sessões TCP, associadas aos dois IPs contidos no pacote sendo processado. *ip* é o apontador para a estrutura contendo o cabeçalho IP. *tcp* é o apontador para a estrutura contendo o cabeçalho TCP. *pkt_dir* contém o código da direção do pacote. *caplen* contém a quantidade de bytes capturados para o pacote, a partir do início do cabeçalho TCP. E *length* contém o tamanho total do datagrama IP, que contém a mensagem TCP, decrementado do tamanho do cabeçalho IP.

Uma sessão TCP tem como identificadores as duas portas contidas nas mensagens TCP dos pacotes que a compõem. Seguindo o mesmo esquema de implementação do processador UDP, utiliza-se o par <*portA*,*portB*> para identificar a sessão, onde a primeira

está associada ao menor endereço IP dos pacotes, e a segunda está associada ao maior endereço IP. Observe que também não há a preocupação com a idéia de porta de origem e destino, pois existe a codificação da direção do pacote, como visto na seção 5.3.8.

Outra informação, que determina se um pacote pertence ou não a uma sessão TCP, é o seu estado. Quando um pacote contém um par de portas, que casa com uma sessão TCP previamente criada, se o estado desta sessão informar que ela já foi terminada, uma nova sessão TCP deverá ser criada, e este pacote TCP deverá ser incluído na lista encadeada de pacotes desta nova sessão.

O primeiro passo, realizado pela função ***tcp_session_proc***, é verificar que ação deve ser tomada para o pacote TCP sendo processado. Para isto é utilizada uma subrotina, que verifica o campo *flags* da estrutura apontada por *tcp*, correspondente ao cabeçalho TCP do pacote, e que leva em consideração o estado das sessões TCP. Existem dois casos, tratados por esta subrotina.

No Caso 1, não existem sessões TCP, associadas ao par de endereços IP contido no pacote. Então, a subrotina faz as devidas checagens, observando que não existe um estado associado, pois ainda não existe uma sessão TCP, onde o pacote deva ser inserido.

E no caso 2, existem sessões TCP associadas. Então, antes de executar a subrotina, é iniciada uma busca por uma sessão, identificada pelo mesmo par de portas do pacote TCP sendo processado. Se esta sessão existir, o seu estado é checado, e se este não informar que a sessão foi terminada, é utilizado para direcionar as checagens realizadas pela subrotina. De outra forma, a execução ocorre de maneira semelhante ao caso 1.

Ao terminar as checagens dos *flags*, contidos no cabeçalho TCP do pacote, a subrotina retorna um código, referente a ação que deve ser tomada para o pacote sendo processado.

O código *PROCEED* informa que os *flags* são válidos e que deve-se proceder com o processamento do pacote, sendo que este pode ou não estar associado a uma sessão TCP previamente criada.

Códigos *DROP_STH_ACT_{FXT,SAPU,FIN,NULL,NXT,VECNA,UNKNOWN}*[5] indicam que o pacote deve ser descartado e inserido na árvore de *logs* TCP (a ser vista na seção 5.3.12.1), pois uma combinação de *flags* indesejada foi observada. O pacote não é associado a uma sessão TCP, e a função termina.

O código *DROP_RESET* indica que o pacote deve ser descartado e inserido na árvore

[5]As chaves ({ }) indicam que são sete códigos diferentes, onde o prefixo do nome dado ao código é semelhante, e a ação a ser tomada é a mesma.

de *logs* TCP, pois o *flag* RST está ativo e o pacote não foi associado a uma sessão TCP. Este código faz com que a função termine.

Códigos $LOG\_TWH\_STH\_ACT\_\{FXT,SAPU,FIN,NULL,NXT,VECNA,UNKNOWN\}^5$ indicam que deve-se proceder com o processamento do pacote, mas este deve ser inserido na árvore de *logs* TCP, pois contém uma combinação de *flags* que caracterizam uma atividade *stealth*. Atividades *stealth* são aquelas que tentam iludir um mecanismo de defesa, por exemplo, um *Firewall*. O pacote é associado à sessão TCP, e está contido em seu *three-way handshake*.

O código *LOG_TWH_WEIRD_BEHAVIOR* indica que deve-se proceder com o processamento do pacote, mas este deve ser inserido na árvore de *logs* TCP, pois caracteriza uma atividade não esperada, durante o *three-way handshake* de uma sessão TCP. O pacote é associado à sessão correspondente.

Códigos $LOG\_STH\_ACT\_\{FXT,SAPU,FIN,NULL,NXT,VECNA,UNKNOWN\}^5$ indicam que deve-se proceder com o processamento do pacote, mas este deve ser inserido na árvore de *logs* TCP, pois contém uma combinação de *flags* que caracterizam uma atividade *stealth*. O pacote é associado à sessão TCP, e está contido em seu fluxo de dados.

Os códigos *LOG_DATA_SEQN_GT_FIN1* e *LOG_DATA_SEQN_GT_FIN2* indicam que deve-se proceder com o processamento do pacote, mas este deve ser inserido na árvore de *logs* TCP, pois caracteriza o envio de dados, onde a mensagem TCP tem o *sequence number* maior que o *sequence number* do pacote FIN, no mesmo sentido. O pacote é associado à sessão TCP correspondente.

E finalmente, *LOG_ACK_GT_FIN1_SEQNplus1* e *LOG_ACK_GT_FIN2_SEQNplus1* são códigos que indicam que deve-se proceder com o processamento do pacote, mas este deve ser inserido na árvore de *logs* TCP, pois caracteriza o envio de um ACK, onde a mensagem TCP tem o *acknowledgement number* maior que o "*sequence number* + 1" do pacote FIN, no sentido oposto. O pacote é associado à sessão TCP correspondente.

A Tabela (5.3) apresenta as combinações de *flags*, para os códigos de retorno da subrotina, relacionados às atividades *stealth*. Convenciona-se que a ocorrência do *flag* URG é representada por "U", ACK por "A", PSH por "P", RST por "R", SYN por "S" e FIN por "F".

TABELA 5.3: Atividades *stealth*

Sufixo no código da ação	*Flags*	*Nome da atividade*
STH_ACT_FXT	UAPRSF	*full xmas tree*
STH_ACT_SAPU	SAPU	*sapu*
STH_ACT_FIN	F	*fin*
STH_ACT_NULL	ausência de *flags*	*null*
STH_ACT_NXT	FPU	*nmap*[6] *xmas tree*
STH_ACT_VECNA	U, PF, UP, P	*vecna*
STH_ACT_UNKNOWN	*flags* não esperados	atividade desconhecida (*unknown*)

Caso o código de retorno da subrotina, que determina a ação a ser tomada, permita a continuidade do processamento do pacote, uma outra subrotina é executada, e tem como finalidade atualizar o estado da sessão TCP associada. Para o caso onde uma nova sessão é criada, estabelece o estado inicial para esta.

Essa subrotina implementa um diagrama de transição de estados, que é uma variante do diagrama proposto na RFC de especificação do protocolo TCP (Postel, 2002d). O diagrama implementado considera o fluxo bidirecional de informações na conexão, diferente do diagrama tradicional, onde estados distintos são criados nos dois extremos da conexão.

A Figura (5.16) apresenta o diagrama proposto e implementado na subrotina de atualização do estado de uma sessão TCP. Combinações de *flags* foram suprimidas do diagrama, para facilitar seu entendimento.

Pode-se observar que um pacote contendo a combinação de *flags* SYN/FIN ocasiona a transição do estado de partida "***No State***" para o estado inicial "***SYN***". Esta condição foi considerada, pois vários sistemas operacionais, baseados na plataforma **UNIX**, tais como **Linux**, **FreeBSD**, **OpenBSD** e **Solaris**, e o sistema operacional **Windows**, responderam com um pacote SYN/ACK, nesta situação. Para realizar estes testes a ferramenta *hping*[7] foi utilizada.

Algumas observações são feitas em relação ao diagrama da Figura 5.16 e são apresentadas como se segue:

a) Os estados, cujos nomes contém "**FIN1**" e "**FIN2**", indicam a ocorrência do

[6]Caracteriza a possível utilização da conhecida ferramenta de varredura (*scanner*) nmap. Página Web em `http://www.insecure.org/nmap`.

[7]Página Web em: `http://www.hping.org`.

primeiro e do segundo pacote FIN, relacionados ao término da sessão TCP, respectivamente;

b) Os estados, que correspondem ao término normal de uma sessão, identificados por **FIN{1,2}-A**, indicam que o pacote FIN foi enviado pelo menor endereço IP associado à sessão. E os identificados por **FIN{1,2}-B** indicam que este foi enviado pelo maior endereço IP. Já para os estados identificados por **ACK/FIN{1,2}-A** e **ACK/FIN{1,2}-B**, indicam que o pacote ACK/FIN foi recebido pelo menor endereço IP e pelo maior, respectivamente;

c) Os estados que contém a palavra "***Closed***", indicam que a sessão foi terminada;

d) Os estados, que correspondem ao término normal de uma sessão, e identificados pela palavra "***Sim.***", indicam o fechamento simultâneo da sessão.

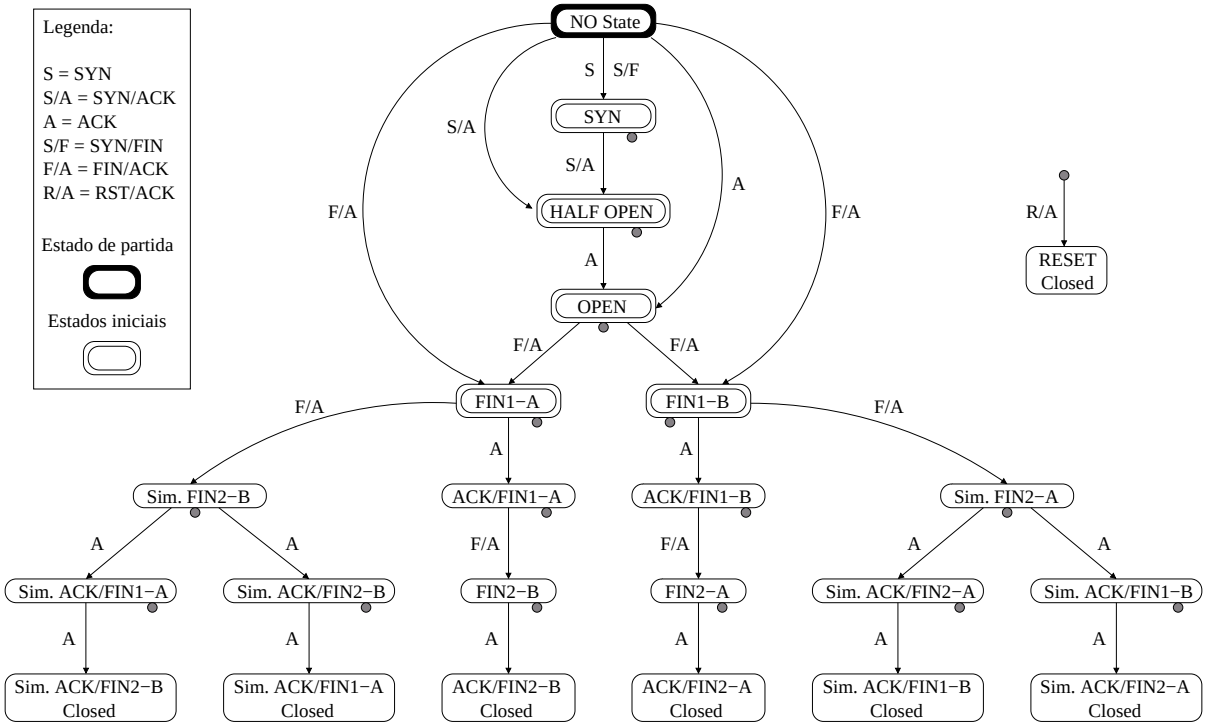

FIGURA 5.16 – Diagrama de transição de estados para as sessões TCP.

A cada sessão TCP, é associado o campo *server*, onde procura-se identificar qual par <endereço IP, porta> é o servidor na sessão. Para isto, a sessão deve ter passado pelo estado ***SYN***, ou pelo menos pelo ***HALF-OPEN***. No primeiro caso, o par que recebeu o pacote SYN é identificado como servidor. Já no segundo caso, o par que enviou o pacote SYN/ACK é identificado como servidor. Caso o servidor esteja relacionado ao menor IP da sessão, o valor "**1**" é associado ao campo *server*. Caso esteja relacionado ao maior IP,

o valor "**2**" é associado. Se não houver possibilidade de realizar esta associação, o campo recebe o valor "**0**".

Para verificar e efetivar o término normal de uma sessão TCP, os *sequence numbers* dos pacotes FIN, enviados pelos dois endereços IP associados à sessão, são armazenados. Estes são utilizados para checar se os pacotes posteriores aos pacotes FIN têm *sequence numbers* e *acknowledgement numbers* válidos.

Quando o processador TCP observa um pacote RST, este procura associá-lo a uma sessão TCP. Caso a associação seja bem sucedida, o *sequence number* e/ou *acknowledgement number* do pacote são checados em relação à sessão associada. Se estes forem válidos, o estado da sessão é atualizado para "***RESET Closed***", e indica o término abrupto da sessão. Caso a associação com uma sessão TCP não seja possível, a árvore de *logs* é checada. Se o pacote de RST for associado a algum *log*, o pacote de *log* em questão recebe um marcador, dizendo que um RST foi associado a ele.

Então, para sumarizar o funcionamento do processador TCP, depois de avaliar que ação deve ser tomada para o pacote sendo processado, se for permitido dar continuidade ao processamento do pacote, é verificado se este deve ser associado a uma sessão TCP. Caso positivo, o estado da sessão é atualizado, e o pacote é inserido na lista encadeada de pacotes TCP, associada à sessão.

Ainda existem alguns atributos que podem ser associados a uma sessão TCP, e são descritos como se segue:

- *COLD_START*: indica que a sessão não teve um início esperado. Normalmente, está relacionado à perda de pacotes no processo de coleta do tráfego de rede e, principalmente, de parte ou todo o *three-way handshake*. Está associado aos estados iniciais ***HALF-OPEN***, ***OPEN***, ***FIN1-A***, ***FIN1-B***;
- *RES1_SET_ON_SYN* e *RES2_SET_ON_SYN*: indicam a presença dos *flags* RES1 e RES2 no pacote SYN, respectivamente;
- *DATA_ON_SYN*: indica que dados foram enviados no pacote SYN;
- *RES1_SET_ON_SYN_FIN* e *RES2_SET_ON_SYN_FIN*: indicam a presença dos *flags* RES1 e RES2 no pacote SYN/FIN,respectivamente;
- *STEALTH_ACTIVITY_SYN_FIN*: quando ocorre um pacote SYN/FIN, a sessão recebe este atributo;
- *RES1_SET_ON_SYN_ACK* e *RES2_SET_ON_SYN_ACK*: indicam a presença dos *flags* RES1 e RES2 no pacote SYN/ACK, respectivamente;

- *SYN_RESET*: indica que a sessão estava no estado ***SYN*** e foi terminada abruptamente;
- *HALF_OPEN_RESET*: indica que a sessão estava no estado ***HALF-OPEN*** e foi terminada abruptamente;

A Figura (5.17) apresenta as estruturas utilizadas no armazenamento: de cada sessão na lista encadeada de sessões TCP (**a**); de cada pacote na lista encadeada de pacotes TCP de uma sessão (**b**); dos dados obtidos da mensagem TCP contida no pacote (**c**); e dos números de sequência dos pacotes TCP utilizados na solicitação do término de conexão (**d**).

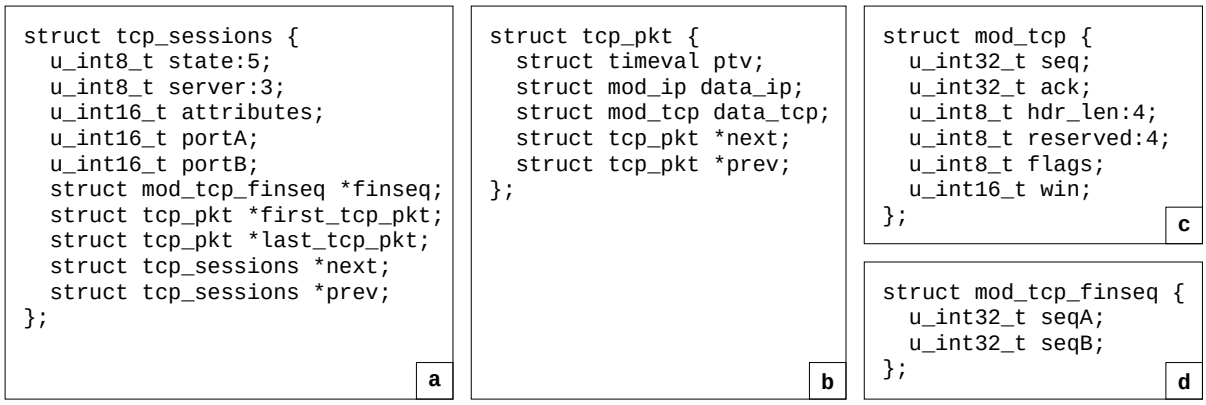

FIGURA 5.17 – Estruturas de armazenamento das sessões TCP.

Na estrutura *tcp_sessions*, correspondente a cada sessão TCP, o campo *state* armazena o estado atual da sessão. *server* contém o código, que diz qual par <endereço IP, porta> é o servidor, caso seja possível determinar. *attributes* contém os atributos associados à sessão. *portA* e *portB* contém as portas associadas à sessão TCP, onde estas referem-se ao maior e menor endereço IP, respectivamente. *finseq* é um apontador para a estrutura, que contém os números de sequência dos pacotes FIN, enviados por cada endereço IP associado à sessão. *first_tcp_pkt* e *last_tcp_pkt* são apontadores para o início e fim da lista encadeada de pacotes TCP, associados à sessão. E *next* e *prev* são apontadores para a próxima sessão e para a anterior.

Na estrutura *tcp_pkt*, correspondente a cada pacote da lista encadeada de pacotes TCP de uma sessão, *ptv* contém o horário de captura do pacote TCP. *data_ip* é a estrutura que contém os dados obtidos do cabeçalho IP do pacote, como visto na seção 5.3.8. *data_tcp* é a estrutura que contém os dados obtidos do cabeçalho TCP do pacote. E *next* e *prev* são apontadores para o próximo pacote e para o anterior.

Na estrutura *mod_tcp*, correspondente aos dados armazenados para o cabeçalho TCP de

cada pacote, *seq* contém o *sequence number*, *ack* o *acknowledgement number*, *hdr_len* o tamanho do cabeçalho TCP, *reserved* os bytes do campo *reserved*, *flags* os *flags*, e *win* o *window size*. Todos estes dados são obtidos da estrutura apontada por *tcp*, correspondente ao cabeçalho da mensagem TCP.

Na estrutura *mod_tcp_finseq*, *seqA* contém o *sequence number* do pacote FIN, enviado pelo menor endereço IP. E *seqB* contém o *sequence number* do pacote FIN, enviado pelo maior endereço IP.

5.3.12.1 A Árvore de *Logs* TCP

A árvore binária de *logs TCP* armazena os pacotes, correspondentes aos *logs* gerados pelo processador TCP. Cada nó da árvore é identificado pelo par <endereço IP de origem, endereço IP de destino>, e constitui uma lista encadeada de pacotes de *logs* TCP, com o mesmo par de endereços do nó da árvore.

A Figura (5.18) apresenta as estruturas correspondentes a: cada nó da árvore (**a**), e cada pacote de *log* TCP, associado ao nó (**b**).

```
struct tcp_log_node {
  u_int32_t ip_A;
  u_int32_t ip_B;
  struct tcp_log_data *first_log;
  struct tcp_log_data *last_log;
  struct tcp_log_node *left;
  struct tcp_log_node *right;
  char balance;
};
```

a

```
struct tcp_log_data {
  u_int8_t log_type:7;
  u_int8_t ass_reset:1;
  struct timeval ptv;
  struct mod_ip data_ip;
  u_int16_t portA;
  u_int16_t portB;
  struct mod_tcp data_tcp;
  struct tcp_log_data *next;
  struct tcp_log_data *prev;
};
```

b

FIGURA 5.18 – Estruturas de armazenamento da árvore de *logs* TCP.

Na estrutura *tcp_log_node*, correspondente a cada nó da árvore, os campos *ip_A* e *ip_B* contém os endereços IP de origem e destino, identificadores do nó. *first_log* e *last_log* são apontadores para o início e para o fim da lista encadeada de pacotes de *log* associados ao nó. *left* e *right* apontam para as sub-árvores esquerda e direita. E *balance* é utilizado no balanceamento da árvore binária de *logs*.

Na estrutura *tcp_log_data*, correspondente a cada pacote de *log* associado a um nó, o campo *log_type* armazena o tipo do pacote de *log*, atribuído pelo processador TCP. *ass_reset* diz se um pacote RST foi associado ao pacote de *log*. *ptv* contém o horário de captura do pacote. *data_ip* é a estrutura que contém os dados obtidos do cabeçalho IP do pacote,

como visto na seção 5.3.8. *portA* e *portB* contém as portas associadas ao pacote, onde estas referem-se ao maior e menor endereço IP, respectivamente. *data_tcp* é a estrutura que contém os dados obtidos do cabeçalho TCP do pacote. E *next* e *prev* são apontadores para o próximo pacote de *log* e para o anterior.

5.3.13 O Gerador de Resultados

O gerador de resultados é responsável por apresentar as informações contidas nas estruturas de armazenamento das sessões TCP/IP reconstruídas, bem como da árvore de desfragmentação e da árvore de *logs* TCP. É implementado através de um conjunto de rotinas destinadas a estas operações.

De acordo com os argumentos obtidos da linha de comandos (como visto na seção 5.3.5) e com os parâmetros obtidos do arquivo de configurações (visto na seção 5.3.4), direciona a geração dos resultados, para que sejam apresentados apenas os desejados pelo usuário.

Por exemplo, se na execução do *RECON*, os argumentos "**-T**" e "**-vv**" forem especificados, todas as sessões TCP serão apresentadas, onde para cada sessão, todos os seus pacotes serão mostrados. Além disso, os resultados serão apresentados no nível *verbose* 2, o que significa que estes terão o maior nível de detalhamento.

Se o argumento "**-f**" for especificado, a árvore de desfragmentação deve ser verificada, e as informações dos elementos de desfragmentação de cada nó, associadas a alguma situação de alarme, como por exemplo, a sobreposição de fragmentos (*overlaping*), devem compor os resultados gerados.

E se o argumento "**-l**" for especificado, todos os pacotes da árvore de *logs* TCP devem ser apresentados, mas de forma colapsada. Isto significa que pacotes que tiverem características muito semelhantes devem ser suprimidos, e apenas um deve ser apresentado, mas com a contagem do número de vezes que se repetiu.

O gerador de resultados dispõe de um mecanismo, que faz o caminhamento pré-ordem (Gonnet e Baeza-Yates, 1991) das árvores binárias balanceadas, de modo que os resultados apresentados sejam ordenados do menor endereço IP para o maior endereço. Com isto, fica fácil localizar um determinado endereço, dentre todos os resultados gerados.

Além disso, os nós das árvores binárias, cujas informações já foram apresentadas, são marcados, evitando assim que resultados sejam duplicados.

CAPÍTULO 6

RESULTADOS OBTIDOS

Este capítulo apresenta os resultados obtidos pelo *RECON*, na reconstrução de sessões TCP/IP e em algumas aplicações relacionadas à detecção de intrusão. É apresentada a forma de obtenção dos dados utilizados na análise dos resultados, bem como uma breve discussão sobre o desempenho da ferramenta desenvolvida.

6.1 OBTENÇÃO DOS DADOS UTILIZADOS NA ANÁLISE

A análise dos resultados gerados pelo *RECON*, relacionados às atividades de reconstrução de sessões TCP/IP, e sua aplicação em algumas atividades relacionadas à detecção de intrusão, fez uso do tráfego direcionado às redes do Instituto Nacional de Pesquisas Espaciais e no tráfego de rede gerado em ambiente de laboratório experimental.

Para a coleta do tráfego de rede do Instituto, foi usado o componente sensor do sistema de detecção de intrusão SHADOW, posicionado estrategicamente de modo a coletar pacotes entrando e saindo de sua rede de comunicação de dados. Foram capturados apenas 68 bytes por pacote e estes foram armazenados em arquivos, de hora em hora.

Cada arquivo gerado foi, então, transferido para uma estação de análise, instalada na rede interna do Instituto. O *RECON* foi executado nesta estação e, a cada hora, realizou-se a análise dos pacotes contidos no arquivo transferido. Além disso, pacotes gerados em ambiente de laboratório foram utilizados no estudo. A principal função deste tráfego de rede experimental foi apresentar alguns casos que não puderam ser observados no tráfego de rede do Instituto.

Algumas considerações foram feitas na apresentação dos resultados, onde são analisadas a reconstrução de sessões e algumas aplicações relacionadas à detecção de intrusão. Estas considerações são apresentadas como se segue:

- Algumas linhas foram quebradas e, em alguns casos, informações nelas contidas foram suprimidas por questões de legibilidade. Os IPs reais foram substituídos por nomes fictícios que, em algumas situações, estão associados ao serviço utilizado na troca de informações;
- Com exceção de algumas situações específicas, não é feita a distinção entre o tráfego experimental e o tráfego real coletado pelo sensor, por questões de sigilo das informações observadas.
- Como a quantidade de dados analisados é consideravelmente grande, os resul-

tados apresentados foram divididos em seções, onde parte do tráfego de rede contido em diferentes arquivos foram selecionados, para ilustrar a funcionalidade da ferramenta desenvolvida.

6.2 DESEMPENHO E ALOCAÇÃO DE RECURSOS

Os arquivos contendo o tráfego de rede, coletados entre os dias 12/07/2002 e 06/08/2002, foram selecionados para a realização de uma breve análise do desempenho do *RECON*, bem como da utilização de recursos na estação de análise onde este é executado.

No período selecionado foram coletados **806.325.074** pacotes, representando uma média diária de 31.012.503 pacotes e, consequentemente, 1.292.188 pacotes por hora. Foi, então, aplicado um filtro sobre os arquivos contendo estes pacotes, de modo que todo o tráfego HTTP relacionado à porta 80/tcp foi excluído da análise. Após a filtragem restaram **536.100.574** pacotes, representando 66,49% do total coletado, com uma média diária de 20.619.253 pacotes e, consequentemente, **859.136** pacotes por hora.

A Tabela (6.1) apresenta as médias horárias de memória alocada na execução do *RECON* e do tempo gasto em sua execução. Também são apresentados os tempos gastos na criação de todo o modelo de reconstrução de sessões e na apresentação dos resultados. A soma destes dois tempos é igual ao tempo total de execução.

TABELA 6.1: Médias horárias calculadas sobre o tráfego do período.

Média horária	*Contagem*
alocação de memória	31,96 Mbytes
tempo gasto na execução	75,091 segundos
tempo gasto na geração do modelo	29,697 segundos
tempo gasto na impressão dos resultados	45,393 segundos

Atualmente, o *hardware* disponível possui uma quantidade de memória bem superior a taxa de alocação observada na execução do *RECON*. Outro aspecto a ser considerado é que a estação de análise de dados utilizada não estava exclusivamente dedicada às tarefas de análise, alocando assim apenas parte dos recursos disponíveis. Em vista disto, a alocação de recursos da estação, relacionada à utilização de memória, foi considerada satisfatória.

O tempo gasto na apresentação dos resultados é razoavelmente maior que o tempo gasto

na geração do modelo, visto que o primeiro envolve operações de I/O (entrada/saída), onde o resultados são enviados para arquivo ou mesmo apresentados na tela, enquanto o segundo envolve apenas operações em memória.

A Tabela (6.2) apresenta as médias de tempos gastos no processamento de cada pacote, divididas entre geração do modelo e apresentação dos resultados. O cálculo consiste na divisão dos tempos observados na Tabela (6.1) pelo total de pacotes analisados.

TABELA 6.2: Média dos tempos por pacote nas etapas de execução.

Média sobre o tempo	*Cálculo*	*Tempo/Pacote (µs/pkt)*
gasto na execução	75,091 / 859136	87,403
gasto na geração do modelo	29,697 / 859136	34,566
gasto na impressão dos resultados	45,393 / 859136	52,836

Apesar dos números associados aos tempos gastos no processamento de cada pacote serem relativamente pequenos, em particular o tempo gasto na geração do modelo, alguns aperfeiçoamentos na implementação do *RECON* são propostos no capítulo 7, com o intuito de melhorar o seu desempenho.

E finalmente, a Tabela (6.3) apresenta a média de área de armazenamento alocada por pacote analisado. O cálculo consiste na divisão da memória alocada, observada na Tabela (6.1), pelo número total de pacotes analisados.

TABELA 6.3: Média de área de armazenamento alocada por pacote.

Média sobre	*Cálculo*	*Bytes/Pacote (b/pkt)*
memória alocada	(31,96*1024*1024) / 859136	39,01

Este número é justificado pela quantidade consideravelmente maior de pacotes TCP, observados no tráfego de rede analisado. Isto se deve ao fato de que os pacotes TCP contém mais informações a serem armazenadas, quando comparados aos pacotes ICMP e UDP. A próxima seção ilustra esta observação.

6.3 ESTATÍSTICAS SUMARIZADAS

O *RECON* adiciona, ao final dos resultados gerados, um conjunto de informações estatísticas, obtidas dos contadores globais do sistema. Estes contadores, apresentados no apêndice B.1, são atualizados durante o processamento dos pacotes contidos em um arquivo, para uma determinada hora. Então, estes valores acumulados são agrupados e apresentados, como mostra o exemplo a seguir.

```
=====================================================================
| Graph Numbers |
+---------------+
Num. IPs (vertex)                                          7528
Num. connections (arests)                                 15037
=====================================================================
| Packet Numbers |
+----------------+
Total packets                                           2985410
Non TCP/IP packets                                            0
Non IP packets                                                0
=====================================================================
| Ethernet Numbers |
+------------------+
Ethernet packets with truncated header                        0
=====================================================================
| IP Numbers |
+------------+
IP datagrams with truncated header                            0
IP datagrams with truncated content                       14297
IP datagrams with options                                     0
IP fragmented datagrams                                       0
IP datagrams with protocols different
  from TCP/UDP/ICMP or Multicast                              0
Multicast messages (IP in IP)                                 0
=====================================================================
| TCP Numbers |
+-------------+
TCP messages with truncated header                            0
-------------
TCP packets inserted in the model                       2906587
TCP packets with bad header length                            1
TCP packets with options                                510796
TCP log packets                                           6381
TCP reset packets (nomatch)                                759
=====================================================================
| UDP Numbers |
+-------------+
UDP messages with truncated header                            0
-------------
UDP packets inserted in the model                         75049
=====================================================================

                                                    continua...
```

```
=====================================================================
| ICMP Numbers |
+--------------+
ICMP messages with truncated header                              0
ICMP messages with unknown or reserved type                      0
ICMP messages - router solicitation                              0
ICMP messages - router advertisement                             7
ICMP messages - time exceeded in frag.                           0
-------------
ICMP packets inserted in the model                            3767
- Info messages
  Requisition ICMP info packets                                1433
  Reply ICMP info packets                                       924
- Error messages
  ICMP error packets                                           1410
  Alarm ICMP error packets (no match)                           142
  ICMP time exceeded in frag. (no match)                          0
=====================================================================
```

O primeiro grupo de informações apresenta a quantidade de endereços IP distintos observados no tráfego da hora em questão, e o número de interligações realizadas entre pares de IPs. Então, o número total de pacotes processados é mostrado e informações sobre o processamento de cada protocolo, desde a camada de enlace até a camada de transporte, são apresentadas.

Pode-se observar que o número de pacotes TCP processados e efetivamente inseridos no modelo é consideravelmente maior do que os pacotes dos outros protocolos e, apesar deste exemplo representar apenas os dados analisados para uma hora, esta característica foi uma constante no período considerado.

Outra observação interessante diz respeito ao número de pacotes TCP com o *flag* RST ativo, e que não foram associados a alguma sessão. Este número expressivo fornece um indicativo de que endereços IP da rede monitorada estão, provavelmente, sendo forjados (*spoofing*) por terceiros para a realização de ataques ou varreduras de máquinas externas.

6.4 RECONSTRUÇÃO DAS SESSÕES ICMP

A reconstrução das sessões ICMP é dividida em duas partes: a associação dos pacotes ICMP de erro aos respectivos pacotes geradores dos erros, e a reconstrução das sessões ICMP de informação. Estas partes são apresentadas nas subseções seguintes.

6.4.1 Pacotes ICMP de Erro

A saída *tcpdump* abaixo apresenta pacotes ICMP de erro, do tipo "*host unreachable*", enviadas de um roteador interno para um roteador externo.

```
12:00:12.366670 int_router > ext_router: icmp: host int_host_90 unreachable
12:00:12.367009 int_router > ext_router: icmp: host int_host_90 unreachable
12:00:22.369117 int_router > ext_router: icmp: host int_host_90 unreachable
...
12:45:48.941137 int_router > ext_router: icmp: host int_host_90 unreachable
12:45:48.941439 int_router > ext_router: icmp: host int_host_90 unreachable
12:45:48.941738 int_router > ext_router: icmp: host int_host_90 unreachable
```

Após a execução do *RECON* com a opção "**-E**", sobre o tráfego que contém os pacotes apresentados, o seguinte resultado foi gerado.

```
[ ext_router (o) <=> int_router (i) ]
  ICMP Error packets
    pkt 1 (<-) 12:00:12.366670 unreachable - host (match icmp) [len 56 id 38786 ttl 64]
    pkt 2 (<-) 12:00:12.367009 unreachable - host (match icmp) [len 56 id 38787 ttl 64]
    pkt 3 (<-) 12:00:22.368789 unreachable - host (match icmp) [len 56 id 38788 ttl 64]
    ...
    pkt 164 (<-) 12:45:48.941137 unreachable - host (match icmp) [len 56 id 43759 ttl 64]
    pkt 165 (<-) 12:45:48.941439 unreachable - host (match icmp) [len 56 id 43760 ttl 64]
    pkt 166 (<-) 12:45:48.941738 unreachable - host (match icmp) [len 56 id 43761 ttl 64]
    --------------------
    duration .............: 12:00:12.366670 - 12:45:48.941738 - diff = 45m 36s 575068ms , 166 packets.
    unreachable .........: type 3 - 166 packets, 0 nomatch
```

A opção "**-E**" faz com que o *RECON* apresente uma saída detalhada para os pacotes de erro. Os identificadores "**(o)**" e "**(i)**", que aparecem logo após os nomes[1] associados aos endereços IP, mostram que o primeiro é externo e o segundo é interno à rede monitorada. Esta identificação é realizada através do uso do endereço e máscara de rede, informados no arquivo de configuração do *RECON* (exemplo no apêndice B.2). São apresentados os pacotes de erro associados aos dois IPs em questão, onde é identificado o tipo (***unreachable***) e o código (***host***) da mensagem ICMP de erro. O identificador ***match icmp*** indica que o pacote de erro foi associado ao pacote gerador do erro, que no caso é ICMP. Ao final, é apresentado um resumo com a duração desde o recebimento do primeiro pacote até o último, e com a totalização dos pacotes para o tipo de mensagem de erro. O valor "**0**" associado ao identificador "***nomatch***" diz que todos os pacotes de erro foram associados aos pacotes geradores do erro.

Já para o próximo exemplo, as mensagens ICMP de erro associadas ao dois IPs em questão não puderam ser associadas aos pacotes geradores dos erros. Este exemplo ilustra também a ocorrência de outro tipo de mensagem ICMP de erro: ***time exceeded***.

[1] À partir daqui estes nomes serão chamados de endereços IP.

```
[ ext_host (o) <=> int_router (i) ]
  ICMP Error packets
    pkt 1 (<-) 04:20:08.192937 time exceeded - time exceeded in-transit (no match udp) \
                               [len 56 id 33050 ttl 64]
    pkt 2 (<-) 04:20:12.830695 time exceeded - time exceeded in-transit (no match udp) \
                               [len 56 id 33052 ttl 64]
    pkt 3 (<-) 04:20:13.124671 time exceeded - time exceeded in-transit (no match udp) \
                               [len 56 id 33053 ttl 64]
    pkt 4 (<-) 04:20:20.201846 unreachable - host (no match udp) [len 56 id 33054 ttl 64]
    pkt 5 (<-) 04:20:20.202369 unreachable - host (no match udp) [len 56 id 33055 ttl 64]
    pkt 6 (<-) 04:20:27.203321 unreachable - host (no match udp) [len 56 id 33092 ttl 64]
    pkt 7 (<-) 04:20:27.203839 unreachable - host (no match udp) [len 56 id 33093 ttl 64]
    pkt 8 (<-) 04:20:34.204790 unreachable - host (no match udp) [len 56 id 33094 ttl 64]
    pkt 9 (<-) 04:20:34.205316 unreachable - host (no match udp) [len 56 id 33095 ttl 64]
    -------------------
    duration ............: 04:20:08.192937 - 04:20:34.205316 - diff = 26s 012379ms , 9 packets.
    unreachable .........: type 3 - 6 packets, 6 nomatch
    time exceeded .......: type 11 - 3 packets, 3 nomatch
```

6.4.2 Sessões ICMP de Informação

A saída *tcpdump* abaixo apresenta pacotes ICMP de informação do tipo ***echo***, trocados entre um *host* interno e um *host* externo.

```
04:41:54.737443 ext_host > int_host: icmp: echo request seq 0 (ttl 50, id 44206, len 64)
04:41:54.739418 int_host > ext_host: icmp: echo reply seq 0 (ttl 253, id 28110, len 64)
04:41:54.884732 ext_host > int_host: icmp: echo request seq 256 (ttl 50, id 44212, len 64)
04:41:54.885131 int_host > ext_host: icmp: echo reply seq 256 (ttl 253, id 28111, len 64)
04:41:55.030296 ext_host > int_host: icmp: echo request seq 512 (ttl 50, id 44219, len 64)
04:41:55.030752 int_host > ext_host: icmp: echo reply seq 512 (ttl 253, id 28112, len 64)
```

Após a execução do *RECON* com a opção "**-I**", sobre o tráfego que contém os pacotes apresentados, o seguinte resultado foi gerado.

```
[ ext_host (o) <=> int_host (i) ]
  ICMP Info sessions
    session 1 [ echo ] id 31346
      (->) 04:41:54.737443  request seq 0 [len 64 id 44206 ttl 50]
      (<-) 04:41:54.739418  reply seq 0 [len 64 id 28110 ttl 253]
      (->) 04:41:54.884732  request seq 256 [len 64 id 44212 ttl 50]
      (<-) 04:41:54.885131  reply seq 256 [len 64 id 28111 ttl 253]
      (->) 04:41:55.030296  request seq 512 [len 64 id 44219 ttl 50]
      (<-) 04:41:55.030752  reply seq 512 [len 64 id 28112 ttl 253]
      ----------------
      duration ......: 04:41:54.737443 - 04:41:55.030752 - diff = 293309ms
      attributes ....: none
      match_reply ...: 0
      payload rcvd ..: (ext_host) 120 bytes , (int_host) 120 bytes , biggest packet (36) , 6 packets.
```

O exemplo mostra que a primeira sessão ICMP de informação, associada aos dois IPs em questão, é do tipo ***echo*** e foi identificada pelo *id* 31346, observado nos pacotes ICMP

associados. Os pacotes são apresentados e diferenciados entre requisições e respostas. Ao final, é apresentado um resumo com a duração da sessão, atributos associados, diferença entre requisições e respostas (***match_reply***) e totalização dos pacotes e dados trocados entre os dois IPs.

Já o próximo exemplo apresenta uma sessão ICMP de informação, onde as respostas para os pacotes de requisição não foram observadas. No fim das linhas que apresentam as informações dos pacotes, foram adicionados os identificadores ***missed*** e ***unreachable***, que informam que pacotes ICMP de erro foram associados aos pacotes originais. No resumo da sessão é apresentado o número total de pacotes associados a pacotes ICMP de erro.

```
[ ext_host (o) <=> int_host (i) ]
  ICMP Info sessions
    session 1 [ echo ] id 3909
      (->) 04:44:27.475093  request seq 20562 [len 40 id 48919 ttl 1]  (missed)
      (->) 04:44:31.604885  request seq 14425 [len 40 id 22399 ttl 2]  (missed)
      (->) 04:44:35.549194  request seq 38495 [len 40 id 55774 ttl 3]  (unreach)
      (->) 04:44:41.603269  request seq 41576 [len 40 id 36198 ttl 3]  (unreach)
      ----------------
      duration ......: 04:44:27.475093 - 04:44:41.603269 - diff = 14s 128176ms
      attributes ....: none
      match_reply ...: 4
      icmp error ....: 4 pkts
      payload rcvd ..: (ext_host) 0 bytes , (int_host) 0 bytes , biggest packet (12) , 4 packets.
```

6.5 DESFRAGMENTAÇÃO IP

A saída *tcpdump* abaixo apresenta uma sequência de pacotes ICMP fragmentados, que constituem uma comunicação entre dois *hosts*. Pode-se observar que o último pacote apresentado é um ICMP de erro, do tipo "*ip reassembly time exceeded*", informando que o último datagrama IP não pôde ser remontado.

```
16:52:20.920219 < host-a > host-b: (frag 9383:48@2960)
16:52:20.920345 < host-a > host-b: (frag 9383:1480@1480+)
16:52:20.920464 < host-a > host-b: icmp: echo request (frag 9383:1480@0+)
16:52:24.248155 < host-b > host-a: icmp: echo reply (frag 25558:1480@0+)
16:52:24.249390 < host-b > host-a: (frag 25558:1480@1480+)
16:52:24.249469 < host-b > host-a: (frag 25558:48@2960)
16:52:31.290616 < host-a > host-b: (frag 9384:48@2960)
16:52:31.290741 < host-a > host-b: (frag 9384:1480@1480+)
16:52:31.290839 < host-a > host-b: icmp: echo request (frag 9384:1480@0+)
16:52:35.827423 < host-b > host-a: icmp: echo reply (frag 25942:1480@0+)
16:52:35.828658 < host-b > host-a: (frag 25942:1480@1480+)
16:52:35.828731 < host-b > host-a: (frag 25942:48@2960)
16:52:39.679566 < host-a > host-b: (frag 9385:48@2960)
16:52:39.679691 < host-a > host-b: (frag 9385:1480@1480+)
16:52:39.679796 < host-a > host-b: icmp: echo request (frag 9385:1480@0+)
16:53:41.372333 < host-b > host-a: icmp: ip reassembly time exceeded
```

Quando o *RECON* é executado com a opção "**-F**", é apresentada a tabela de fragmentos, a partir da árvore binária de desfragmentação implementada pela ferramenta, gerando assim o seguinte resultado:

```
================================================================
| Fragment Table |
+----------------+
[ host-a > host-b ]
  defrag. 1 - 16:52:20.920219 - frags 3  holes 0  ICMP  id 9383  len 3028  state ( reassembled )
    frag. 1 -> [offset = 0] [len = 1480]
    frag. 2 -> [offset = 1480] [len = 1480]
    frag. 3 -> [offset = 2960] [len = 48]
  defrag. 2 - 16:52:31.290616 - frags 3  holes 0  ICMP  id 9384  len 3028  state ( reassembled )
    frag. 1 -> [offset = 0] [len = 1480]
    frag. 2 -> [offset = 1480] [len = 1480]
    frag. 3 -> [offset = 2960] [len = 48]
  defrag. 3 - 16:52:39.679566 - frags 3  holes 0  ICMP  id 9385  len 3028  state ( time_exceeded )
    frag. 1 -> [offset = 0] [len = 1480]
    frag. 2 -> [offset = 1480] [len = 1480]
    frag. 3 -> [offset = 2960] [len = 48]
[ host-b > host-a ]
  defrag. 1 - 16:52:24.248155 - frags 3  holes 0  ICMP  id 25558  len 3028  state ( reassembled )
    frag. 1 -> [offset = 0] [len = 1480]
    frag. 2 -> [offset = 1480] [len = 1480]
    frag. 3 -> [offset = 2960] [len = 48]
  defrag. 2 - 16:52:35.827423 - frags 3  holes 0  ICMP  id 25942  len 3028  state ( reassembled )
    frag. 1 -> [offset = 0] [len = 1480]
    frag. 2 -> [offset = 1480] [len = 1480]
    frag. 3 -> [offset = 2960] [len = 48]
```

Os fragmentos foram agrupados de acordo com a direção em que foram enviados. Então, tem-se um conjunto de fragmentos que vão do ***host-a*** para o ***host-b***, e outro do ***host-b*** para o ***host-a***. Dentro de cada um destes grupos, os fragmentos são organizados em elementos de desfragmentação, identificados pelo *id*, que é comum a cada conjunto de fragmentos relacionados. Então, para cada elemento de desfragmentação é informado o horário do primeiro fragmento observado, o número de fragmentos, se há espaços entre os fragmentos, o protocolo utilizado, o tamanho total do datagrama e o estado da desfragmentação. Nos casos onde o estado é ***reassembled***, o fragmento foi completamente remontado e foi inserido em uma sessão válida.

No caso em que o estado da desfragmentação é ***time exceeded***, uma situação interessante ocorreu. Os três fragmentos que compõem o elemento de desfragmentação em questão ocorreram antes da mensagem de erro associada, como observado na saída *tcpdump*. Portanto, antes da mensagem de erro ser processada, o datagrama IP foi dado como remontado e inserido em uma sessão válida. Após a análise da mensagem de erro, realizada pelo processador ICMP de erro, o estado do elemento de desfragmentação foi atualizado para ***time exceeded***, o pacote remontado foi removido da sessão onde foi inserido, e ajustes necessários na sessão em questão foram realizados.

O resultado destas operações é ilustrado nas estatísticas apresentadas abaixo, geradas na análise do tráfego em questão.

```
================================================================
| ICMP Numbers |
+--------------+
...
ICMP messages - time exceeded in frag.                          1
-------------
ICMP packets inserted in the model                              4
- Info messages
  Requisition ICMP info packets                                 2
  Reply ICMP info packets                                       2
- Error messages
  ...
  ICMP time exceeded in frag. (no match)                        0
================================================================
```

Um problema conhecido do processo de desfragmentação IP está associado à fragmentação do cabeçalho da camada de transporte. O processador IP, ao observar um pacote desta natureza, irá descartá-lo, eliminando assim a possibilidade deste ser remontado. Apesar de algumas implementações da pilha TCP/IP não permitirem este tipo de pacote, isto não é regra e, portanto, aperfeiçoamentos na implementação do *RECON* serão necessários. Esta proposta é apresentada no capítulo 7.

6.6 RECONSTRUÇÃO DAS SESSÕES UDP

A saída *tcpdump* abaixo apresenta uma sequência de pacotes UDP, associados ao serviço SNMP (seção 2.7), trocados entre um *host* interno e um roteador externo. Pode-se observar que os pacotes são identificados como requisições ou respostas.

```
04:00:04.246473 int_snmp_mon.56682 > border_router.161: { SNMPv1 { GetRequest(11) R=1850734177 \
                                                          [|snmp]} } (DF) (ttl 251, id 40245, len 133)
04:00:04.363584 border_router.161 > int_snmp_mon.56682: { SNMPv1 { GetResponse(9) R=1850734177 \
                                                          [|snmp]} } (ttl 252, id 15830, len 173)
...
04:55:04.657942 int_snmp_mon.60154 > border_router.161: { SNMPv1 { GetRequest(11) R=95081423 \
                                                          [|snmp]} } (DF) (ttl 251, id 63825, len 133)
04:55:04.747432 border_router.161 > int_snmp_mon.60154: { SNMPv1 { GetResponse(9) R=95081423 \
                                                          [|snmp]} } (ttl 252, id 15841, len 173)
```

Após a execução do *RECON* com a opção "**-U**", sobre o tráfego que contém os pacotes apresentados, o seguinte resultado foi gerado:

```
[ border_router (o) <=> int_snmp_mon (i) ]
 UDP sessions
  session 1 [ 161 <=> 56682 ]
   (<-) 04:00:04.246473 udp 113 (105) [ip - hd_len 20 len 133 id 40245 ttl 251]
   (->) 04:00:04.363584 udp 153 (145) [ip - hd_len 20 len 173 id 15830 ttl 252]
   ---------------
   duration .....: 04:00:04.246473 - 04:00:04.363584 - diff = 117111ms
   payload rcvd .: (border_router) 105 bytes, (int_snmp_mon) 145 bytes, biggest packet (145), 2 packets.
   ...
  session 12 [ 161 <=> 60154 ]
   (<-) 04:55:04.657942 udp 113 (105) [ip - hd_len 20 len 133 id 63825 ttl 251]
   (->) 04:55:04.747432 udp 153 (145) [ip - hd_len 20 len 173 id 15841 ttl 252]
   ---------------
   duration .....: 04:55:04.657942 - 04:55:04.747432 - diff = 089490ms
   payload rcvd .: (border_router) 105 bytes, (int_snmp_mon) 145 bytes, biggest packet (145), 2 packets.
```

Os pacotes associados aos dois IPs em questão foram agrupados em sessões, identificadas pelo par de portas comum a cada pacote UDP. Para cada pacote é informado o tamanho total da mensagem UDP, incluindo seu cabeçalho e, entre parênteses, a quantidade de bytes que compõe seu conteúdo. Ao final de cada sessão é apresentado um resumo com a duração da sessão e a totalização dos pacotes e dados trocados entre os dois IPs.

Deve-se observar que não é feita distinção entre pacotes de requisição e resposta. Estas informações são encontradas no conteúdo da mensagem UDP e, com exceção das aplicações de DNS e NTP, não são tratadas pela ferramenta. Todas as sessões UDP associadas à outras portas de serviços, mas análogas à apresentada anteriormente, são processadas da mesma forma.

Já as sessões UDP, associadas aos serviços de DNS e NTP, recebem um tratamento diferenciado. A saída *tcpdump* abaixo apresenta uma sequência de pacotes trocados entre um *host* interno e um servidor de DNS externo.

```
12:10:37.263680 int_host.1365 > ext_dns_serv.53: 37416 [1au][|domain] (ttl 63, id 21374, len 74)
12:10:37.263709 int_host.1365 > ext_dns_serv.53: 58597 [1au][|domain] (ttl 63, id 21375, len 76)
12:10:37.330077 ext_dns_serv.53 > int_host.1365: 37416-[|domain] (ttl 55, id 36215, len 149)
12:10:37.338636 ext_dns_serv.53 > int_host.1365: 58597-[|domain] (ttl 55, id 36216, len 155)
```

Após a execução do *RECON*, sobre o tráfego que contém os pacotes apresentados, o seguinte resultado foi gerado:

```
[ int_host (i) <=> ext_dns_serv (o) ]
 UDP sessions
  session 1 [ 1365 <=> 53 ]
   (->) 12:10:37.263680 udp 54 (46) [dns request - id 37416] [ip - hd_len 20 len 74 id 21374 ttl 63]
   (->) 12:10:37.263709 udp 56 (48) [dns request - id 58597] [ip - hd_len 20 len 76 id 21375 ttl 63]
   (<-) 12:10:37.330077 udp 129 (121) [dns reply - id 37416] [ip - hd_len 20 len 149 id 36215 ttl 55]
   (<-) 12:10:37.338636 udp 135 (127) [dns reply - id 58597] [ip - hd_len 20 len 155 id 36216 ttl 55]
   ---------------
   duration .....: 12:10:37.263680 - 12:10:37.338636 - diff = 074956ms
   attributes ...: none
   match_reply ..: 0
   payload rcvd .: (int_host) 248 bytes , (ext_dns_serv) 94 bytes , biggest packet (127) , 4 packets.
```

Os pacotes UDP associados aos dois IPs foram agrupados em uma sessão, identificada pelo par de portas comum a cada pacote. A diferença aqui vem do conhecimento específico, associado à forma de comunicação da aplicação de DNS. Sabe-se que os dois primeiros bytes do conteúdo do pacote são utilizados para diferenciar entre requisição e resposta e para identificar e associar cada requisição a sua respectiva resposta.

Portanto, cada pacote da sessão apresenta o *id* da mensagem UDP de DNS e informa se esta é uma requisição ou resposta. Ao final, é apresentado um resumo da sessão, com sua duração, atributos associados, diferença entre pacotes de requisição e resposta (***match_reply***) e a totalização dos pacotes e dados trocados entre os dois IPs.

Para o caso das sessões UDP associadas ao serviço de NTP, o tratamento é análogo. A saída *tcpdump* abaixo apresenta uma sequência de pacotes trocados entre dois *hosts*, utilizando o serviço em questão.

```
01:10:02.421237 < int_ntp_p1.123 > int_ntp_p2.123: v4 client strat 4 (ttl 64, id 22803)
01:10:02.421532 < int_ntp_p2.123 > int_ntp_p1.123: v4 server strat 3 (ttl 64, id 3484)
...
01:16:07.810014 < int_ntp_p2.123 > int_ntp_p1.123: v3 sym_act strat 3 (ttl 64, id 19352)
01:16:07.815641 < int_ntp_p1.123 > int_ntp_p2.123: v3 sym_pas strat 4 (ttl 64, id 33990)
...
```

Após a execução do *RECON*, sobre o tráfego que contém os pacotes apresentados, o seguinte resultado foi gerado:

```
[ int_ntp_p1 (i) <=> int_ntp_p2 (i) ]
 UDP sessions
  session 1 [ 123 <=> 123 ]
   (->) 01:10:02.421237 udp 56 (48) [ntp - v4 client] [ip - hd_len 20 len 76 id 22803 ttl 64]
   (<-) 01:10:02.421532 udp 56 (48) [ntp - v4 server] [ip - hd_len 20 len 76 id 3484 ttl 64]
   (<-) 01:16:07.810014 udp 56 (48) [ntp - v3 symmetric active] [ip - hd_len 20 len 76 id 19352 ttl 64]
   (->) 01:16:07.815641 udp 56 (48) [ntp - v3 symmetric passive] [ip - hd_len 20 len 76 id 33990 ttl 64]
   ...
   ---------------
   duration .....: 01:10:02.421237 - 14:55:19.817520 - diff = 13h 45m 17s 396283ms
   attributes ...: none
   match_reply ..: 0
   payload rcvd .: (int_ntp_p1) 4704 bytes, (int_ntp_p2) 4704 bytes, biggest packet (48), 196 packets.
```

Os pacotes UDP associados aos dois IPs foram agrupados em uma sessão, identificada pelo par de portas comum a cada pacote. Sabe-se que o primeiro byte do conteúdo do pacote é utilizado para identificar o modo de operação do pacote e para diferenciar entre requisição e resposta. Portanto, cada pacote da sessão apresenta o modo de operação da mensagem UDP de NTP. Os modos ***client*** e ***symmetric active*** estão associados à requisições, e os modos ***server*** e ***symmetric passive*** à respostas. Ao final, é apresentado um resumo da sessão, com sua duração, atributos associados, diferença entre pacotes de requisição e resposta (***match_reply***) e a totalização dos pacotes e dados trocados entre os dois IPs.

6.7 RECONSTRUÇÃO DAS SESSÕES TCP

A saída *tcpdump* abaixo apresenta uma sequência de pacotes típica em uma conexão bidirecional TCP, trocados entre um *host* externo e um servidor interno.

```
04:28:00.942144 ext_host.61183 > int_smtp_serv.25: S 365735391:365735391(0) win 24820 \
                                                   (DF) (ttl 48, id 52574, len 48)
04:28:00.943500 int_smtp_serv.25 > ext_host.61183: S 1881033246:1881033246(0) ack 365735392 win 16560
                                                   (DF) (ttl 62, id 4311, len 44)
04:28:01.135432 ext_host.61183 > int_smtp_serv.25: . ack 1881033247 win 24840 \
                                                   (DF) (ttl 48, id 52575, len 40)
04:28:01.489202 int_smtp_serv.25 > ext_host.61183: P 1881033247:1881033278(31) ack 365735392 win 16560 \
                                                   (DF) (ttl 62, id 12319, len 71)
04:28:01.680527 ext_host.61183 > int_smtp_serv.25: . ack 1881033278 win 24840 \
                                                   (DF) (ttl 48, id 52578, len 40)
...
04:28:02.624829 int_smtp_serv.25 > ext_host.61183: P 1881033415:1881033453(38) ack 365742172 win 16560 \
                                                   (DF) (ttl 62, id 12673, len 78)
04:28:02.625069 int_smtp_serv.25 > ext_host.61183: F 1881033453:1881033453(0) ack 365742172 win 16560 \
                                                   (DF) (ttl 62, id 17737, len 40)
04:28:02.817370 ext_host.61183 > int_smtp_serv.25: . 1881033454 win 24840 \
                                                   (DF) (ttl 48, id 52601, len 40)
04:28:02.817399 ext_host.61183 > int_smtp_serv.25: F 365742172:365742172(0) ack 1881033453 win 24840 \
                                                   (DF) (ttl 48, id 52600, len 40)
04:28:02.818360 int_smtp_serv.25 > ext_host.61183: . ack 365742173 win 16560 \
                                                   (DF) (ttl 62, id 23176, len 40)
```

Após a execução do *RECON* com a opção "**-T**", sobre o tráfego que contém os pacotes apresentados, o seguinte resultado foi gerado:

```
[ ext_host (o) <=> int_smtp_server (i) ]
  TCP sessions
    session 1 [ 61183 (c) <=> 25 (s) ]
      (->) 04:28:00.942144 S 365735391:365735391(0) ack 0 win 24820 [len 48 id 52574 ttl 48]
      (<-) 04:28:00.943500 SA 1881033246:1881033246(0) ack 365735392 win 16560 [len 44 id 4311 ttl 62]
      (->) 04:28:01.135432 A 365735392:365735392(0) ack 1881033247 win 24840 [len 40 id 52575 ttl 48]
      (<-) 04:28:01.489202 AP 1881033247:1881033278(31) ack 365735392 win 16560 [len 71 id 12319 ttl 62]
      (->) 04:28:01.680527 A 365735392:365735392(0) ack 1881033278 win 24840 [len 40 id 52578 ttl 48]
      ...
      (<-) 04:28:02.624829 AP 1881033415:1881033453(38) ack 365742172 win 16560 [len 78 id 12673 ttl 62]
      (<-) 04:28:02.625069 AF 1881033453:1881033453(0) ack 365742172 win 16560 [len 40 id 17737 ttl 62]
      (->) 04:28:02.817370 A 365742173:365742173(0) ack 1881033454 win 24840 [len 40 id 52601 ttl 48]
      (->) 04:28:02.817399 AF 365742172:365742172(0) ack 1881033453 win 24840 [len 40 id 52600 ttl 48]
      (<-) 04:28:02.818360 A 1881033454:1881033454(0) ack 365742173 win 16560 [len 40 id 23176 ttl 62]
      ----------------
      state .........: Fin2 Closed Ok
      duration ......: 04:28:00.942144 - 04:28:02.818360 - diff = 01s 876216ms
      attributes ....: none
      payload rcvd ..: (ext_host) 206 bytes, (int_smtp_server) 6780 bytes , \
                       biggest packet (1380) , 24 packets.
```

Os pacotes TCP associados aos dois IPs foram agrupados em uma sessão, identificada pelo par de portas comum a cada pacote. Os identificadores "**(c)**" e "**(s)**", que aparecem logo após as portas, mostram que primeiro par <IP,porta> é o cliente na sessão, e que o segundo par corresponde ao servidor. Foi possível realizar esta identificação, pois o *three-way handshake* está presente no início da sessão. São apresentados, para cada pacote associado à sessão, os *flags* ativos, *sequence number*, *acknowledge number* e o *window size*. Ao final, é apresentado um resumo da sessão, seu estado final, duração, atributos associados e a totalização dos pacotes e dados trocados entre os dois IPs.

Para este exemplo, o estado final da sessão indica o término normal da sessão, caracterizado pela sequência de pacotes FIN1, ACK, FIN2, ACK. Deve-se observar que a retransmissão dos pacotes FIN não influenciou no estado final determinado.

Os próximos exemplos caracterizam outras situações relevantes, e apresentam sessões com diferentes estados terminais.

O quadro abaixo apresenta uma sessão TCP, onde seu estado final determina o término simultâneo da sessão, caracterizado pela sequência de pacotes FIN1, FIN2, ACK, ACK. O nome ***Simultaneous Fin1*** dado ao estado deve-se ao fato do último ACK da sessão estar associado ao primeiro FIN.

```
[ ext_host (o) <=> int_smtp_serv (i) ]
  TCP sessions
    session 1 [ 14323 (c) <=> 25 (s) ]
      (->) 04:42:16.196996 S 1687056646:1687056646(0) ack 0 win 16384 [len 60 id 29103 ttl 47]
      (<-) 04:42:16.198029 SA 4294966272:4294966272(0) ack 1687056647 win 33580 [len 48 id 36793 ttl 58]
      (->) 04:42:16.384472 A 1687056647:1687056647(0) ack 4294966273 win 17520 [len 40 id 29813 ttl 47]
      (<-) 04:42:22.214794 AP 4294966273:4294966357(84) ack 1687056647 win 33580 [len 124 id 36810 ...]
      (->) 04:42:22.401051 A 1687056647:1687056647(0) ack 4294966357 win 17436 [len 40 id 48594 ttl 47]
      ...
      (<-) 04:42:28.928780 AP 4294966771:4294966813(42) ack 1687068518 win 33580 [len 82 id 36826 ...]
      (<-) 04:42:28.929985 AF 4294966813:4294966813(0) ack 1687068518 win 33580 [len 40 id 36827 ttl 58]
      (->) 04:42:29.114862 A 1687068518:1687068518(0) ack 4294966813 win 17478 [len 40 id 3502 ttl 47]
      (->) 04:42:29.116175 AF 1687068518:1687068518(0) ack 4294966813 win 17520 [len 40 id 3507 ttl 47]
      (->) 04:42:29.116305 AF 1687068518:1687068518(0) ack 4294966814 win 17520 [len 40 id 3510 ttl 47]
      (<-) 04:42:29.116901 AF 4294966813:4294966813(0) ack 1687068519 win 33580 [len 40 id 36828 ttl 58]
      (<-) 04:42:29.117162 A 4294966814:4294966814(0) ack 1687068519 win 33580 [len 40 id 36829 ttl 58]
      (->) 04:42:29.302867 A 1687068519:1687068519(0) ack 4294966814 win 17520 [len 40 id 4156 ttl 47]
      ----------------
      state .........: Simultaneous Fin1 Closed Ok
      duration ......: 04:42:16.196996 - 04:42:29.302867 - diff = 13s 105871ms
      attributes ....: none
      payload rcvd ..: (ext_host) 540 bytes, (int_smtp_serv) 11871 bytes , \
                       biggest packet (1460) , 39 packets.
```

No quadro seguinte, o estado final determina também o término simultâneo da sessão TCP, mas o nome ***Simultaneous Fin2*** dado ao estado deve-se ao fato do último ACK da sessão estar associado ao segundo FIN. Uma característica desta sessão é que o ACK para o primeiro FIN faz parte do pacote que contém o segundo FIN. E, portanto, não influenciou no estado final da sessão.

Também pode-se observar a ocorrência do atributo ***Cold_Start*** e a não determinação do cliente e servidor da sessão, devido a ausência de todo o *three-way handshake*. Esta situação ocorre quando o sistema de captura falha na coleta de pacotes ou mesmo quando estes estão contidos no arquivo da hora anterior.

```
[ ext_ftp_serv (o) <=> int_host (i) ]
  TCP sessions
    session 1 [ 21 <=> 42582 ]
      (<-) 04:00:03.112594 A 235869536:235869536(0) ack 234991991 win 5840 [len 52 id 55350 ttl 62]
      (<-) 04:00:03.225639 AP 235869536:235869555(19) ack 234991991 win 5840 [len 71 id 55351 ttl 62]
      (->) 04:00:03.374717 AP 234991991:234992066(75) ack 235869555 win 32120 [len 127 id 20899 ttl 48]
      ...
      (->) 04:00:32.444908 AP 234992090:234992104(14) ack 235869561 win 32120 [len 66 id 26418 ttl 48]
      (->) 04:00:32.445378 AF 234992104:234992104(0) ack 235869561 win 32120 [len 52 id 26419 ttl 48]
      (<-) 04:00:32.446454 A 235869561:235869561(0) ack 234992104 win 5840 [len 52 id 55355 ttl 62]
      (<-) 04:00:32.446516 AF 235869561:235869561(0) ack 234992105 win 5840 [len 52 id 55356 ttl 62]
      (->) 04:00:32.663171 A 234992105:234992105(0) ack 235869562 win 32120 [len 52 id 26422 ttl 48]
      ----------------
      state .........: Simultaneous Fin2 Closed Ok
      duration ......: 04:00:03.112594 - 04:00:32.663171 - diff = 29s 550577ms
      attributes ....: Cold_Start
      payload rcvd ..: (ext_ftp_serv) 25 bytes, (int_host) 113 bytes , biggest packet (75) , 12 packets.
```

Na saída mostrada abaixo, a sessão TCP foi terminada normalmente, mas o atributo ***Cold_Start*** está presente. O cliente e o servidor da sessão puderam ser determinados pois, apesar do primeiro pacote do *three-way handshake* ter sido perdido, o pacote SYN/ACK (SA) pôde ser observado.

```
[ int_host (i) <=> ext_web_serv (o) ]
  TCP sessions
    session 1 [ 3369 (c) <=> 8080 (s) ]
      (<-) 12:00:05.134030 SA 13545257:13545257(0) ack 3455609422 win 1460 [len 48 id 47002 ttl 117]
      (->) 12:00:05.136451 A 3455609422:3455609422(0) ack 13545258 win 17520 [len 40 id 44737 ttl 127]
      (->) 12:00:05.137337 AP 3455609422:3455609725(303) ack 13545258 win 17520 [len 343 id 44738 ...]
      (<-) 12:00:05.352495 AP 13545258:13545378(120) ack 3455609725 win 8457 [len 160 id 47770 ttl 117]
      (<-) 12:00:05.352605 AF 13545378:13545378(0) ack 3455609725 win 8457 [len 40 id 48026 ttl 117]
      (->) 12:00:05.353264 A 3455609725:3455609725(0) ack 13545379 win 17400 [len 40 id 44739 ttl 127]
      (->) 12:00:05.355022 AF 3455609725:3455609725(0) ack 13545379 win 17400 [len 40 id 44740 ttl 127]
      (<-) 12:00:05.582214 A 13545379:13545379(0) ack 3455609726 win 8457 [len 40 id 48282 ttl 117]
      ----------------
      state .........: Fin2 Closed Ok
      duration ......: 12:00:05.134030 - 12:00:05.582214 - diff = 448184ms
      attributes ....: Cold_Start
      payload rcvd ..: (int_host) 120 bytes, (ext_web_serv) 303 bytes , biggest packet (303) , 8 packets.
```

Já para a situação ilustrada abaixo, o estado final determina o término abrupto da sessão, caracterizado pela ocorrência do pacote RST. Deve-se observar que a retransmissão do pacote RST foi associada a esta mesma sessão.

```
[ ext_ftp_serv (o) <=> int_host (i) ]
  TCP sessions
    session 1 [ 21 <=> 1178 ]
      (<-) 12:00:30.067065 AF 970217047:970217047(0) ack 2811158358 win 15372 [len 40 id 4601 ttl 127]
      (->) 12:00:30.527401 A 2811158358:2811158358(0) ack 970217048 win 24840 [len 40 id 29242 ttl 46]
      (->) 12:00:30.527424 AP 2811158358:2811158395(37) ack 970217048 win 24840 [len 77 id 29243 ttl 46]
      (->) 12:00:30.527453 AF 2811158395:2811158395(0) ack 970217048 win 24840 [len 40 id 29244 ttl 46]
      (<-) 12:00:30.542158 R 970217048:970217048(0) ack 4100948994 win 0 [len 40 id 4603 ttl 127]
      (<-) 12:00:30.542225 R 970217048:970217048(0) ack 970217048 win 0 [len 40 id 4604 ttl 127]
      ----------------
      state .........: Reset Closed Ok
      duration ......: 12:00:30.067065 - 12:00:30.542225 - diff = 475160ms
      attributes ....: Cold_Start
      payload rcvd ..: (ext_ftp_serv) 0 bytes, (int_host) 37 bytes , biggest packet (37) , 6 packets.
```

E finalmente, no quadro abaixo, o estado final da sessão mostra que esta não foi terminada, provavelmente porque os pacotes relativos a seu término foram armazenados em algum arquivo subsequente.

```
[ ext_ftp_serv (o) <=> int_host (i) ]
  TCP sessions
    session 1 [ 21 (s) <=> 1651 (c) ]
      (<-) 12:45:43.451144 S 3663629258:3663629258(0) ack 0 win 16384 [len 48 id 10124 ttl 125]
      (->) 12:45:43.602494 SA 2585636504:2585636504(0) ack 3663629259 win 5840 [len 48 id 0 ttl 46]
      (<-) 12:45:43.603723 A 3663629259:3663629259(0) ack 2585636505 win 17520 [len 40 id 10125 ttl 125]
      (->) 12:45:43.752914 AP 2585636505:2585636547(42) ack 3663629259 win 5840 [len 82 id 37372 ttl 46]
      (<-) 12:45:43.757569 AP 3663629259:3663629274(15) ack 2585636547 win 17478 [len 55 id 10126 ...]
      ...
      (->) 12:57:02.805034 AP 2585641646:2585641662(16) ack 3663631084 win 5840 [len 56 id 37584 ttl 46]
      (<-) 12:57:02.805914 A 3663631084:3663631084(0) ack 2585641662 win 16785 [len 40 id 12390 ttl 125]
      ----------------
      state .........: Open
      duration ......: 12:45:43.451144 - 12:57:02.805914 - diff = 11m 19s 354770ms
      attributes ....: none
      payload rcvd ..: (ext_ftp_serv) 1825 bytes, (int_host) 5157 bytes , \
                       biggest packet (87) , 374 packets.
```

6.8 ATIVIDADES RELACIONADAS À DETECÇÃO DE INTRUSÃO

Esta seção apresenta os resultados obtidos, através da execução do *RECON*, na sua aplicação para a detecção de intrusão. Foram utilizadas informações geradas pelas árvores binárias de *logs* TCP e de desfragmentação, e por algumas rotinas implementadas e integradas ao *RECON*.

6.8.1 A Árvore de *Logs* TCP

A árvore de *logs* TCP tem como finalidade armazenar pacotes TCP em duas situações: os que não foram associados a alguma sessão e os associados a alguma sessão, mas que apresentaram um comportamento não esperado.

Nesta última situação, onde o pacote de *log* faz parte de uma sessão, são observados os *flags* e o momento em que o pacote em questão foi associado a uma determinada sessão. Por exemplo, um pacote com os *flags* PSH e ACK ativos, contendo dados e associado a uma sessão onde o *three-way handshake* ainda não tenha sido concluído.

E na primeira situação, são observados os pacotes que apresentam uma combinação ou ausência de *flags*, de modo que não devem ser responsáveis pela criação de uma nova sessão. Este tipo de pacote normalmente está associado a varreduras (*scans*), consideradas pela comunidade de segurança como precursoras de ataques ou intrusões. Como visto na seção 5.3.12, as ocorrências relacionadas ao uso destes pacotes são conhecidas como atividades *stealth*.

Atualmente, existe uma grande variedade de ferramentas que geram tais pacotes. Os intrusos normalmente fazem uso destas ferramentas para investigar se um determinado

serviço está ativo em um ou mais *hosts*, e até mesmo para descobrir que sistemas operacionais estão sendo executados nos *hosts* investigados (Fyodor, 2002b). Um bom exemplo é a ferramenta de varredura de rede *nmap* (*network mapper*) desenvolvida por Fyodor (2002a).

A grande maioria dos pacotes TCP, analisados pelo *RECON* durante todo o período em que o tráfego de rede foi examinado e que foram associados à árvore de *logs* TCP, se enquadraram na primeira situação. O *RECON* foi executado com a opção "**-l**", para apresentar os *logs* de forma colapsada, mas devido ao grande número de ocorrências deste tipo, alguns resultados foram compilados em uma única saída, apresentada abaixo, de modo a exemplificar a maioria dos casos observados no período em questão.

```
[ host-a - host-k ]
  09:04:59.554629 (73 -> 1438) : P 5242941:5242953(12) ack 4032219641 win 20496 - \
                                 vecna sth act (no session).
[ host-b - host-n ]
  09:18:23.197408 (21 -> 1078) : SAPRUF 36306988:36306992(4) ack 2193322316 win 20496 - \
                                 fxmas tree sth act (no session).
  09:22:01.512284 (234 -> 1116) : 1PU 5242930:5242942(12) ack 1549244900 win 20484 - \
                                  unknown sth act (no session).
[ host-c - host-p ]
  10:16:22.750380 (52558 -> 53) : PUF 1150218965:1150218965(0) ack 0 win 3072 - \
                                  nmap xmas tree sth act (no session). (repeated 3 times)
[ host-d - host-x ]
  13:14:56.050751 (0 -> 2901) : F 5243167:5243175(8) ack 1733885952 win 20496 - \
                                fin sth act (no session).
[ host-e - host-y ]
  13:52:10.857162 (0 -> 2142) : SAPU 5243017:5243057(40) ack 1768544445 win 20496 - \
                                sin/ack/push/urg sth act (no session).
[ host-a - host-w ]
  15:17:52.517083 (4 -> 1375) : 12SPUF 5242907:5242911(4) ack 1104258650 win 20496 - \
                                unknown sth act (no session).
[ host-c - host-z ]
  16:02:36.896668 (50832 -> 7) : SPUF 2300745433:2300745433(0) ack 0 win 4096 - \
                                 unknown sth act (no session).
  16:02:36.896908 (50836 -> 1) : PUF 2300745433:2300745433(0) ack 0 win 4096 - \
                                 nmap xmas tree sth act (no session). {rst}
  16:02:38.895885 (50831 -> 7) : . 2300745433:2300745433(0) ack 0 win 4096 - \
                                 nxmas tree sth act (no session). (repeated 2 times)
```

No quadro acima, o pacote enviado do ***host-a*** para o ***host-k*** apresenta apenas o *flag* PSH ativo. Este pacote gerou o *log* que caracteriza a atividade *stealth* intitulada ***vecna***. Pacotes com apenas os *flags* URG ou PSH/FIN ou URG/PSH são caracterizados da mesma forma.

No primeiro pacote do ***host-b*** para o ***host-n*** todos os *flags* estão ativos. Este gerou o *log* que caracteriza a atividade *stealth* intitulada ***full christmas tree***.

No pacote enviado do ***host-c*** para o ***host-p*** os *flags* PSH/URG/FIN estão ativos. Este

gerou o *log* que caracteriza a atividade *stealth* intitulada ***nmap christmas tree*** e indica o provável uso da ferramenta *nmap*. Também ilustra a apresentação colapsada de pacotes associados a um mesmo tipo de *log*, onde "***repeated 3 times***" diz que mais três pacotes com as mesmas características foram enviados.

No pacote enviado do ***host-d*** para o ***host-x*** apenas o *flag* FIN está ativo. Este gerou o *log* que caracteriza a atividade *stealth* intitulada ***fin***.

No pacote enviado do ***host-e*** para o ***host-y*** os *flags* SYN/ACK/PSH/URG estão ativos. Este gerou o *log* que caracteriza a atividade *stealth* intitulada ***sapu***.

No segundo pacote do ***host-c*** para o ***host-z***, o "**{*rst*}**" no final da linha diz que um pacote de RST foi enviado como resposta ao pacote que gerou o *log* em questão. E na última linha, o pacote não contém *flags* ativos, o que caracteriza a atividade *stealth* intitulada ***null christmas tree***.

E finalmente, os pacotes que geraram *logs* de eventos que caracterizam atividades *stealth* do tipo ***unknown***, estão associados a combinações de *flags* não conhecidas pelo processador de pacotes TCP.

6.8.2 Alertas da Árvore de Desfragmentação

A execução do *RECON* com a opção "**-f**" faz com que as entradas na árvore de desfragmentação, associadas a algum alerta, sejam apresentadas nos resultados gerados para o tráfego de rede analisado, caso ocorram.

O resultado apresentado abaixo ilustra uma situação, onde uma sequência de fragmentos, ao serem remontados, disparou um alerta indicando a ocorrência de uma sobreposição. Este alerta é gerado através da verificação dos atributos associados à entrada na árvore de desfragmentação e que, para este caso, indica o alerta ***OVERLAP***.

```
================================================================
| Fragment Table |
+----------------+
[ host-a > host-b ]
  defrag. 1 - 17:49:12.283209 - frags 9 holes 0 ICMP id 42949 len 84 state ( reassembled )  \
                                attributes ( OVERLAP )
================================================================
```

Ao executar o *RECON* com a opção **-F**, sobre o mesmo tráfego de rede, esta entrada na árvore de desfragmentação é apresentada de forma mais detalhada, onde pode-se observar a sobreposição do fragmento de *offset* igual a **48**.

```
=================================================================
| Fragment Table |
+----------------+
[ host-a > host-b ]
  defrag. 1 - 17:49:12.283209 - frags 9 holes 0 ICMP id 42949 len 84 state ( reassembled )  \
                                attributes ( OVERLAP )
    frag. 1 -> [offset = 0] [len = 8]
    frag. 2 -> [offset = 8] [len = 8]
    frag. 3 -> [offset = 16] [len = 8]
    frag. 4 -> [offset = 24] [len = 8]
    frag. 5 -> [offset = 32] [len = 8]
    frag. 6 -> [offset = 40] [len = 8]
    frag. 7 -> [offset = 48] [len = 8]
    frag. 8 -> [offset = 48] [len = 8]
    frag. 9 -> [offset = 56] [len = 8]
=================================================================
```

A saída *tcpdump* associada a este caso é apresentada abaixo.

```
17:49:12.283209 host-a > host-b: icmp: echo request (frag 42949:8@0+)
17:49:12.335185 host-a > host-b: (frag 42949:8@8+)
17:49:12.336641 host-a > host-b: (frag 42949:8@16+)
17:49:12.337812 host-a > host-b: (frag 42949:8@24+)
17:49:12.338090 host-a > host-b: (frag 42949:8@32+)
17:49:12.338295 host-a > host-b: (frag 42949:8@40+)
17:49:12.338506 host-a > host-b: (frag 42949:8@48+)
17:49:12.338708 host-a > host-b: (frag 42949:8@48+)
17:49:12.338894 host-a > host-b: (frag 42949:8@56)
```

Segundo Ptacek e Newsham (1998), esta técnica de ataque pode ser utilizada para tentar iludir um sistema de detecção de intrusão, pois este pode remontar o datagrama de uma forma completamente diferente da adotada pelo sistema final, para onde os fragmentos foram direcionados.

O fato é que a fragmentação de um datagrama com sobreposição (*overlaping*) de *offsets* constitui uma atividade maliciosa, que não ocorre na utilização normal de um sistema e, portanto, foi detectada e registrada pelo *RECON*.

6.8.3 Detecção de *Host Scans*

Normalmente, intrusos realizam varreduras (*scans*) para identificar *hosts* ativos em um ou mais domínios, ou mesmo para identificar se um determinado serviço está sendo oferecido nos *hosts* investigados. Estas atividades fazem parte do processo malicioso de obtenção de informações, e têm como principal finalidade direcionar os ataques a serem realizados pelos intrusos.

Apesar dos resultados gerados a partir da árvore de *logs* TCP, vistos na seção 6.8.1, indicarem na maioria dos casos atividades relacionadas a varreduras, estes não estão

associados a sessões estabelecidas. Portanto, faz-se necessário verificar tais atividades nas sessões reconstruídas pelo *RECON*.

Baseado nestas idéias, uma rotina foi implementada e integrada ao *RECON*, no intuito de detectar estas atividades maliciosas e, normalmente, precursoras de ataques.

A rotina utiliza a árvore binária de IPs e as listas de adjacências associadas, geradas após o processamento do tráfego de rede. Para cada endereço IP da árvore é verificado o número de elementos na sua lista de adjacências. Caso este número seja igual ou maior que um limiar, as próximas verificações são realizadas. Para facilitar o entendimento, seja IP_1 o endereço IP identificado pelo nó da árvore, e IP_n $(n \neq 1)$ os endereços IP referenciados na lista de adjacências de IP_1.

Verifica-se se o mesmo tipo de sessão ICMP de informação ocorreu entre cada par $[IP_1, IP_n]$ levando em consideração a direção. Neste caso, as requisições devem partir do IP_1 para cada IP_n. Se a contagem destas ocorrências exceder o limiar pré-estabelecido, uma linha é acrescentada aos resultados, contendo o IP_1, origem do *scan*, o número de destinos acessados e o recurso utilizado, por exemplo, **echo/icmp**.

Também verifica-se se sessões TCP ocorreram entre cada par $[IP_1, IP_n]$ levando em consideração a direção. Neste caso, as sessões devem ter sido iniciadas pelo IP_1 e a porta de destino deve ser igual para cada IP_n. Se a contagem destas ocorrências for igual ou maior que o limiar, uma linha é acrescentada aos resultados, contendo o IP_1, origem do *scan*, o número de destinos acessados e o recurso utilizado, no formato **porta/tcp**.

São utilizados os seguintes parâmetros, que devem estar presentes no arquivo de configuração do *RECON* (ver apêndice B.2):

- *FIND_HOST_SCAN*: este parâmetro diz para o *RECON* que a busca por varreduras de *hosts* deve ser realizada;
- *IN_HOST_SCAN_THRESHOLD* e *OUT_HOST_SCAN_THRESHOLD*: estes dois parâmetros informam limiares a serem verificados, e que reduzem as buscas sobre a árvore binária de IPs. Estão relacionados ao acesso de um *host* interno para a rede externa, e ao acesso de um *host* externo para a rede monitorada, respectivamente.

Estes limiares foram configurados para alertar ocorrências, onde um *host* acessou pelo menos **10** destinos diferentes. Alguns dos resultados mais relevantes, obtidos da execução do *RECON* sobre o tráfego de rede para todo o período em que este foi testado, são apresentados. Em cada linha onde é informada a direção (*direction*) dos acessos, isto é, se

estes partiram de dentro da rede monitorada para fora, ou de fora para dentro, a direção observada foi editada e removida.

```
================================================================
| Host Scan Logs |
+----------------+
direction    source IP          num. dests    resource
---------    ---------------    ----------    ----------------
*********    host-a             20            00113 / tcp
*********    host-b             31            00113 / tcp
*********    host-c             268           00025 / tcp
*********    host-d             14            00113 / tcp
*********    host-e             11            00025 / tcp
*********    host-f             20            00025 / tcp
*********    host-g             12            00113 / tcp
*********    host-h             20            00113 / tcp
*********    host-i             10            00025 / tcp
*********    host-j             25            00025 / tcp
*********    host-k             14            00113 / tcp
*********    host-l             21            00025 / tcp
*********    host-m             18            00025 / tcp
*********    host-n             3964          echo / icmp
================================================================
```

A última linha do resultado apresentado indica que o ***host-n*** acessou **3964** *hosts* distintos, no intervalo de uma hora, através da utilização de mensagens ICMP do tipo ***echo***. Esta ocorrência caracteriza a utilização de uma das técnicas de varredura mais triviais, onde o objetivo é descobrir quais *hosts* estão ativos em um ou mais domínios.

A linha de maior relevância corresponde aos acessos do ***host-c*** a **286** *hosts* distintos, na porta 25/tcp. Esta porta é normalmente utilizada por servidores de SMTP, no envio e recebimento de mensagens de correio eletrônico. Esta atividade ocorreu durante 40 horas consecutivas, com picos de 586, 665, 702 e 807 acessos a *hosts* distintos, em horas e dias diferentes.

As sessões correspondentes aos acessos observados foram reconstruídas, através da execução do *RECON* com a opção "**-t**", de modo que uma saída resumida fosse apresentada. Parte da comunicação entre o ***host-c*** e um dos *hosts* acessados é mostrado logo abaixo.

```
[ smtp-serv-n <=> host-c ]
  TCP sessions
    session 1 [ 25 (s) <=> 4876 (c) ] -> state ( Fin2_Closed ), \
              duration ( 12:00:19.243924 - 12:00:20.736278 - diff = 01s 492354ms ),
              attributes ( none ), \
              payload rcvd ( smtp-serv-n - 133 b , host-c - 336 b , biggest packet 100 , 20 packets ).
    session 2 [ 25 (s) <=> 4916 (c) ] -> state ( Rst_Closed ), \
              duration ( 12:01:56.290414 - 12:01:56.469436 - diff = 179022ms ), \
              attributes ( Syn_RST ), \
              payload rcvd ( smtp-serv-n - 0 b , host-c - 0 b , biggest packet 0 , 2 packets ).
    ...
                                                                                    continua...
```

```
...
session 10 [ 25 (s) <=> 1155 (c) ] -> state ( Rst_Closed ), \
           duration ( 12:06:46.154101 - 12:06:46.333630 - diff = 179529ms ), \
           attributes ( Syn_RST ), \
           payload rcvd ( smtp-serv-n - 0 b , host-c - 0 b , biggest packet 0 , 2 packets ).
...
```

O ***host-c*** inicia uma sessão com o ***smtp-serv-n*** na porta 25/tcp. Dados são enviados e recebidos e a sessão é terminada. Em um curto intervalo de tempo uma tentativa de abertura de sessão é realizada, mas a porta 25/tcp encontra-se indisponível. Este comportamento é repetido por várias vezes. Pode-se observar que há uma lacuna entre o número da porta utilizada pelo ***host-c*** na primeira sessão e na tentativa de abertura da segunda sessão. Isto ocorreu porque várias outras sessões estavam sendo abertas com os outros *hosts*. Então, foi caracterizado que os *hosts* não estavam suportando o grande número de tentativas de abertura de sessões.

Não havia uma grande variação nos *hosts* acessados em diferentes horas, e as sessões apresentavam as mesmas características. Houve, então, o forte indício de que o ***host-c*** estava realizando uma varredura, mas sendo utilizado para enviar mensagens de correio eletrônico com propagandas comerciais, conhecidas como SPAM. O administrador da rede onde estava localizado o ***host-c*** foi contactado, e pôde-se comprovar que este disponibilizava um serviço HTTP com uma funcionalidade mal configurada, e que permitia o envio indiscriminado de mensagens de correio eletrônico. Este serviço estava sendo explorado e utilizado em tais atividades.

As outras ocorrências mostradas no quadro com o *host scan* anterior estavam associadas ao envio normal de mensagens de correio eletrônico de um servidor SMTP para outros na porta 25/tcp, e a identificação de usuário normalmente realizada em aplicações de FTP, através da utilização da porta 113/tcp. Uma possibilidade para retirar estas ocorrências dos resultados é a criação de uma lista de servidores conhecidos, associados à portas conhecidas, com limiares específicos. Estes limiares podem ser estabelecidos através da própria análise dos resultados gerados pelo *RECON*, caracterizando assim um perfil de comportamento esperado para estes servidores.

Outro resultado relevante observado durante o período em que o *RECON* foi testado é apresentado a seguir:

```
================================================================
| Host Scan Logs |
+----------------+
direction    source IP          num. dests    resource
---------    ---------------    ----------    ----------------
*********    host-p             391           01214 / tcp
*********    host-q             81            04662 / tcp
*********    host-r             201           01214 / tcp
================================================================
```

As portas 1214/tcp (catalogada em IANA (2002)) e 4662/tcp são relacionadas à aplicações *p2p*, normalmente, associadas ao compartilhamento de arquivos *mp3* através da rede. A natureza destas aplicações justifica o grande número de acessos observados no resultado anterior, sendo que estes ocorreram de forma análoga durante todo o período em que o *RECON* analisou o tráfego de rede. A questão é que a utilização destes tipos de serviço pode não estar em conformidade com a política de uso adequado das redes e sistemas de informação de uma organização, e são conhecidos por consumirem uma grande quantidade de recursos da rede onde estão sendo disponibilizados. Apesar de não indicarem uma varredura, estes resultados fornecem boas fontes de informação relacionadas a utilização da rede em questão.

Atualmente, ferramentas para a realização de varreduras específicas vêm sendo amplamente utilizadas, onde estas procuram identificar a versão do serviço (aplicação) executado em uma determinada porta TCP. Existem vários serviços que enviam no primeiro pacote contendo dados, após o estabelecimento da conexão, informações que indicam seu nome e sua versão (*banner*). Um intruso, de posse destas informações, pode utilizar em um ataque a ferramenta certa, desenvolvida para explorar uma vulnerabilidade conhecida e relacionada a uma versão específica do serviço. Este tipo de varredura é conhecido como *banner scan*.

Baseado nestas idéias, foi acrescentado à implementação da rotina de detecção de varreduras de *hosts* do *RECON* um conjunto de verificações adicionais para dois serviços específicos: o FTP e o SSH. A implementação deste conjunto de verificações baseou-se em duas ferramentas: o *ftpscan*[2] e o *scanssh*[3].

Estas duas ferramentas estabelecem uma conexão com a porta 21/tcp e 22/tcp, respectivamente, e após receberem as informações de nome e versão do serviço, abortam a conexão, enviando um pacote de RST, ou fazem um encerramento normal. Portanto, para detectar estas varreduras o limiar especificado anteriormente continua sendo utilizado, mas para acessos destinados às portas 21/tcp e 22/tcp o número de pacotes enviados do

[2]Pode ser encontrado em: `http://www.secureaustin.com/nlog/nlog-1.6.0/tools/`.
[3]Pode ser encontrado em: `http://http://monkey.org/~provos/scanssh/`.

servidor para o *host* em questão é avaliado. Retransmissões são checadas e descartadas da contagem, caso ocorram.

São utilizados os seguintes parâmetros, que devem estar presentes no arquivo de configuração do *RECON* (ver apêndice B.2):

- *FIND_HOST_SCAN_WITH_BANNER*: este parâmetro diz para o *RECON* que a busca por varreduras de *hosts* deve ser realizada, mas com a checagem adicional em busca de *banner scans* para as portas 21/tcp e 22/tcp;
- *FTP_BANNER_PKTS_FROM_SERVER*: este parâmetro indica qual é o número de pacotes contendo o *banner*, enviados do servidor FTP para o *host*;
- *SSH_BANNER_PKTS_FROM_SERVER*: este parâmetro indica qual é o número de pacotes contendo o *banner*, enviados do servidor SSH para o *host*;

O parâmetro que indica o número de pacote enviados pelo servidor é configurado para **2** no caso do FTP e para **1** no caso do SSH. Estes valores foram determinados, através da avaliação do tráfego gerado na utilização das ferramentas *ftpscan* e *scanssh*, tomadas como base para o desenvolvimento das verificações adicionais.

O saída abaixo, gerada pelo *RECON*, ilustra uma das situações onde foi realizada uma varredura do tipo *banner scan* para o serviço de FTP, detectada pela rotina implementada.

```
==================================================================
| Host Scan Logs |
+----------------+
direction    source IP          num. dests    resource            banner grab. sessions
---------    ---------------    ----------    ----------------    ---------------------
*********    scanner-host       156           00021 / tcp         82
==================================================================
```

O resultado apresentado diz que ***scanner-host*** acessou **156** *hosts* distintos na porta 21/tcp no intervalo de uma hora e, destes 156 acessos, 82 foram caracterizados como sessões de varredura para obtenção de *banner*.

As sessões TCP, relacionadas ***scanner-host*** e à porta 21/tcp foram reconstruídas, através da execução do *RECON* com a opção "**-t**", gerando assim uma saída resumida. Parte desta saída é apresentada a seguir:

```
[ host-x <=> scanner-host ]
  TCP sessions
    session 1 [ 21 (s) <=> 1760 (c) ] -> state ( Rst_Closed ), \
              duration ( 01:16:16.424417 - 01:16:16.584467 - diff = 160050ms ), \
              attributes ( Syn_RST ), \
              payload rcvd ( host-x - 0 b , scanner-host - 0 b , biggest packet 0 , 2 packets ).
    session 2 [ 21 (s) <=> 1760 (c) ] -> state ( Rst_Closed ), \
              duration ( 01:16:17.305382 - 01:16:17.464738 - diff = 159356ms ), \
              attributes ( Syn_RST ), \
              payload rcvd ( host-x - 0 b , scanner-host - 0 b , biggest packet 0 , 2 packets ).
    ...
...
[ ftp-serv-w <=> scanner-host ]
  TCP sessions
    session 1 [ 21 (s) <=> 1245 (c) ] -> state ( Rst_Closed ), \
              duration ( 01:13:47.307012 - 01:13:51.093051 - diff = 03s 786039ms ), \
              attributes ( none ), \
              payload rcvd ( ftp-serv-w - 0 b , scanner-host - 79 b , biggest packet 42 , 9 packets ).
    ...
...
```

As sessões 1 e 2 entre o ***scanner-host*** e ***host-x*** mostram que a porta 21/tcp do ***host-x*** não estava aberta. O atributo ***Syn_RST*** indica que um pacote RST foi enviado em resposta ao pacote SYN do ***scanner-host***.

E a sessão 1 entre o ***scanner-host*** e ***ftp-serv-w*** mostra que a porta 21/tcp do ***ftp-serv-w*** estava aberta, a sessão foi criada e 79 bytes foram recebidos pelo ***scanner-host***.

A saída mais detalhada, gerada pela execução do *RECON* com a opção "**-T**" para esta última sessão é apresentada abaixo.

```
[ ftp-serv-w <=> scanner-host ]
  TCP sessions
    session 1 [ 21 (s) <=> 1245 (c) ]
      (<-) 01:13:47.307012 S 50527054:50527054(0) ack 0 win 8192 [ip - hd_len 20 len 48 ...]
      (->) 01:13:47.613496 SA 914461385:914461385(0) ack 50527055 win 5840 [ip - hd_len 20 len 48 ...]
      (<-) 01:13:48.109228 A 50527055:50527055(0) ack 914461386 win 8576 [ip - hd_len 20 len 40 ...]
      (<-) 01:13:48.121909 AF 50527055:50527055(0) ack 914461386 win 8576 [ip - hd_len 20 len 40 ...]
      (->) 01:13:48.426781 A 914461386:914461386(0) ack 50527056 win 5840 [ip - hd_len 20 len 40 ...]
      (->) 01:13:50.695368 AP 914461386:914461428(42) ack 50527056 win 5840 [ip - hd_len 20 len 82 ...]
      (->) 01:13:50.695428 APF 914461428:914461465(37) ack 50527056 win 5840 [ip - hd_len 20 len 77 ...]
      (<-) 01:13:51.080727 R 50527056:50527056(0) ack 3762913726 win 0 [ip - hd_len 20 len 40 ...]
      (<-) 01:13:51.093051 R 50527056:50527056(0) ack 50527056 win 0 [ip - hd_len 20 len 40 ...]
      ----------------
      state .........: Reset Closed Ok
      duration ......: 01:13:47.307012 - 01:13:51.093051 - diff = 03s 786039ms
      attributes ....: none
      payload rcvd ..: (ftp-serv-w) 0 bytes, (scanner-host) 79 bytes , biggest packet (42) , 9 packets.
```

Pode-se observar que os 79 bytes recebidos pelo ***scanner-host*** provêem de **2** pacotes enviados pelo ***ftp-serv-w***. Em seguida a sessão é abortada pelo ***scanner-host***.

Um exemplo de outro caso observado nos resultados gerados pelo *RECON*, onde foi detectada uma varredura do tipo *banner scan* para o serviço de SSH, é apresentado logo abaixo.

```
================================================================
| Host Scan Logs |
+----------------+
direction    source IP          num. dests    resource            banner grab. sessions
---------    ---------------    ----------    ----------------    ---------------------
*********    scanner-host       19            00022 / tcp         2
================================================================
```

O resultado apresentado diz que ***scanner-host*** acessou **19** *hosts* distintos na porta 22/tcp no intervalo de uma hora e, destes 19 acessos, 2 foram caracterizados como sessões de varredura para obtenção de *banner*.

A saída mais detalhada, gerada pela execução do *RECON* com a opção "**-T**" para uma das sessões que caracterizou o *banner scan*, é apresentada abaixo.

```
[ ssh-serv-y <=> scanner-host ]
  TCP sessions
    session 1 [ 22 (s) <=> 1552 (c) ]
      (<-) 16:56:38.345818 S 1720751383:1720751383(0) ack 0 win 32120 [ip - hd_len 20 len 60 ...]
      (->) 16:56:38.346875 SA 2097474906:2097474906(0) ack 1720751384 win 5792 [ip - hd_len 20 len 60 ..]
      (<-) 16:56:38.347000 A 1720751384:1720751384(0) ack 2097474907 win 32120 [ip - hd_len 20 len 52 ..]
      (->) 16:56:38.357915 AP 2097474907:2097474930(23) ack 1720751384 win 5792 [ip - hd_len 20 len 75 .]
      (<-) 16:56:38.358061 A 1720751384:1720751384(0) ack 2097474930 win 32120 [ip - hd_len 20 len 52 ..]
      (<-) 16:56:38.358556 AP 1720751384:1720751412(28) ack 2097474930 win 32120 [ip - hd_len 20 len 80.]
      (<-) 16:56:38.358666 AF 1720751412:1720751412(0) ack 2097474930 win 32120 [ip - hd_len 20 len 52 .]
      (->) 16:56:38.359660 A 2097474930:2097474930(0) ack 1720751412 win 5792 [ip - hd_len 20 len 52 ...]
      (->) 16:56:38.362873 AF 2097474930:2097474930(0) ack 1720751413 win 5792 [ip - hd_len 20 len 52 ..]
      (<-) 16:56:38.363052 A 1720751413:1720751413(0) ack 2097474931 win 32120 [ip - hd_len 20 len 52 ..]
      ----------------
      state .........: Simultaneous Fin2 Closed Ok
      duration ......: 16:56:38.345818 - 16:56:38.363052 - diff = 017234ms
      attributes ....: none
      payload rcvd ..: (ssh-serv-y) 28 bytes, (scanner-host) 23 bytes , biggest packet (28) , 10 packets.
```

Pode-se observar que os 23 bytes recebidos pelo ***scanner-host*** provêem de **1** único pacote enviado pelo ***ssh-serv-y***. Em seguida, a sessão é terminada pelo ***scanner-host***. O pacote enviado do ***scanner-host*** para o ***ssh-serv-y*** não influencia na detecção.

6.8.4 Investigação do Uso de ICMP/Echo

Sabe-se que o tráfego ICMP, do tipo ***echo***, é intensivamente utilizado em redes, baseadas na pilha de protocolos TCP/IP, para gerenciamento, testes e análise de desempenho destas redes. Normalmente, é encapsulado no pacote ICMP de requisição uma sequência de bytes aleatória, e esta mesma sequência é retornada no pacote de resposta. Portanto,

os pacotes têm conteúdos semelhantes. A questão é que estes conteúdos e mesmo seus tamanhos normalmente não são checados.

Uma ferramenta desenvolvida para explorar esta lacuna (*LOKI*), utiliza pacotes deste tipo, para enviar e receber informações arbitrárias, caracterizando assim o uso de um túnel, ou *covert channel*. Assim, comandos do sistema podem ser enviados em pacotes ICMP de requisição, de modo que o resultado da execução destes comandos é retornado em mensagens ICMP de resposta.

Baseado nestas idéias, uma rotina foi implementada e integrada ao *RECON*, no intuito de detectar este uso indevido, que na maioria dos casos, indica a invasão e o comprometimento de um sistema.

A rotina busca por diferenças no tamanho dos conteúdos enviados nos pacotes de requisição e recebidos nos pacotes de resposta, para as sessões ICMP reconstruídas à partir do tráfego de rede analisado. Seu desenvolvimento foi baseado na ferramenta *LOKI*[4] (daemon9, 1996) (daemon9, 1997).

São utilizados os seguintes parâmetros, que devem estar presentes no arquivo de configuração do *RECON* (ver apêndice B.2):

- *DETECT_COVERT_CHANNELS*: este parâmetro diz para o *RECON* que as sessões ICMP do tipo ***echo*** devem ser checadas, em busca da utilização de "túneis".
- *COVERT_CHANNELS_REQ_REPLY_DIFF*: este parâmetro é utilizado para reduzir as buscas nas sessões ICMP reconstruídas. Por exemplo, se o valor a ele associado for igual a **1**, nenhuma sessão ICMP com o número de pacotes de requisição igual ao número de pacotes de resposta será checada. Para os casos onde a diferença entre respostas e requisições for pelo menos igual a 1, a checagem é realizada.

O resultado gerado pelo *RECON*, e apresentado abaixo, ilustra uma detecção deste gênero.

```
==================================================================
| ICMP/Echo Investigation |
+-------------------------+
IP's: host-a <=> host-b - init 18:55:36.251989 , end 18:55:37.249949
resource [ ICMP/Echo ] , id [ 15726 ] , excd. replies [ 2 ] , attributes [ none ]
WARNING: different payload lenghts for icmp packets.
==================================================================
```

[4]Implementação pode ser encontrada em: `http://www.phrack.org`.

Para avaliar mais detalhadamente a sessão associada à detecção apresentada, o *RECON* foi executado com a opção "**-I**", sobre o mesmo tráfego. O resultado é apresentado logo abaixo.

```
[ host-a <=> host-b ]
  ICMP Info sessions
    session 1 [ echo ] id 15726
      (<-) 18:55:36.251989  request seq 0 [ip - hd_len 20 len 84 id 52337 ttl 64] [icmp - type 8 code 0]
      (->) 18:55:36.253330  reply seq 0 [ip - hd_len 20 len 84 id 64434 ttl 255] [icmp - type 0 code 0]
      (->) 18:55:37.249121  reply seq 0 [ip - hd_len 20 len 100 id 52339 ttl 64] [icmp - type 0 code 0]
      (->) 18:55:37.249949  reply seq 0 [ip - hd_len 20 len 86 id 64435 ttl 255] [icmp - type 0 code 0]
      ----------------
      duration ......: 18:55:36.251989 - 18:55:37.249949 - diff = 997960ms
      attributes ....: none
      match_reply ...: -2
      payload rcvd ..: (hoat-a) 128 bytes , (host-b) 114 bytes , \
                       biggest packet (72) , 4 packets.
```

Pode-se observar que o primeiro pacote de resposta contém o mesmo tamanho do pacote de requisição, mas os pacotes de resposta subsequentes, apesar de terem o mesmo *id* e número de sequência (*seq*) do pacote de requisição, apresentam tamanhos diferentes.

Apesar desta rotina detectar o uso indevido das mensagens ICMP do tipo ***echo***, não há impedimentos para o uso de outros tipos de mensagens ICMP de informação, portanto a rotina de detecção deve ser extendida para abranger outros casos.

6.8.5 Detecção de Ferramentas Automatizadas para a Criação de *Backdoors*

Para concretizar o comprometimento de um sistema é necessária a realização de um conjunto de operações. Estas operações, normalmente, iniciam com a varredura de um domínio ou sistema, em busca de uma vulnerabilidade conhecida, a escolha de um ou mais *hosts*-alvo, e a realização do ataque para este(s) *host(s)*. Caso o ataque seja bem sucedido, o intruso normalmente habilita uma porta no sistema comprometido, onde é executado um serviço que permite a sua volta, sem que o ataque tenha que ser realizado novamente. Esta porta habilitada pelo intruso é conhecida como *backdoor*.

Atualmente, estes ataques estão se tornando mais sofisticados, de modo que existem ferramentas que realizam todas essas operações de forma automatizada. Algumas destas ferramentas apresentam o seguinte comportamento: testam se a porta onde o *backdoor* será instalado está habilitada; caso não esteja, realizam o ataque na porta onde o serviço vulnerável está sendo executado; habilitam a *backdoor*, e realizam uma conexão na *backdoor*, onde uma série de atividades são executadas no *host* recém-comprometido.

Uma característica marcante das atividades realizadas por estas ferramentas é que todas as operações descritas no parágrafo acima são realizadas em curto intervalo de tempo, na

maioria dos casos dentro de um mesmo segundo.

Baseado nestas idéias, uma rotina foi implementada e integrada ao *RECON*, no intuito de detectar o uso destas ferramentas. Esta rotina busca por duas sessões TCP, abertas concomitantemente ou em um curto intervalo de tempo, e checa se uma tentativa de conexão com a segunda porta foi realizada antes do início das sessões. Esta checagem é realizada para cada par de endereços IP, onde existam sessões TCP associadas.

São utilizados os seguintes parâmetros, que devem estar presentes no arquivo de configuração do *RECON* (ver apêndice B.2):

- *DETECT_BACKDOORS*: este parâmetro diz para o *RECON* que as sessões TCP reconstruídas devem ser checadas, em busca da detecção de *backdoors*;
- *BACKDOOR_TIME_TOLERANCE*: este parâmetro é utilizado para indicar o intervalo de tempo a ser checado, em segundos, entre a primeira conexão e a segunda, caso a primeira tenha sido terminada antes do início da segunda.

O saída gerada pelo *RECON*, e apresentada abaixo, ilustra algumas situações deste gênero, detectadas pela rotina implementada. Cada linha onde é informada a direção (*direction*) dos acessos, isto é, se estes partiram de dentro da rede monitorada para fora, ou de fora para dentro, foi editada e a direção observada foi removida.

```
================================================================
| TCP Backdoor Detection for Automated Tools |
+--------------------------------------------+
direction: ******
PROBE    : host-a (34230) -> host-d [10007] - state (Reset Closed Ok) - attr. ( Syn_RST ) \
                                            - init 14:02:46.095267 , end 14:02:55.890798
access   : host-a (34277) -> host-d (10002) - state (Fin2 Closed Ok) - attr. ( none ) \
                                            - init 14:02:59.147389 , end 14:02:59.452274
backdoor : host-a (34281) -> host-d [10007] - state (Reset Closed Ok) - attr. ( none ) \
                                            - init 14:02:59.584230 , end 14:02:59.931630
================================================================

================================================================
| TCP Backdoor Detection for Automated Tools |
+--------------------------------------------+
direction: ******
PROBE    : host-a (53889) -> host-d [08888] - state (Reset Closed Ok) - attr. ( Syn_RST ) \
                                            - init 15:01:35.799028 , end 15:01:44.288762
access   : host-a (53902) -> host-d (00443) - state (Fin2 Closed Ok) - attr. ( none ) \
                                            - init 15:01:47.872501 , end 15:03:13.874060
backdoor : host-a (53973) -> host-d [08888] - state (Fin2 Closed Ok) - attr. ( none ) \
                                            - init 15:03:10.301851 , end 15:03:14.570973
================================================================

                                                                              continua...
```

```
================================================================
| TCP Backdoor Detection for Automated Tools |
+--------------------------------------------+
direction: ******
PROBE    : host-b (01590) -> host-c [05443] - state (Reset Closed Ok) - attr. ( Syn_RST ) \
                                            - init 15:42:43.926817 , end 15:42:43.931631
access   : host-b (01667) -> host-c (00443) - state (Fin2 Closed Ok) - attr. ( none ) \
                                            - init 15:55:43.612203 , end 15:55:52.817577
backdoor : host-b (01668) -> host-c [05443] - state (Fin2 Closed Ok) - attr. ( none ) \
                                            - init 15:55:49.818310 , end 15:55:53.825410
================================================================
```

O *RECON* foi, então, utilizado para avaliar as sessões TCP correspondentes às detecções apresentadas e após uma análise aprofundada destas sessões, pôde-se observar que os *hosts* para onde os acessos foram direcionados executavam aplicações HTTP nas portas que foram acessadas. Estas aplicações apresentaram um comportamento semelhante ao das ferramentas de ataque para as quais a rotina foi desenvolvida. Isto caracteriza um falso-positivo gerado pela rotina.

Detecções de eventos realmente associados a invasões não puderam ser observados durante todo o período em que o *RECON* foi executado. Este fato indica que a rotina apresenta um alto índice de falso-positivos. Além disso, o fato da rotina estar baseada apenas no período de ocorrência dos acessos às portas não é suficiente para concluir, detectar e alertar que o tipo de ferramenta de ataque, objetivo de seu desenvolvimento, foi utilizada para realizar tais acessos.

Há, então, a necessidade de realizar um estudo mais aprofundado destas ferramentas e do tipo de tráfego de rede que elas geram, para que outros parâmetros possam ser associados à rotina de detecção, e os falso-positivos sejam reduzidos ou até eliminados.

CAPÍTULO 7

CONCLUSÕES

Este trabalho apresentou um sistema baseado em um modelo de reconstrução de sessões TCP/IP, que faz uso dos cabeçalhos de pacotes extraídos do tráfego de redes. Os resultados obtidos a partir do *RECON* indicam que é viável utilizar um número reduzido de informações por pacote obtido do tráfego de redes TCP/IP, e a partir daí associá-las de modo que estas representem, na forma de sessões, o fluxo de dados observado. Mostrou também possibilidades de aplicações desta metodologia em atividades relacionadas à detecção de intrusão, onde a tomada de decisões é realizada através da análise de um conjunto de informações. Esta característica fornece ao *RECON* a capacidade de detectar atividades hostis subversivas, que tentam iludir os sistemas de detecção.

O *RECON* mostrou ser um sistema capaz de tratar uma quantidade considerável de dados obtidos do tráfego de rede, visto que este utiliza um número reduzido de informações obtidas de cada pacote. Além disso, poucos recursos são exigidos de um equipamento executando o *RECON*, de modo que este tem um baixo custo atrelado. De maneira oposta, os sistemas de detecção de intrusão comerciais atualmente disponíveis apresentam grandes exigências relacionadas à alocação de recursos, tratamento de dados e causam um grande impacto sobre a performance dos equipamentos envolvidos.

Com relação à extrapolação do conceito relacionado à sessão, não houve grandes dificuldades em sua aplicação para o protocolo ICMP, visto que seu cabeçalho dispõe de campos bem definidos, com números de identificação e sequência, que permitem distinguir os pacotes de diferentes sessões.

A maior dificuldade ocorreu na identificação e distinção entre diferentes sessões para o protocolo UDP. As informações que compõem os pacotes UDP e que são utilizadas na associação de um conjunto de pacotes a uma mesma sessão, fazem parte do conteúdo da mensagem. Portanto, esta associação depende de detalhes de implementação dos protocolos da camada de aplicação. A dificuldade, então, está na análise dos vários protocolos de aplicação disponíveis, onde faz-se necessário identificar se o protocolo disponibiliza alguma informação que possibilite esta associação e se esta informação está contida na sequência de bytes capturados por pacote.

Ao contrário do que se pensava, para o protocolo TCP, que está diretamente associado ao conceito de sessão, algumas dificuldades também foram encontradas. Após exaustiva análise do tráfego de redes TCP/IP, pôde-se observar uma grande variação na combinação de *flags* dos pacotes, que são de fundamental importância para a distinção entre sessões.

Desta forma, um grande número de possibilidades foi adicionado ao desenvolvimento do *RECON*, mas alterações ou inclusões de novos casos ou casos não observados devem ser considerados.

O desenvolvimento deste sistema resultou em uma ferramenta que pode ser aplicada não só à detecção de intrusões, mas também no gerenciamento, monitoramento e estudo do comportamento de redes de computadores.

As próximas seções apresentam algumas sugestões e propostas para a continuação destes estudos, bem como as considerações finais e contribuições deste trabalho.

7.1 SUGESTÕES PARA TRABALHOS FUTUROS

Uma primeira sugestão seria a de criar uma biblioteca de rotinas. O modelo para reconstrução de sessões TCP/IP desenvolvido está integrado a todos os outros módulos que compõem o *RECON* e constitui o núcleo deste sistema. Portanto, a primeira necessidade para tornar este modelo independente de uma aplicação específica é transformá-lo em um biblioteca. A idéia é, então, modificar as principais funções responsáveis pela reconstrução das sessões e criar uma API (*Application Programming Interface*), ou seja, uma interface para a programação de aplicações que permita o desenvolvimento de diversas ferramentas, não só para a detecção de intrusão, mas também para o monitoramento e estudo do comportamento de redes.

Em seguida, são discutidas algumas propostas para o aperfeiçoamento de alguns dos módulos de reconstrução de sessões e rotinas aplicadas à detecção de atividades maliciosas, bem como para a expansão do sistema desenvolvido.

7.1.1 Estruturas de Armazenamento do Tráfego de Rede

Os módulos do *RECON* utilizam, além das árvores binárias, listas encadeadas para armazenar informações. Pode-se considerar que a utilização destas listas é essencial em alguns casos, pois mantém a sequência associada à ocorrência de eventos. Este é o caso da lista encadeada de pacotes, associada a cada sessão reconstruída.

Mas para os outros casos, tais como a lista de adjacências dos nós da árvore de IPs, a lista de elementos de desfragmentação e as listas de sessões, pode-se substituí-las por árvores binárias balanceadas. Esta alteração impacta diretamente no desempenho do sistema, pois sabe-se que as operações de busca em uma árvore têm complexidade logarítmica, ao passo que as mesmas operações em uma lista têm complexidade linear. Isto representa um ganho de três ordens de magnitude, em relação a tais operações.

7.1.2 Módulos Utilizados na Reconstrução de Sessões

Apesar do *RECON* tratar uma grande quantidade de situações que influenciam na criação e atualização dos dados de sessões TCP/IP, ainda existem lacunas a serem preenchidas. Portanto, são feitas algumas sugestões e propostas para seu aperfeiçoamento, de acordo com os módulos responsáveis pela reconstrução das sessões.

O módulo de processamento de pacotes, responsável por tratar o cabeçalho do protocolo da camada de enlace de dados, considera apenas pacotes utilizando o protocolo *Ethernet*. Propõe-se, então, a extensão deste módulo para o tratamento de outros protocolos desta camada.

O módulo de desfragmentação, responsável por remontar datagramas IP fragmentados, não trata o caso onde os dados do cabeçalho da camada de transporte foram divididos. Logo, há uma proposta para que este módulo seja aperfeiçoado, de modo que permita remontar não só o datagrama, mas também o cabeçalho da camada de transporte, caso seja possível e que possa alertar sobre tais ocorrências.

O módulo responsável pelo processamento das mensagens UDP realiza um tratamento diferenciado para os protocolos DNS e NTP da camada de aplicação. Nestes dois casos, são extraídas do início do conteúdo da mensagem UDP informações que permitem verificar se o pacote é uma requisição ou uma resposta. Existem outros protocolos da camada de aplicação que têm este mesmo comportamento e possibilitam o mesmo tipo de análise. Portanto, sugere-se que rotinas para o tratamento de outros protocolos da camada de aplicação sejam agregadas ao *RECON*.

E por fim, o módulo responsável pelo processamento das mensagens TCP não checa o *sequence number* e o *acknowledgement number* de todo e qualquer pacote TCP. No sistema desenvolvido, estas verificações são realizadas a partir do momento em que é observado um pacote solicitando o término de conexão, ou para associar pacotes RST recorrentes às suas respectivas sessões. O fato é que existem técnicas relacionadas à atividades hostis, para a subversão de sistemas de detecção de intrusão, que manipulam os números de sequência de pacotes TCP, e têm como finalidade realizar a sobreposição de dados (Ptacek e Newsham, 1998). Desta forma, sugere-se que sejam armazenados em cada sessão TCP os próximos *sequence numbers* esperados e os últimos *acknowledgement numbers* recebidos, associados aos dois IPs da sessão. Estes números poderiam ser checados e atualizados, à medida que pacotes TCP forem sendo processados, de modo que tais atividades hostis possam ser alertadas.

7.1.3 Rotinas para a Aplicação em Detecção de Intrusão

Algumas rotinas para a aplicação em detecção de intrusão foram desenvolvidas e integradas ao *RECON*, mas como observado, aperfeiçoamentos podem ser realizados e outros critérios podem ser agregados à estas rotinas, no intuito de obter resultados mais apurados.

A rotina responsável pela detecção de *host scans* realiza as verificações nas sessões TCP e ICMP reconstruídas. Para o caso das sessões TCP, não é feita a correlação com informações da árvore de *logs* TCP. Portanto, há uma sugestão para que estas sejam analisadas em conjunto, de modo a complementar os resultados gerados por esta rotina. Além disso, há a necessidade de criação de uma lista de servidores internos à rede monitorada, associados a limiares específicos, de forma que checagens diferenciadas sejam realizadas, reduzindo assim a ocorrência de falso-positivos. E ainda há a sugestão de extensão desta rotina para a checagem das sessões UDP.

Outra sugestão refere-se à rotina de investigação da utilização de mensagens ICMP do tipo ***echo***. Através do estudo dos outros tipos de mensagens ICMP de informação, pode-se verificar a viabilidade de se extender as checagens realizadas pela rotina para abranger os outros tipos.

Em relação à rotina desenvolvida para a detecção de *backdoors* criados por ferramentas automatizadas, há uma proposta para a realização de um estudo aprofundado acerca do funcionamento destas. Através deste estudo, pode-se aperfeiçoar os critérios utilizados na geração de um alerta, de modo que os falso-positivos observados sejam reduzidos.

E por fim, sugere-se a criação de uma rotina para checar as listas de pacotes ICMP de erro. Os resultados gerados por esta rotina podem fornecer uma relação de endereços IP da rede interna que possivelmente estão sendo forjados (*spoofing*).

7.2 CONSIDERAÇÕES FINAIS

De forma geral, entende-se que esta pesquisa forneceu uma contribuição para o desenvolvimento de aplicações em detecção de intrusão, baseadas em uma abordagem alternativa e utilizando um número reduzido de recursos. Através da aplicação de metodologias relativamente simples, tais como a associação do conceito de grafos ao tráfego de redes, a utilização de árvores binárias balanceadas, para melhorar o desempenho, e a extrapolação do conceito associado à "sessão", foi possível especificar, implementar e testar um novo modelo para fornecer meios de tratar um problema comum em muitas aplicações associadas à detecção de intrusão, que é a tomada de decisões baseadas em informações

isoladas.

Uma possível aplicação do sistema desenvolvido consiste em seu uso como uma ferramenta de análise *pos-mortem* que atuaria de forma complementar com um sistema de detecção de intrusão por padrão intrusivo de tempo real, visto que permitiria detectar alguns tipos de ataques que, normalmente, não são observados pelo último. Isto seria feito por meio da correlação de pacotes e sessões, algo que o *Snort*, por exemplo, não faz.

Por este assunto ser extremamente abrangente, houve também a preocupação de que o desenvolvimento da pesquisa apresentada possa servir como referência para novos estudos nesta área.

Com relação às vantagens da modelagem e do sistema desenvolvido pode-se considerar que possibilita a redução de falso-positivos e falso-negativos em aplicações de detecção de intrusão, pois decisões podem ser tomadas através da análise de um conjunto de eventos relacionados a uma ou mais sessões, em contrapartida a eventos isolados. Desta forma, permite correlacionar informações de um conjunto de sessões na identificação de atividades hostis, que não podem ser observadas em uma única sessão. Além disso, o sistema trata uma quantidade relativamente grande de informações, pois utiliza um número reduzido de dados por pacote. Também pode ser utilizado no monitoramento e estudo do comportamento de redes TCP/IP e não necessita da alocação de muitos recursos computacionais, portanto tem custo reduzido.

Com relação às desvantagens pode-se considerar que o sistema utiliza arquivos contendo o tráfego de rede e, portanto, não realiza a reconstrução e análise das informações em tempo real. Também não realiza análises sobre o conteúdo dos pacotes, exceto para uma pequena porção dos dados nas mensagens do protocolo UDP. E para os exemplos de aplicações em detecção de intrusão já desenvolvidas, depende de pessoal especializado, pois a determinação de limiares a partir dos quais um alerta é gerado é de fundamental importância na redução de falsos-positivos.

Para finalizar, é importante ressaltar que a segurança de um sistema de informação deve ser implementada em camadas, ou seja, através de um conjunto de técnicas, procedimentos e mecanismos, envolvendo uma política de segurança bem determinada e estabelecida para a organização, o treinamento adequado dos administradores, o uso e configuração apropriados dos sistemas computacionais, o comprometimento e conscientização dos usuários, e mais...

REFERÊNCIAS BIBLIOGRÁFICAS

Allen, J.; Christie, A.; Fithen, W.; McHugh, J.; Pickel, J.; Stoner, E. **State of the practice of intrusion detection technologies**. Pittsburgh: Carnegie Mellon University, 2000. 220 p. (CMU/SEI-99-TR-028). 28, 56

Amoroso, E. G. **Intrusion detection**: an introduction to internet surveillance, correlation, traps, trace-back, and response. New Jersey: Intrusion.Net Books, 1999. 218 p. 55

Anderson, D.; Frivold, T.; Valdes, A. **Next-generation intrusion detection expert system (NIDES)**: a summary. Menlo Park: SRI International, 1995. 47 p. (SRI-CSL-95-07). 63

Anderson, J. P. **Computer security threat monitoring and surveillance**. Fort Washington: James P. Anderson Co., 1980. 53 p. (79F296400). 54, 55

Arkin, O. **ICMP usage in scanning**: the complete know-how. The Sys-Security Group [online].
<http://www.sys-security.com/archive/papers/ICMP_Scanning_v3.0.pdf>.
July 2002. 36, 37

Bace, R.; Mell, P. **Intrusion detection systems**. Gaithersburg: National Institute of Standards and Technology, 2000. 51 p. (NIST-SP-800-31). 28, 56

Balasubramaniyan, J. S.; Garcia-Fernandez, J. O.; Isacoff, D.; Spafford, E.; Zamboni, D. **An architecture for intrusion detection using autonomous agents**. West Lafayette: Purdue University, 1998. 19 p. (COAST-TR-98-05). 64

Cansian, A. M. **Desenvolvimento de um sistema adaptativo de detecção de intrusos em redes de computadores**. São Carlos. 154 p. Dissertação (Doutorado em Física Aplicada) – Universidade de São Paulo, 1997. 68

Carstens, T. **Programming with pcap**. [online].
<http://www.tcpdump.org/pcap.htm>. July 2002. 92

Case, J.; Fedor, M.; Schoffstall, M.; Davin, J. **A simple network management protocol (SNMP)**. Request for Comments RFC 1157. [online].
<http://www.rfc-editor.org/rfc/rfc1157.txt>. July 2002. 45

Chaves, M. H. P. C.; Montes, A. Modelo de dados para sistema de detecção de intrusão visando a reconstrução de sessões. In: Simpósio sobre Segurança e Informática. **Anais**. São José dos Campos: CTA/ITA, 2001. v. 3, p. 141–150. 75

Clark, D. D. **IP datagram reassembly algorithms**. Request for Comments RFC 815. [online]. <http://www.rfc-editor.org/rfc/rfc815.txt>. July 2002. 102

COAST. **Intrusion detection systems**. [online]. <http://www.cerias.purdue.edu/coast/intrusion-detection/ids.html>. July 2002. 64

Comer, D. E. **Internetworking with TCP/IP**. 4. ed. New Jersey: Prentice Hall, 2000. v. 1: principles, protocols, and architectures. 750 p. 31, 34, 38, 40

Crispin, M. **Internet message access protocol - version 4rev1**. Request for Comments RFC 2060. [online]. <http://www.rfc-editor.org/rfc/rfc2060.txt>. July 2002. 45

Crocker, D. **Standard for the format of ARPA Internet text messages**. Request for Comments RFC 822. [online]. <http://www.rfc-editor.org/rfc/rfc822.txt>. July 2002. 43

daemon9. Project Loki. **Phrack**. [online], v. 5, n. 49, Feb. 1996. <http://www.phrack.org/show.php?p=49&a=6>. 148

——. LOKI2 (the implementation). **Phrack**. [online], v. 7, n. 51, Sep. 1997. <http://www.phrack.org/show.php?p=51&a=6>. 148

Denning, D. E. An intrusion-detection model. **IEEE Transactions on Software Engineering**, v. SE-13, n. 2, p. 222–232, Feb. 1987. 28, 62

Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. **Standard for the format of ARPA Internet text messages**. Request for Comments RFC 822. [online]. <http://www.rfc-editor.org/rfc/rfc822.txt>. July 2002. 44

Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. **Hypertext transfer protocol - HTTP/1.1**. Request for Comments RFC 2616. [online]. <http://www.rfc-editor.org/rfc/rfc2616.txt>. July 2002. 44

Finlayson, R.; Mann, T.; Mogul, J.; Theimer, M. **A Reverse address resolution protocol**. Request for Comments RFC 903. [online]. <http://www.rfc-editor.org/rfc/rfc903.txt>. July 2002. 33

Fyodor. **Nmap - network mapper**. [online]. <http://www.insecure.org/nmap>. July 2002a. 138

——. **Remote OS detection via TCP/IP stack fingerprinting**. [online]. <http://www.insecure.org/nmap/nmap-fingerprinting-article.html>. July 2002b. 138

Garfinkel, S.; Spafford, G. **Pratical UNIX and Internet secutity**. 2. ed. Sebastopol: OReilly & Associates, 1996. 917 p. 53

Gibbons, A. **Algorithmic graph theory**. Cambridge: Cambridge University Press, 1985. 259 p. 79, 80, 81

Giovanni, C. **Fun with packets**: designing a stick. [online]. <http://www.eurocompton.net/stick/papers/Peopledos.pdf>. July 2002. 28

Gonnet, G. H.; Baeza-Yates, R. **Handbook of algorithms and data structures**: in Pascal and C. 2. ed. Boston: Addison-Wesley, 1991. 424 p. 81, 120

Hartmeier, D. Design and performance of the openbsd stateful packet filter (pf). In: USENIX Annual Technical Conference, Monterey, 2002. **Proceedings**. Monterey: USENIX, 2002. FREENIX Track. 27

IANA. **Protocol numbers and assignment services**. [online]. <http://www.iana.org/numbers.htm>. July 2002. 41, 42, 144

Kernighan, B. W.; Ritchie, D. M. **C, a linguagem de programação**: padrão ANSI. Rio de Janeiro: Campus, 1990. 289 p. 51, 87

Kumar, S.; Spafford, E. H. **An application of pattern matching in intrusion detection**. West Lafayette: Purdue University, 1994. 55 p. (CSD-TR-94-013). 56, 60

Kumar, S. **Classification and detection of computer intrusions**. West Lafayette. 165 p. Thesis (Doctor of Philosophy in Computer Sciences) – Purdue University, 1995. 60

Lunt, T. L.; Tamaru, A.; Gilham, F.; Jagannathan, R.; Jalali, C.; Neumann, P. G.; S., J. H.; Valdes, A.; Garvey, T. D. **A real-time intrusion-detection expert system (IDES)**. Washington DC: SRI International, 1992. 166 p. (SRI Project 6784). 63

McCanne, S.; Jacobson, V. The BSD packet filter: a new arquitecture for user-level packet capture. In: Winter USENIX Technical Conference, San Diego, 1993. **Proceedings**. San Diego: USENIX, 1993. p. 259–269. 47, 48, 50

McHugh, J.; Christie, A.; Allen, J. Defending yourself: the hole of intrusion detection systems. **IEEE Software**, v. 17, n. 5, p. 42–51, Set.-Out. 2000. 26, 76

Miller, I. **Protection against a variant of the tiny fragment attack**. Request for Comments RFC 3128. [online]. <http://www.rfc-editor.org/rfc/rfc3128.txt>. July 2002. 102

Mills, D. L. **Network time protocol (version 1)**: specification and implementation. Request for Comments RFC 1059. [online]. <http://www.rfc-editor.org/rfc/rfc1059.txt>. July 2002a. 44

——. **Network time protocol (version 2)**: specification and implementation. Request for Comments RFC 1119. [online]. <http://www.rfc-editor.org/rfc/rfc1119.txt>. July 2002b. 44, 109

——. **Network time protocol (version 3)**: specification and implementation. Request for Comments RFC 1305. [online]. <http://www.rfc-editor.org/rfc/rfc1305.txt>. July 2002c. 44, 109

Mockapetris, P. **Domain names**: concepts and facilities. Request for Comments RFC 1034. [online]. <http://www.rfc-editor.org/rfc/rfc1034.txt>. July 2002a. 43

——. **Domain names**: implementation and specification. Request for Comments RFC 1035. [online]. <http://www.rfc-editor.org/rfc/rfc1035.txt>. July 2002b. 43, 109, 111

Mueller, P.; Shipley, G. DRAGON claws its way to the top. **Network Computing**, p. 45–67, Aug. 20 2001. 29

Mukherjee, B.; Heberlein, L. T.; Levitt, K. N. Network intrusion detection. **IEEE Network**, v. 8, n. 3, p. 26–41, May-June 1994. 61, 64, 65

Myers, J.; Rose, M. **Post office protocol - version 3**. Request for Comments RFC 1939. [online]. <http://www.rfc-editor.org/rfc/rfc1939.txt>. July 2002. 44

Northcutt, S. **Network intrusion detection: an analyst's handbook**. Indianapolis: New Riders, 1999. 267 p. 69

Northcutt, S.; Cooper, M.; Fearnow, M.; Frederick, K. **Intrusion signatures and analysis**. Indianapolis: New Riders, 2001. 408 p. 102

Northcutt, S.; Novak, J. **Network intrusion detection: an analyst's handbook**. 2. ed. Indianapolis: New Riders, 2001. 450 p. 102

Plummer, D. C. **An ethernet address resolution protocol**. Request for Comments RFC 826. [online]. <http://www.rfc-editor.org/rfc/rfc826.txt>. July 2002. 32

Porras, P. A.; Neumann, P. G. **EMERALD**: event monitoring enabling responses to anomalous live disturbances. [online].
<http://www.sdl.sri.com/emerald/emerald-niss97.html>. July 2002. 68

Postel, J. **Internet control message protocol**: DARPA Internet program, protocol specification. Request for Comments RFC 792. [online].
<http://www.rfc-editor.org/rfc/rfc792.txt>. July 2002a. 35

——. **Internet protocol**: DARPA Internet program, protocol specification. Request for Comments RFC 791. [online].
<http://www.rfc-editor.org/rfc/rfc791.txt>. July 2002b. 33, 34, 35

——. **Simple mail transfer protocol**. Request for Comments RFC 821. [online].
<http://www.rfc-editor.org/rfc/rfc821.txt>. July 2002c. 43

——. **Transmission control protocol**: DARPA Internet program, protocol specification. Request for Comments RFC 793. [online].
<http://www.rfc-editor.org/rfc/rfc793.txt>. July 2002d. 38, 39, 41, 115

——. **User datagram protocol**. Request for Comments RFC 768. [online].
<http://www.rfc-editor.org/rfc/rfc768.txt>. July 2002e. 37

Postel, J.; Reynolds, J. **File transfer protocol (FTP)**. Request for Comments RFC 959. [online]. <http://www.rfc-editor.org/rfc/rfc959.txt>. July 2002a. 43

——. **TELNET protocol specification**. Request for Comments RFC 854. [online].
<http://www.rfc-editor.org/rfc/rfc854.txt>. July 2002b. 43

Ptacek, T. H.; Newsham, T. N. **Insertion, evasion, and denial of service: eluding network intrusion detection**. [online]. <http://www.silicondefense.com/research/itrex/archive/tracing-papers/ptacek98insertion.pdf>. July 2002. 28, 102, 140, 155

Ranum, M. J. **Experiences benchmarking intrusion detection systems**. Rockville: NFR Security, Inc., 2001. 10 p. 82

Roesch, M. SNORT - lightweigth intrusion detection for networks. In: Systems Administration Conference, 13., Seattle, 1999. **Proceedings**. Seattle: USENIX, 1999. p. 229–238. LISA'99. 72

Roesch, M.; Green, M. **Snort users manual**. Release 1.9.X. [online].
<http://www.snort.org/docs/SnortUsersManual.pdf>. July 2002. 73

Rooij, G. **Real stateful TCP packet filtering in IP filter**. [online]. <http://home.iae.nl/users/guido/papers/tcp_filtering.ps.gz>. July 2002. 27

Smaha, E. S. Haystack: an intrusion detection system. In: AeroSpace Computer Security Applications Conference, 4., Austin, 1988. **Proceedings**. Austin: IEEE, 1988. p. 37–44. 55, 66

Snapp, S. R.; Brentano, J.; Dias, G. V.; Goan, T. L.; Heberlein, L. T.; Ho, C.; Levitt, K. N.; Mukherjee, B.; Smaha, S. E.; Grance, T.; Teal, D. M.; Mansur, D. **DIDS (distributed intrusion detection system)**: motivation, arquitecture, and an early prototype. [online]. <http://olympus.cs.ucdavis.edu/papers/DIDS.ncsc91.pdf>. May 2000. 66

SRI. **History of intrusion detection at SRI/CSL**. [online]. <http://www.sdl.sri.com/programs/intrusion/history.html>. July 2002. 62, 63

Stevens, W. R. **TCP/IP illustrated**: the protocols. Boston: Addisson-Wesley, 1994. v. 1. 576 p. 32, 33, 34, 35, 38, 39

——. **UNIX network programming**. 2. ed. Upper Saddle River: Prentice Hall, 1998. v. 1, Cap. 2: the transport layer: UDP and TCP, p. 29–53. 39, 41

Stocksdale, G. **CIDER documents**. [online]. <http://www.nswc.navy.mil/ISSEC/CID>. July 2002. 69

Sun. **NFS**: network file system protocol specification. Request for Comments RFC 1094. [online]. <http://www.rfc-editor.org/rfc/rfc1094.txt>. July 2002a. 45

——. **RPC**: remote procedure call protocol specification version 2. Request for Comments RFC 1057. [online]. <http://www.rfc-editor.org/rfc/rfc1057.txt>. July 2002b. 44

Sundaram, A. **An introduction to intrusion detection**. [online]. <http://coast.cs.purdue.edu/pub/doc/intrusion_detection>. July 2000. 56, 57, 59

Teng, H. S.; Chen, K.; Lu, S. C. Security audit trail analysis using inductively generated predictive rules. In: Conference on Artificial Intelligence Applications, 6., Santa Barbara, 1990. **Proceedings**. Los Alamitos: IEEE, 1990. p. 24–29. 58

Wall, L.; Christiansen, T.; Randal, L. S. **Programming perl**. 2. ed. Sebastopol: OReilly & Associates, 1996. 547 p. 70

Ylonen, T.; Kivinen, T.; Saarinen, M.; Rinne, T.; Lehtinen, S. **SSH connection protocol**. Internet Draft. [online]. <http://search.ietf.org/internet-drafts/draft-ietf-secsh-connect-15.txt>. July 2002. 43

Ziemba, G.; Reed, D.; Traina, P. **Security considarations for IP fragment filtering**. Request for Comments RFC 1858. [online]. <http://www.rfc-editor.org/rfc/rfc1858.txt>. July 2002. 102

Zimmerman, D. **The finger user information protocol**. Request for Comments RFC 1288. [online]. <http://www.rfc-editor.org/rfc/rfc1288.txt>. July 2002. 44

APÊNDICE A

MENSAGENS ICMP: TIPOS E CÓDIGOS

TABELA A.1: Tipos e códigos das mensagens ICMP

Tipo	*Código*	*Descrição*
0	0	Echo reply
3	—	Destination unreachable
	0	Net unreachable
	1	Host unreachable
	2	Port unreachable
	3	Protocol unreachable
	4	Fragment needed but DF was set
	5	Source route failed
	6	Destination network unknown
	7	Destination host unknown
	8	Source host isolated
	9	Communication with destination network is administratively prohibited
	10	Communication with destination host is administratively prohibited
	11	Destination network unreachable for type of service
	12	Destination host unreachable for type of service
	13	Communication administratively prohibited
	14	Host precedence violation
	15	Precedence cuttof in effect
4	0	Source quench
5	—	Redirect
	0	Redirect datagram for the network
	1	Redirect datagram for the host
	2	Redirect datagram for type of service and network
	3	Redirect datagram for type of service and host
8	0	Echo request
9	0	Router advertisement
10	0	Router solicitation
11	—	Time exceeded
	0	Time to live exceeded in transit

continua na próxima página

TABELA A.1: (continuação)

Tipo	*Código*	*Descrição*
	1	Fragment reassembly time exceeded
12	—	Parameter problem
	0	Pointer indicates the error
	1	Missing a required option
	2	Bad length
13	0	Timestamp request
14	0	Timestamp reply
15	0	Information request
16	0	Information reply
17	0	Address mask request
18	0	Address mask reply

APÊNDICE B

DETALHES DE IMPLEMENTAÇÃO DO SISTEMA *RECON*

B.1 OS CONTADORES GLOBAIS DO SISTEMA

TABELA B.1: Contadores globais do *RECON*

Contador	*Descrição*
packet_count	número total de pacotes lidos
ip_count	número de endereços IP distintos
connection_count	número de conexões distintas entre pares de endereços IP
trunc_ether_hdr_packets	*frames Ethernet* com tamanho menor que a estrutura do cabeçalho *Ethernet*
non_tcpip_packets	protocolo encapsulado no frame *Ethernet* não é um protocolo TCP/IP
non_ip_packets	protocolo utilizado não é o IP
multicast_tunnel_packets	datagrama IP encapsulando datagrama IP (*multicast tunnel*)
non_tcp_udp_icmp_packets	datagramas IP contendo mensagens diferentes de TCP/UDP/ICMP ou *Multicast*
trunc_ip_hdr_packets	datagramas IP com tamanho menor que a estrutura do cabeçalho IP
trunc_ip_content_packets	datagramas IP com o conteúdo truncado
ip_opts_packets	datagramas IP cujos cabeçalhos contém opções
ip_frag_packets	datagramas IP fragmentados
trunc_icmp_hdr_packets	Mensagens ICMP com tamanho menor que a estrutura do cabeçalho ICMP
icmp_routeradvert_packets	Mensagens ICMP do tipo *router advertisement*
icmp_routersolicit_packets	Mensagens ICMP do tipo *router solicitation*
icmp_res_or_unknown_packets	Mensagens ICMP desconhecidas ou reservadas
icmp_timxceed_frag	Mensagens ICMP do tipo *fragment reassembly time exceeded*
icmp_not_matched_timxceed	Mensagens ICMP do tipo anterior que não foram encontradas na árvore de desfragmentação
icmp_proc_packets	Mensagens ICMP realmente inseridas nas estruturas de reconstrução de sessões

continua na próxima página

TABELA B.1: (continuação)

Contador	*Descrição*
icmp_proc_packets_req	Mensagens ICMP de informação, do tipo *request* inseridas
icmp_proc_packets_rep	Mensagens ICMP de informação, do tipo *reply* inseridas
icmp_proc_packets_error	Mensagens ICMP de erro inseridas
icmp_proc_packets_error_nomatch	Mensagens ICMP de erro não associadas ao pacote gerador do erro
trunc_udp_hdr_packets	Mensagens UDP com tamanho menor que a estrutura do cabeçalho UDP
udp_proc_packets	Mensagens UDP realmente inseridas nas estruturas de reconstrução de sessões
trunc_tcp_hdr_packets	Mensagens TCP com tamanho menor que a estrutura do cabeçalho TCP
tcp_proc_packets	Mensagens TCP realmente inseridas nas estruturas de reconstrução de sessões
tcp_bad_hdr_len	Mensagens TCP com o cabeçalho corrompido
tcp_packets_with_options	Mensagens TCP cujos cabeçalhos contém opções
tcp_log_packets	Mensagens TCP inseridas na árvore de *logs* TCP
tcp_packets_nomatch_reset	Mensagens TCP com o *flag* RST ativo, não associadas a alguma sessão
mem_amount	total de memória alocada pelo sistema
icmp_Isession_mem_amount	total de memória alocada no armazenamento das sessões ICMP de informação
icmp_Ipkt_mem_amount	total de memória alocada no armazenamento dos pacotes ICMP de informação
icmp_Epkt_mem_amount	total de memória alocada no armazenamento dos pacotes ICMP de erro
udp_session_mem_amount	total de memória alocada no armazenamento das sessões UDP
udp_pkt_mem_amount	total de memória alocada no armazenamento dos pacotes UDP
tcp_session_mem_amount	total de memória alocada no armazenamento das sessões TCP
tcp_pkt_mem_amount	total de memória alocada no armazenamento dos

continua na próxima página

TABELA B.1: (continuação)

Contador	*Descrição*
	pacotes TCP
tcp_log_node_mem_amount	total de memória alocada no armazenamento dos nós da árvore de *logs* TCP
tcp_log_pkt_mem_amount	total de memória alocada no armazenamento dos pacotes de *logs* TCP

B.2 EXEMPLO DE UM ARQUIVO DE CONFIGURAÇÃO

A moldura abaixo apresenta um exemplo do arquivo de configuração do *RECON*.

```
#### INICIO DO ARQUIVO ####
#
# RECON.CONF v.1 c 2002/06/17
#

# Paths to Data, Temp and Output Directories
DATA_DIR = /path_to_my_raw_data_directory
TEMP_DIR = /tmp
OUTPUT_DIR = /path_to_my_output_directory

# Organization Network and Mask Addresses
NETWORK = xxx.xxx.xxx.xxx
NETMASK = xxx.xxx.xxx.xxx

# Determines if Scan Checks will be made
#FIND_HOST_SCAN = YES
FIND_HOST_SCAN_WITH_BANNER = YES

# Scan Threshholds
IN_HOST_SCAN_THRESHOLD  = 10
OUT_HOST_SCAN_THRESHOLD = 10
FTP_BANNER_PKTS_FROM_SERVER = 2
SSH_BANNER_PKTS_FROM_SERVER = 1

# ICMP/Echo Investigation
DETECT_COVERT_CHANNELS = YES
COVERT_CHANNEL_REQ_REPLY_DIFF = 1

# TCP Backdoor Detection
DETECT_BACKDOORS = YES
BACKDOOR_TIME_TOLERANCE = 1

# end.
#### FIM DO ARQUIVO ####
```

B.3 LISTAGEM DESCRITIVA DOS ARGUMENTOS DO *RECON*

A moldura abaixo apresenta os argumentos, a serem passados para o *RECON*, no momento de sua execução. Esta saída foi obtida através da execução do *RECON* com o argumento **-h**.

```
Usage: recon [-AaEeFfhIiLlOSTtUuv] [-c config_file] [-C #]
             <-r dump_file | -d yyyymmddhh:yyyymmddhh> [ -R filter_file | expression ]

   -A: print all sessions, including packets
   -a: print all sessions
   -C: stop reading after receiving # packets
   -c: specifies an alternative configuration file
   -d: selectt dump_files for reading, according to date of capture
       (comming soon)
   -E: print ICMP error packets
   -e: print ICMP error packets summary
   -F: print fragment table
   -f: print just fragment table alerts
   -h: show this help information
   -I: print ICMP sessions, including packets
   -i: print ICMP sessions
   -L: print TCP log packets
   -l: print collapsed TCP log packets
   -O: don't run packet-matching code optimizer
   -R: use filter_file as input for filter expression
   -r: read packets from dump_file
   -S: don't print final statistics
   -T: print TCP sessions, including packets
   -t: print TCP sessions
   -U: print UDP sessions, including packets
   -u: print UDP sessions
   -v: be verbose (level 1)
   -vv: be even more verbose (level 2)
```