UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Ferramenta de Injeção de Falhas para Avaliação de Segurança

por

PAULO CÉSAR HERRMANN WANNER

Dissertação submetida à avaliação,
como requisito para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Raul Fernando Weber
Orientador

Porto Alegre, junho de 2003

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Wanner, Paulo César Herrmann

Ferramenta de Injeção de Falhas para Avaliação de Segurança em Rede / por Paulo César Herrmann Wanner. - Porto Alegre: PPGC da UFRGS, 2003.

62f. : il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2003. Orientador: Weber, Raul Fernando.

1. Avaliação de Segurança. 2. Injeção de Falhas. 3. Segurança de Redes. I. Weber, Raul Fernando. II. Título.

Agradecimentos

Todas as pessoas que passaram pela minha vida com certeza deixaram alguma influência em mim e de certa forma contribuíram para a realização deste trabalho e a elas agradeço. Infelizmente, muitas são as pessoas que entram em nossas vidas por acaso, mas nem todas continuam, e as que permanecem não o fazem por acaso. Estas, com certeza, são especiais. E, por isso, quero agradecer em especial:

À minha família, mãe, Larissa e Tante: vocês são muito importantes para mim.

Aos meus amigos desde sempre: Biti, Jaime, Mim, Teco e família, por todas as alegrias e "indiadas".

Aos meus amigos: Enrique, Lomando, Lucas, Pinto, Fernanda e Quel pela amizade e pelo carinho.

Aos amigos que fiz durante todos esses anos aqui na UFRGS: todos os barra 96 e agregados, o pessoal do GSeg, do Labcom, de Belém e demais funcionários e professores.

E, claro, ao meu orientador, Raul Fernando Weber, que entre uma Triffia e outra me "agüentou" e ajudou na realização deste trabalho.

Sumário

Lista de Abreviaturas

API	Application Program Interface
ARP	Address Resolution Protocol
CERT/CC	Computer Emergency Response Team / Coordination Center
CIDF	Common Intrusion Detection Framework
CRC	Cyclic Redundancy Check
DNS	Domain Name System
FTP	File Transfer Protocol
GTK	Gimp ToolKit
HTTP	Hyper-Text Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
LLC	Logical Link Control
MTBF	Mean Time Between Failure
MTTR	Mean Time to Repair
SAINT	Security Administrator's Integrated Network Tool
SARA	Security Auditor's Research Assistant
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Lista de Figuras

Lista de Tabelas

Resumo

Nos últimos anos, com a crescente popularização das redes de computadores baseadas no protocolo IP, desde pequenas redes até redes metropolitanas começaram a se agrupar e a fazer parte do que hoje se conhece como a rede mundial de computadores. Apesar dos benefícios de comunicação e troca de informação da Internet, esse fenômeno global também trouxe problemas de segurança, pois a origem e estrutura dos protocolos utilizados na comunicação entre as diversas máquinas limitam as possibilidades de prevenir, identificar ou detectar possíveis ataques ou intrusos.

Assim, várias ferramentas surgiram para prevenir e auxiliar na tarefa de identificar problemas de segurança nas redes como *firewalls*, *sniffers* e sistemas de detecção de intrusão. Apesar dos benefícios trazidos por essas novas tecnologias, surgiram muitas dúvidas referentes a segurança que esses recursos proporcionam. Afinal, o desenvolvimento e validação desses sistemas são procedimentos bastante complexos e, freqüentemente, esses sistemas têm se tornado o alvo primário de um atacante. O resultado disso, não raramente, é uma falsa noção de segurança devido à utilização inadequada desses mecanismos, o que é, normalmente, mais prejudicial do que a simples inexistência de segurança em uma organização, mas cujas falhas são conhecidas por seus administradores.

A realização de testes para verificação da segurança de uma organização é uma atividade fundamental a fim de alcançar um ambiente operacional seguro e de verificar a correta aplicação dos requisitos de segurança de uma organização. O uso de testes permite a uma empresa verificar com precisão a postura de segurança utilizada em seus sistemas ao mesmo tempo em que permite visualizar a sua rede da mesma maneira que um atacante a visualizaria. Ao visualizar a rede como atacante, pode-se verificar com que facilidade obtém-se informações da rede, quais suas fragilidades e a dificuldade que se tem para invadi-la. Assim, obtém-se uma visão mais realista da segurança de uma organização.

Além de técnicas para a avaliação de segurança, é muito importante que se possua ferramentas para a realização desses testes. Assim, é possível automatizar a realização de testes e verificar com maior facilidade a existência de problemas em uma rede. A existência de ferramentas que testem sistemas de segurança é extremamente importante e necessária, pois, afinal, a segurança de toda uma rede pode depender fortemente de algum desses sistemas.

Este trabalho apresenta as técnicas existentes para a injeção de falhas visando verificar as que são mais eficientes para a avaliação de sistemas de segurança em rede. Adicionalmente, são apresentadas algumas técnicas para o teste de mecanismos de segurança e algumas ferramentas existentes para a realização de tais testes. A partir desses estudos, é apresentado um modelo de ferramenta adequando as funções de um sistema de injeção de falhas ao teste de mecanismos de segurança em rede.

Um protótipo dessa ferramenta foi desenvolvido e é apresentado neste trabalho. Esse protótipo permite o envio e o recebimento de pacotes da pilha TCP/IP, podendo testar problemas em protocolos e sistemas utilizados na rede. E, através da utilização de *plug-ins*, permite que diversos tipos de ataque mais sofisticados possam ser realizados.

Palavras-Chave: avaliação de segurança, injeção de falhas e segurança de redes.

TITLE: "FAULT INJECTOR TOOL FOR NETWORK SECURITY EVALUATION"

Abstract

In the last years, with the increasing use of the computer networks based on the IP protocol, small networks and metropolitan networks were joined and became part of what today it is known as the world-wide computer network. Despite the benefits of communication and information exchange through the Internet, this global phenomenon also brought some security problems. The origin and structure of the protocols used in the communication between the diverse computers limit the possibilities to prevent, identify or even detect possible attacks or intruders.

Thus, some tools were developed to prevent attacks and to assist in the task to identify security problems in networks as firewalls, sniffers and intrusion detection systems. Despite the benefits brought with these new technologies, some doubts were raised about the security that these resources provide. After all, the development and validation of these systems are sufficiently complex and, frequently, these systems become the primary target of an intruder. The result of this is, not rare, a false notion of security due to the inadequate use of these equipment. This inadequacy is, usually, more harmful than the simple inexistence of security in an organization, but whose unreliability is known by its administrators.

Carrying out tests for security verification in an organization is a basic activity in order to reach a safe operational environment and to verify the correct application of the security requirements. The use of tests allows a company to verify with precision the security policies used in its systems. At the same time it allows visualizing its network in the same way that an intruder would visualize it. Visualizing the network as an intruder, we are able to verify the ease of getting information of the network, its fragilities and the difficulty to invade it. Thus, a more realistic vision of the security of an organization is obtained.

Besides the security evaluation techniques, it is very important to have tools for carrying out these tests. Thus, it is possible to automate the test execution and to easily verify the existence of problems in a network. The existence of tools that test security of systems is extremely important and necessary, because the security of the whole network can be strongly dependent on some of these systems.

This work presents the existing fault injection techniques, aiming to verify the ones that are more efficient for the evaluation of network security systems. Additionally, some security mechanisms testing techniques and some tools for performing such tests are presented. After these studies, a model of a specific tool to evaluate network security systems is presented tuning up the functions of a fault injection system to the testing of security mechanisms.

A prototype of this tool was developed and is presented in this work. This prototype allows to send and receive TCP/IP packets, enabling to test protocols and systems used in a network. And, through the use of plug-ins, allows that several sophisticated attacks can be executed.

Keywords: security evaluation, fault injection and network security.

1 Introdução

A segurança de redes é uma questão atual à medida que novas tecnologias e serviços são disponibilizados e suas vulnerabilidades são descobertas. O número de vulnerabilidades encontradas vem crescendo a cada ano e elas são normalmente o principal meio de ataque utilizados por intrusos, quando não corrigidas por empresas ou organizações. A tabela a seguir demonstra a elevação do número de incidentes e vulnerabilidades registrados nos últimos anos segundo dados do CERT/CC [CER 88].

TABELA 1.1 - Aumento do Número de Incidentes e Vulnerabilidades Relatados ao CERT/CC

Ano	**1995**	**1996**	**1997**	**1998**	**1999**	**2000**	**2001**	**2002**
Incidentes	2412	2573	2134	3734	9859	21756	52658	82094
Vulnerabilidades	171	345	311	262	417	1090	2437	4129

A existência de conexões entre a rede de computadores de uma instituição com a Internet põe em risco os seus dados, os seus recursos computacionais e a sua reputação [CHA 2000]. Os dados de uma organização que circulam em sua rede devem ser protegidos quanto ao seu sigilo, integridade e disponibilidade, pois podem ser dados sensíveis. Mesmo que seus dados não sejam considerados valiosos, os recursos computacionais de uma rede são fundamentais para o bom andamento e funcionamento de uma organização. Além disso, qualquer invasão da rede de uma empresa pode ter conseqüências sérias para sua reputação, ainda mais se seus dados forem importantes ou se seus serviços basearem-se na Internet.

Por isso, uma organização ou empresa que disponibiliza serviços na Internet ou que possua algum tipo de acesso à mesma, deve se preocupar com a necessidade de segurança em sua rede. E, para tanto, é preciso a adoção de políticas de segurança, de mecanismos de segurança que as implementem e uma avaliação contínua da segurança fornecida por ambos, adequando-os conforme necessário.

A política de segurança tem por objetivo estabelecer diretrizes claras sobre a postura que deve ser adotada quanto à segurança da informação apresentando definições, metas e escopos de atuação [ABN 2001]. Contudo a existência de uma política de segurança não é de nenhuma valia caso não existam mecanismos de segurança que a implementem como *firewalls*, sistemas de detecção de intrusão (IDS), comunicação segura ou caso esses mecanismos não forneçam uma segurança efetiva. Assim, a avaliação da segurança computacional de uma empresa ou organização é uma tarefa bastante complexa que deve ser executada continuamente a fim de verificar as deficiências e problemas de sua segurança e envolve todo um processo de gerenciamento de riscos e testes para avaliação de segurança.

O principal objetivo de um gerenciamento de risco deve ser o de proteger uma organização e suas habilidades a fim de atingir sua missão como empresa [STO 2001]. Portanto, o gerenciamento de risco destina-se a toda uma organização, não se restringindo a sua área tecnológica. Contudo, neste trabalho, é dado um enfoque à área computacional. Considerando essa abordagem, o risco é o impacto negativo da exploração de uma vulnerabilidade, considerando sua probabilidade e as conseqüências dela ocorrer. E gerenciamento de risco é o processo de identificar, qualificar e reduzir (a níveis aceitáveis) esses riscos. O gerenciamento de risco pode ser dividido em três etapas [STO 2001], figura 1.1:

- classificação de risco (*Risk Assessment*): consiste na identificação e avaliação do risco e das suas conseqüências;
- atenuação de risco (*Risk Mitigation*): consiste em solucionar ou atenuar os riscos encontrados na etapa anterior;
- monitoração e controle de risco (*Evaluation and Assessment*): consiste em avaliar as soluções adotadas e requalificar o risco existente.

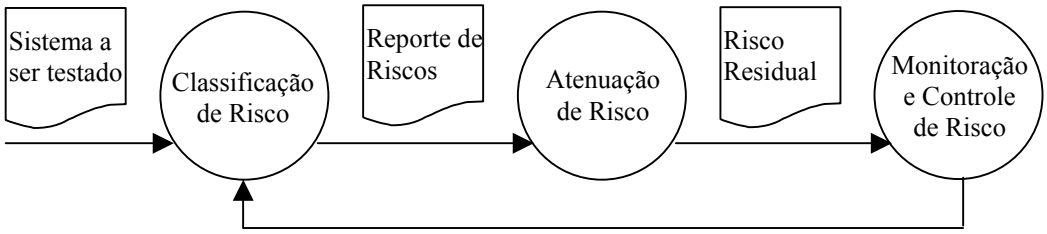

FIGURA 1.1 - Gerenciamento de Risco

1.1 Classificação de Risco

O risco presente em um sistema computacional é uma função da probabilidade de alguma fonte de ameaça explorar uma particular vulnerabilidade e seus impactos negativos para uma organização ou empresa [STO 2001]. Fonte de ameaça é um termo genérico que pode referenciar um *hacker*, uma empresa rival, um funcionário da organização ou mesmo um fenômeno natural que tenha acesso a uma vulnerabilidade, enquanto que a probabilidade de ameaça de uma vulnerabilidade é calculada em função da vulnerabilidade e dos mecanismos de controles existentes em um sistema. O impacto, por sua vez, refere-se à magnitude do dano que pode ser produzido através da exploração de uma vulnerabilidade.

A etapa de classificação de risco visa a determinar e qualificar os riscos existentes nos sistemas de uma organização e propor soluções para os problemas encontrados. A classificação de riscos pode ser subdividida segundo Stonebumer et al. [STO 2001] em:

- caracterização do sistema: consiste na coleta de dados sobre o sistema, configurações, usuários, dados manipulados, arquitetura, políticas, etc;
- identificação de ameaças: consiste em identificar fontes de ameaça que possam explorar uma vulnerabilidade. Fontes normais de ameaça são: *hackers, crackers*, funcionários, terroristas, espiões industriais, países estrangeiros;
- identificação de vulnerabilidades: consiste em identificar vulnerabilidades de sistemas que podem vir a ser exploradas por atacantes;
- análise de mecanismos de controle: consiste em analisar os mecanismos de controle implementados ou planejados a fim de minimizar ou eliminar a possibilidade de um ataque;
- determinação da probabilidade de ameaça: consiste em calcular a taxa de probabilidade de uma vulnerabilidade ser explorada por um atacante. Os fatores que influem nesta taxa são as fontes de ameaça, a natureza da vulnerabilidade e os mecanismos de controle;
- análise de impacto: consiste na análise das possíveis conseqüências na ocorrência de um ataque. A análise de impacto pode ser medida a partir da perda ou degradação da integridade, disponibilidade e confidencialidade de dados ou serviços;

- determinação do risco: consiste em determinar o grau de risco de um sistema. Deve-se levar em conta a probabilidade da ameaça, o impacto de um ataque e os mecanismos de controle existentes;
- soluções recomendadas: consiste em listar e recomendar possíveis soluções para reduzir o nível de risco encontrado nos sistemas.
- documentação dos resultados: consiste na elaboração de uma documentação para análise do pessoal responsável pela segurança computacional da organização ou empresa.

1.2 Atenuação de Risco

A segunda etapa do gerenciamento de risco envolve o estudo, a avaliação e a implementação dos mecanismos de controle sugeridos a partir do processo de classificação de risco. A eliminação total de um risco é impraticável ou quase impossível, por isso deve-se analisar as opções existentes a fim de utilizar a que permite obter uma maior cobertura do risco a um menor custo possível, assim diminuindo o nível de risco de um sistema a um nível aceitável com um menor impacto adverso a uma organização.

Existem diversas opções para se lidar com o risco de um sistema. Pode-se simplesmente aceitar o risco, neste caso o custo de implementação de uma solução muitas vezes é superior ao impacto de um ataque, ou evitar propriamente o risco, eliminando-se a causa ou suas conseqüências. Outra solução é limitar o risco, utilizando-se mecanismos de controle que minimizem o impacto, a propagação ou as conseqüências de um ataque ou ainda transferir o risco, ou seja, utilizar mecanismos que compensem as perdas de um ataque, como, por exemplo, um seguro.

Uma boa estratégia para verificar se ações para atenuação de risco devem ser tomadas é mostrada na figura 1.2. Pode-se observar que riscos somente existem a partir da combinação de ameaças e vulnerabilidades e que a existência de risco não implica, necessariamente, a tomada de medidas para contenção do mesmo.

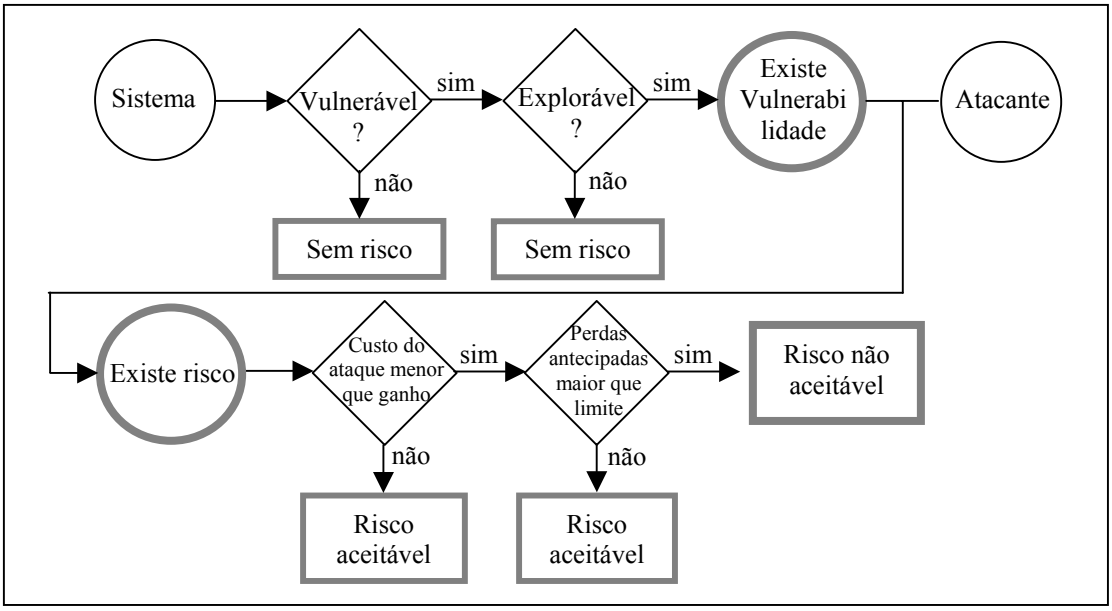

FIGURA 1.2 - Necessidade de Atenuação de Risco [STO 2001]

Independente das medidas de controle adicionadas visando atenuar os riscos de um sistema, sejam elas para reduzir o número de vulnerabilidades, adicionar mecanismos de controle, ou reduzir as conseqüências de um ataque, sempre tem-se um risco residual resultante das medidas adotadas. Esse risco residual é o objetivo da próxima etapa do gerenciamento de risco, a monitoração e controle de risco.

1.3 Monitoração e Controle de Risco

A maioria das redes, independentemente de organizações, sempre está em mudança, seja pela modificação de sua topologia, atualização de *softwares* e sistemas ou pela instalação de novos serviços. Além disso, os próprios funcionários que utilizam e mantêm a rede podem mudar durante os anos. Assim, as ameaças, as vulnerabilidades e os próprios mecanismos de segurança modificam-se durante o tempo.

A monitoração e controle dos riscos de um sistema e de uma organização ou empresa deve ser contínua a fim de evitar que mecanismos de controle se tornem obsoletos e que novos riscos surjam. Por isso, a realização de testes de segurança deve ser rotineira para a avaliação da segurança de uma rede ou organização.

1.4 Organização da Dissertação

A realização de testes é fundamental para a avaliação de segurança e para o processo de gerência de riscos de uma organização. Os testes de segurança visam, principalmente, identificar vulnerabilidades, analisar mecanismos de controle, determinar a probabilidade de ameaças e analisar o impacto de possíveis ataques. Uma das técnicas mais utilizadas para a realização de tais testes é a injeção de falhas (ver capítulo 3), uma vez que visa a acelerar possíveis falhas e estudar suas ocorrências e conseqüências no sistema testado.

Assim, para que o processo de gerenciamento de riscos seja efetivo, deve-se possuir uma idéia bem clara de quais são os métodos existentes para a injeção de falhas e quais desses aplicam-se ao objeto que se deseja testar. Adicionalmente, deve-se saber os tipos de testes que podem ser feitos e quais devem ser implementados para se verificar uma determinada falha e, finalmente, que ferramentas existentes podem ser utilizadas para a realização de tais testes.

Este trabalho visa a facilitar o processo de gerenciamento de riscos através da especificação de uma ferramenta desenvolvida para avaliação de mecanismos de segurança em rede. Essa ferramenta é projetada para otimizar a implementação de testes de segurança em rede e, conseqüentemente, agilizar as fases de avaliação e classificação de risco e monitoração e controle de risco desses sistemas.

Inicialmente é realizado um estudo das técnicas existentes para injeção de falhas, objetivando adotar aquela que mais se enquadra ao teste de equipamentos de segurança em rede, capítulo 2. Após, são vistas as técnicas existentes para a avaliação de segurança de rede, capítulo 3. E, no capítulo 4, é feita uma análise de ferramentas para a injeção de falhas em rede, visando verificar suas características, vantagens e desvantagens. A partir desses estudos e análises, é proposto um modelo de ferramenta para a injeção de falhas, capítulo 5, e é descrita a implementação de um protótipo desse modelo, capítulo 6.

2 Técnicas de Injeção de Falhas

Este capítulo visa a estudar as técnicas existentes para injeção de falhas segundo uma abordagem da área de tolerância a falhas. Inicialmente, são vistos alguns conceitos de tolerância a falhas e as diversas técnicas existentes para a avaliação de um sistema tolerante a falhas durante o seu ciclo de vida. E, posteriormente, são vistos os elementos que compõem um sistema injetor de falhas e os métodos que podem ser utilizados para a injeção de falhas: *hardware* e/ou *software*. Uma comparação dos métodos estudados é apresentada no final do capítulo.

O objetivo de tolerância a falhas é alcançar dependabilidade, que indica a qualidade de serviço fornecido por um dado sistema e a confiança depositada nele. As principais medidas de dependabilidade segundo Laprie [LAP 98] são:

- disponibilidade (*availability*): probabilidade de o sistema estar operacional em um instante de tempo determinado. É medido pelo tempo de funcionamento;
- confiabilidade (*reliability*): capacidade de atender as especificações, dentro de condições definidas durante um certo período de funcionamento. É medido pelo MTBF (*Mean Time Between Failure*);
- seguro (*safety*): o sistema, na ocorrência de falhas, interrompe o seu funcionamento de forma a não provocar danos a outros sistemas ou pessoas;
- confidencialidade (*confidentiality*): capacidade de evitar o acesso não autorizado a informações mantidas pelo sistema;
- integridade (*integrity*): capacidade do sistema em evitar a alteração indevida de dados;
- manutenção (*maintainability*): facilidade de realizar manutenções sem ser necessário para tanto paradas muito grandes do sistema. É medido pelo MTTR (*Mean Time To Repair*).

Assim, sistemas tolerantes a falhas são aqueles que, mesmo durante a ocorrência de falhas, mantêm seu funcionamento normal devido ao uso de técnicas de tolerância a falhas. Essas técnicas garantem o funcionamento correto do sistema mesmo na ocorrência de falhas e são todas baseadas em redundância, exigindo componentes adicionais ou algoritmos especiais. As técnicas de tolerância de falhas são basicamente duas:

- detecção, localização, reconfiguração e tratamento;
- mascaramento.

Esses sistemas utilizam-se de metodologias específicas para o seu desenvolvimento uma vez que necessitam manter o seu correto funcionamento mesmo na presença de falhas. Contudo o sucesso de tais métodos depende, em grande parte, de uma correta validação da metodologia de desenvolvimento empregada. A avaliação da dependabilidade de um sistema envolve o estudo de falhas e erros. O mais importante nesse estudo é o tempo de latência entre a ocorrência da falha e o surgimento do erro, conhecido como latência de falha. Como o tempo de latência pode variar muito entre a ocorrência da falha e a manifestação do erro, é muito difícil recriar a situação que originou o erro. Assim, a fim de identificar e entender potenciais falhas, utilizam-se duas abordagens: uma experimental e outra analítica.

Na abordagem experimental, deve-se conhecer os mecanismos de tolerância a falhas e os métodos de detecção e recuperação de erros do sistema a ser testado. Para

tanto é necessário o uso de ferramentas específicas para injeção e monitoração de falhas. A avaliação experimental de um sistema pode ser realizada durante as diversas fases do seu ciclo de vida.

Durante a fase de definição e projeto, quando o sistema ainda está em estudo e não se possui nenhum detalhe de implementação, é comum o uso de simulação. A injeção de falhas via simulação permite testar as concepções adotadas e os mecanismos de tolerância adotados, provendo um bom *feedback* para os desenvolvedores. Contudo um dos problemas da simulação é que seus dados se baseiam em parâmetros de entradas que especificam o sistema em si e a distribuição de falhas e erros no modelo definido. Assim, a qualidade dos valores de saída dependerá da precisão dos dados de entrada, os quais normalmente são difíceis de se obter, pois mudanças no projeto e na tecnologia utilizada complicam o uso de medidas anteriores.

O teste de protótipos, que utiliza uma abordagem experimental, de outro modo, permite uma melhor avaliação do sistema e de suas características. Uma vez que não se baseia em concepções, os resultados obtidos durante a fase de protótipo são mais precisos. Nessa fase, a injeção de falhas visa a:

- identificar gargalos de dependabilidade;
- estudar o funcionamento do sistema na presença de falhas;
- determinar a cobertura dos mecanismos de detecção e recuperação de erros;
- avaliar os mecanismos de tolerância a falhas quanto a sua eficiência, performance, etc.

A injeção de falhas baseada em protótipos pode ser feita através do nível de *hardware* (falhas lógicas ou elétricas) ou do nível de *software* (código ou dados corrompidos). Apesar de permitir um grande número de informações, essa técnica somente permite a obtenção de dados de falhas simuladas. Ela não oferece nenhuma medida de dependabilidade sobre o tempo médio entre falhas (MTBF) ou sobre a disponibilidade do sistema.

Durante a fase operacional, pode-se utilizar tanto a abordagem experimental, através da injeção de falhas, como a abordagem analítica. Uma abordagem analítica consiste na análise de dados coletados durante a operação normal do sistema que possuem grande quantidade de informação sobre a ocorrência de erros, defeitos e possíveis tentativas de recuperação. A análise desses dados pode prover um maior conhecimento das características de erros e defeitos atuais e pistas sobre um possível modelo analítico. Contudo essa análise baseada em medidas, apesar de ser útil na avaliação de sistemas reais, é limitada a erros detectados. Além disso, os dados devem ser coletados durante um longo período, visto que a ocorrência de erros e defeitos não é freqüente e as condições do ambiente podem variar drasticamente, provocando dúvidas na validade estatística do resultado.

Como pôde ser observado, tanto a abordagem experimental quanto a abordagem analítica possuem suas limitações e somente com a utilização ambas é possível uma análise precisa da dependabilidade de um sistema. O uso de técnicas de injeção de falhas visa a testar os mecanismos de detecção, isolamento, reconfiguração e recuperação de falhas que um sistema tolerante a falhas utiliza. Um ambiente para testes de injeção de falhas, como pode ser visto na figura 2.1, consiste basicamente em [HSU 97]:

- um alvo (sistema a ser testado);
- um injetor de falhas;
- uma biblioteca de falhas;

- um gerador de *workload*;
- uma biblioteca de *workload*;
- um controlador;
- um monitor;
- um coletor de dados;
- um analisador de dados.

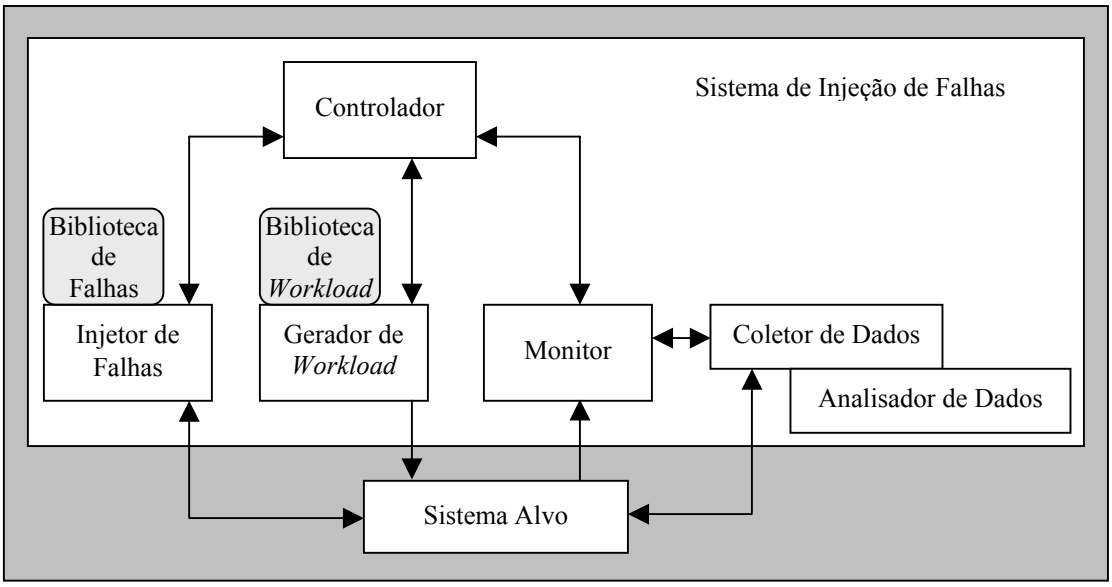

FIGURA 2.1 - Sistema de Injeção de Falhas [HSU 97]

O injetor de falhas tem como função injetar falhas no sistema sendo testado, enquanto este executa as tarefas do gerador de *workloads* (aplicações, *benchmarks*). O monitor observa toda a execução do sistema alvo e inicia o coletor de dados sempre que for necessário armazenar alguma informação. A análise dos dados pode ser feita tanto *on-line* como *off-line* através do analisador de dados. Todos estes elementos são gerenciados por um único componente: o controlador.

Fisicamente, o controlador pode executar tanto no sistema alvo como em um computador separado. O injetor de falhas pode ser desenvolvido em *hardware* ou em *software* e pode suportar falhas de tipo, localização, tempo, semântica e estrutura. As diversas falhas são configuradas através de bibliotecas, assim permitindo uma maior flexibilidade e portabilidade. Os demais componentes podem ser implementados da mesma maneira.

De modo geral, existe a injeção de falhas por *hardware* e a por *software*. A escolha do método e do modo de implementação de injeção de falhas baseia-se no tipo de falha que se deseja e do esforço necessário para criá-la. Por exemplo, a injeção de falhas permanentes usando-se *software* pode gerar muito *overhead* e muitas vezes é até impraticável, dependendo da falha. Contudo se o objetivo é inserir falhas nos dados, a injeção por *software* é suficiente. Nos casos em que a falha pode ser injetada por ambos os métodos, características como custo, precisão, grau de intrusão e repetição podem influenciar na escolha.

2.1 Injeção de Falhas por *Hardware*

A injeção de falhas implementada por *hardware* utiliza-se de um *hardware* adicional para introduzir as falhas no sistema alvo. Dependendo da falha e da

localização da mesma, os métodos de injeção de falhas por *hardware* classificam-se em duas categorias:

- por contato: o injetor possui contato físico direto com o sistema a ser testado;
- sem contato: não existe contato direto entre o injetor e o sistema alvo.

Esse método serve para estudar características de dependabilidade que necessitem de uma grande precisão de tempo; por exemplo, latência de falhas em CPU, ou que necessitem de acesso a locais pouco alcançáveis através de outros métodos.

Os métodos de injeção por contato são provavelmente os mais comuns entre os métodos de injeção por *hardware*. As duas técnicas principais para alteração de características elétricas e voltagem nos pinos são:

- *active probes*: utilizado para gerar falhas do tipo *stuck-at*;
- *socket insertion*: este método insere um *socket* entre o *hardware* do alvo e a sua placa de circuitos. Pode gerar falhas mais complexas de lógica como: inversão de sinais, *and* de sinais ou *or* de sinais.

Ambos os métodos provêm um bom controle de tempo e localização das falhas com pouca influência sobre o sistema testado.

Já os métodos de injeção sem contato visam reproduzir erros causados por fenômenos naturais como radiações iônicas ou campos magnéticos e, por isso, também são muito utilizados. Contudo esse tipo de falha é difícil de controlar, pois não se pode precisar o tempo nem a localização da falha criada pelo campo magnético ou pela emissão de radiação iônica.

2.2 Injeção de Falhas por *Software*

Enquanto a injeção de falhas por *hardware* necessita de circuitos e equipamentos adicionais para gerar falhas, a injeção de falhas por *software* é mais barata e fácil de controlar. Além disso, a abordagem por *software* permite simular falhas tanto em *hardware* como em aplicativos e sistemas operacionais.

TABELA 2.1 - Técnicas de Injeção de Falhas por Software

Técnicas Utilizadas para Injeção de Falhas por Software	
Tipo	**Método**
Falha de Software	Modifica o segmento de código do programa
Erro de Software	Modifica o segmento de dados do programa
Falha de Memória	Altera bits de memória
Falha de CPU	Utiliza uma *trap* para modificar a área de memória salva pelos registradores da CPU
Falha de Bus	Utiliza *traps* antes e depois de instruções para modificar código e dados usados pela instrução restaurando-os após a execução da mesma
Falhas de Rede	Modifica ou apaga mensagens de transmissão

Fonte: PRADHAN. Fault-Tolerant Computer System Design. p. 334

Diversas técnicas foram desenvolvidas para gerar diferentes erros de *hardware* e *software* por injeção de falha por *software* como pode ser visto na tabela 2.1. Apesar da flexibilidade existente na abordagem por software, existem algumas desvantagens:

- não se pode injetar falhas em áreas inacessíveis via *software*;
- a injeção por *software* muitas vezes pode modificar o modo de execução normal ou até mesmo alterar as estruturas do sistema a ser testado;
- pouca precisão temporal pode dificultar a monitoração dos erros. Para falhas de latência, como falhas de CPU, esta abordagem pode não ser a mais indicada, pois pode falhar na captura de certos comportamentos gerados pelo erro, como sua propagação. Nesses casos, o uso de uma abordagem híbrida, combinando injeção por *software* e monitoração por *hardware*, pode solucionar o problema.

Pode-se classificar os métodos de injeção de falhas por *software* com base no momento da injeção: em tempo de compilação ou em tempo de execução. A injeção em tempo de compilação baseia-se em alterar o código fonte ou *assembler* do aplicativo a ser testado a fim de emular falhas de *hardware*, *software* ou falhas de comunicação. Assim, ao se executar as instruções modificadas do programa, causa-se a injeção de falhas. Além disso, a injeção em tempo de compilação não causa perturbações no sistema alvo durante a execução, já que a injeção de falhas faz parte do sistema. Sua implementação é bem simples, desde que se possua o código fonte do programa que se deseja testar.

Na injeção em tempo de execução é necessário um mecanismo adicional para ativar a injeção de falhas. Os mecanismos utilizados são [HSU 97]:

- *time-out*: é a técnica mais simples. Um relógio é utilizado para gerar uma interrupção num determinado tempo e executar a injeção. Nenhuma modificação é necessária na aplicação ou no *workload*. É bastante utilizado para simulação de falhas transitórias e intermitentes de *hardware*;
- *exception/Trap*: neste caso, exceções de *hardware* ou *software* transferem o controle da execução para o injetor de falhas. Diferente da técnica anterior, pode-se injetar falhas quando certas condições ocorrerem. Para tanto, instruções devem ser inseridas no código fonte para invocar o injetor de falhas;
- *code insertion*: instruções são adicionadas no programa alvo, permitindo a injeção de falhas ocorrer antes de determinados comandos. Diferente do método *exception/trap*, o injetor de falhas deve existir como parte do sistema alvo e executar em modo de usuário em vez de em modo de sistema.

2.3 Comparação dos Métodos de Injeção de Falhas

Existem diversas abordagens e métodos para injeção de falhas e ainda diferentes modos de utilização que permitem obter características distintas quanto à abrangência, eficiência e custo da injeção de falhas. A determinação da técnica a utilizar depende do sistema a ser testado e dos objetivos específicos envolvidos no teste. Uma comparação dos métodos de injeção de falhas descritos pode ser vista na tabela 2.2.

TABELA 2.2 - Comparação dos métodos de injeção de falhas

Classificação dos Métodos de Injeção de Falhas				
	Hardware		**Software**	
	por contato	**sem contato**	**compilação**	**execução**
Custo	Alto	Alto	Baixo	Baixo
Perturbação	Nenhum	Nenhum	Baixo	Alto
Risco de danos	Alto	Baixo	Nenhum	Nenhum
Precisão de monitoração	Alto	Alto	Alto	Baixo
Acessibilidade de injeção de falhas	*Chip pin*	*Chip internal*	Registro de memória do software	Registro de memória I/O controlador/porta
Controlabilidade	Alto	Baixo	Alto	Alto
Ativação	Sim	Não	Sim	Sim
Repetição	Alto	Baixo	Alto	Alto

Fonte: HSUEH. Fault Injection Techniques and Tools. p. 07

Dessa forma, abordagens analíticas ou técnicas de injeção de falhas por *hardware* possuem pouca ou nenhuma utilidade para o teste de sistemas de segurança em rede, visto que a maioria das análises é feita por *software* com base nos dados obtidos na rede através da monitoração de comunicações e protocolos. Assim, das técnicas de injeção de falhas vistas anteriormente, apenas duas podem ser mais amplamente utilizadas no teste de sistemas de segurança de rede. São elas a simulação e a injeção de falhas por *software*.

A simulação pode ser utilizada pelo desenvolvedor durante a fase de projeto do sistema a fim de testar as abordagens e os mecanismos de segurança especificados, enquanto a injeção de *software* pode ser utilizada tanto pelo desenvolvedor do *software* para teste de protótipos como pelo usuário final do sistema a fim de verificar a real eficácia do sistema de segurança. Portanto, devido às necessidades de verificação do sistema em tempo de execução, nesse trabalho é utilizada a injeção de falhas por *software*, visando gerar falhas na rede e, conseqüentemente, em seus sistemas de segurança através da modificação, inserção e deleção de mensagens.

Entretanto não é qualquer tipo de falha que se pretende injetar. Uma vez que se está avaliando mecanismos de segurança, visa-se injetar apenas falhas maliciosas a fim de verificar a prevenção de falhas, detecção de erros e o confinamento de erro implementado por esses mecanismos.

Falhas são inerentes aos sistemas e podem representar erros de configuração de um equipamento, erros de projeto em protocolos, entre outros. Assim, mecanismos de segurança em redes como *firewalls* visam prevenir que falhas maliciosas entrem na rede de uma organização. Outros mecanismos, como sistemas de detecção de intrusão, visam a detectar a existência de algum erro de segurança como a ocorrência de um ataque, enquanto que mecanismos de controle de integridade visam a confinar possíveis erros evitando que se propaguem ou se tornem um problema maior, o que resultaria em um desvio da política de segurança, figura 2.2.

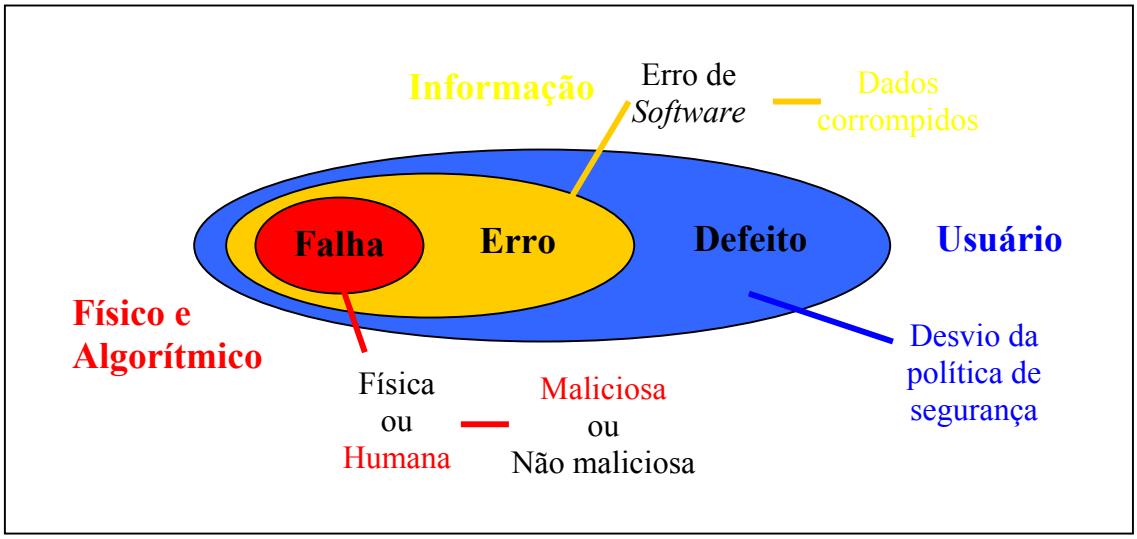

FIGURA 2.2 - Injeção de Falhas Maliciosas

3 Injeção de Falhas em Sistemas de Segurança de Rede

A área de segurança pode ser vista como relacionada com a área de tolerância a falhas no sentido que investiga apenas os defeitos causados por falhas maliciosas. Por isso, a injeção de falhas na área de segurança não visa a testar todas as características de dependabilidade de um sistema. As características que se desejam testar são: disponibilidade, confidencialidade e integridade.

A disponibilidade é medida pelo tempo total de funcionamento do sistema, mas no caso de segurança deseja-se saber se o sistema é capaz de resistir a ataques e manter o atendimento aos serviços prestados mesmo havendo queda de performance. A confidencialidade refere-se à segurança oferecida no acesso a dados, ou seja, os dados somente podem ser acessados pelas pessoas ou programas que tenham permissão. Além disso, deseja-se que todo e qualquer dado seja protegido contra modificações indesejáveis, ou, pelo menos, que qualquer modificação de um dado possa ser detectada.

As redes de computadores atuais são mais difíceis de serem protegidas, pois o número e a variedade de computadores existentes em uma rede tende a aumentar cada vez mais, o que aumenta em muito a complexidade para se garantir a segurança individual de cada máquina. Assim, a tendência que existe atualmente é a de aumentar a segurança das redes de computadores, controlando os dados que trafegam na rede, o acesso às diversas máquinas e aos tipos de serviços fornecidos por cada computador em uma rede, em vez de se aumentar a segurança de cada computador individualmente [CHA 2000].

Para verificar se este controle de segurança é efetivo, é necessário avaliá-lo. Neste capítulo são enumeradas diversas técnicas existentes para a avaliação de segurança de uma organização, e algumas delas, utilizadas para a avaliação de segurança em rede, são analisadas. Posteriormente, são descritos dois tipos de mecanismos de segurança em rede: *firewalls* e IDSes, apresentado-se características e métodos que podem ser utilizados para avaliá-los.

3.1 Técnicas para Avaliação de Segurança

A razão principal da realização de testes de segurança em sistemas computacionais é a de identificar vulnerabilidades e conseqüentemente repará-las. A realização de testes para verificação de segurança é uma atividade fundamental a fim de alcançar um ambiente operacional seguro e de verificar a correta aplicação dos requisitos de segurança desta organização. O uso de testes permite a uma empresa verificar com precisão a postura de segurança utilizada em seus sistemas ao mesmo tempo que permite visualizar a sua rede da mesma maneira que um atacante a visualizaria.

Pode-se dizer que essa é a grande vantagem da realização de teste de segurança em relação a outros mecanismos de verificação de segurança, como certificações e auditorias. Ao visualizar a rede conforme um atacante, pode-se verificar com que facilidade obtém-se informações da rede, quais suas fragilidades e a dificuldade que se tem para invadi-la. Assim, obtém-se uma visão mais realista da segurança de uma organização.

Existem diferentes tipos de testes de segurança que podem ser realizados. Alguns são predominantemente manuais e requerem intervenção humana para serem conduzidos, enquanto que outros são altamente automatizados e necessitam pouca

interação humana. Indiferentemente do tipo de teste de segurança, todos necessitam ser aplicados por pessoas especializadas na área de segurança para melhor acompanhar os testes e interpretar os resultados obtidos. Os tipos de testes de segurança mais comuns existentes são [WAC 2002]:

- mapeamento de rede (*Network Mapping*);
- varredura de vulnerabilidade (*Vulnerability Scanning*);
- teste de intrusão (*Penetration Testing*);
- teste e avaliação de segurança (*Security Test and Evaluation*);
- descoberta de senhas (*Password Cracking*);
- análise de *logs* (*Log Review*);
- verificação de integridade (*Integrity Checkers*);
- detecção de vírus (*Virus Detection*);
- mapeamento de modems (*War Dialing*).

Não necessariamente os testes são aplicados individualmente. Freqüentemente, vários testes são utilizados em conjunto para se obter uma maior compreensão da segurança da rede e informações necessárias para testes mais sofisticados. É importante salientar também que nenhum dos testes listados anteriormente tem a capacidade, se aplicado sozinho, de prover uma visão completa da rede testada ou de sua segurança.

Os resultados obtidos através desses testes devem ser documentados e estar disponíveis em uma organização para os profissionais envolvidos com áreas relacionadas a segurança, podendo ser utilizados como [WAC 2002]:

- referência a ações corretivas;
- método de verificação da segurança de uma organização;
- auxílio em análises de custo/benefício;
- referência para medir a evolução de uma organização.

A seguir são estudadas algumas técnicas de avaliação de segurança. Nem todas as técnicas mencionadas anteriormente são vistas, apenas se analisam aquelas pertinentes ao teste de mecanismos de segurança em rede.

3.1.1 Mapeamento de Rede

O mapeamento de rede é um teste que envolve a utilização de programas de varredura de portas para a identificação de todos os potenciais alvos, estações de trabalho ativas, conectados em uma rede [SCA 2001]. Uma vez identificado um alvo em potencial é feita uma varredura dos serviços disponibilizados pela estação, e identificada a aplicação específica sendo executada para cada serviço de rede. O resultado da varredura consiste em uma lista de todos os computadores ativos, os sistemas operacionais utilizados, os serviços disponibilizados e a aplicação que executa o serviço.

O mapeamento de rede é uma atividade bastante automatizada, mas necessita de intervenção humana para a interpretação de seus resultados. Através desse teste, pode-se controlar as estações acessáveis pela rede, os serviços sendo disponibilizados e corrigir problemas como:

- desconectar estações não autorizadas;
- desabilitar ou remover serviços desnecessários ou vulneráveis;
- restringir o acesso a serviços vulneráveis.

Existem inúmeros programas de varredura de portas e as informações que podem ser obtidas, bem como a sua precisão, pode variar conforme o programa utilizado. O mapeamento de rede é um dos primeiros passos feito por um atacante para a realização de uma invasão. Através de seus resultados, é possível verificar os potenciais alvos de uma rede, contudo a grande limitação dessas ferramentas é a de, apesar de identificarem estações, serviços, aplicações e sistemas operacionais acessáveis em uma rede, não identificar possíveis vulnerabilidades nos mesmos automaticamente.

3.1.2 Varredura de Vulnerabilidade

A varredura de vulnerabilidades, pode-se dizer, é uma evolução do mapeamento de rede, pois não apenas identifica computadores, serviços, aplicações e sistemas operacionais como também identifica vulnerabilidades já conhecidas associadas a eles automaticamente. Essa vantagem em relação ao simples mapeamento de rede, permite ao administrador de uma rede identificar antes de um atacante:

- vulnerabilidades conhecidas;
- necessidade de *patches* ou *upgrades*;
- programas com versões antigas;
- violações da política de segurança.

Por outro lado esse tipo de programa ainda possui algumas deficiências. Geralmente, são capazes de identificar e classificar as vulnerabilidades encontradas nas estações de uma rede, mas não são capazes de classificar o nível de risco da rede testada. Além disso, os programas atuais ainda apresentam um elevado índice de falsos positivos, ou seja, possuem uma alta taxa de erro, reportando vulnerabilidades inexistentes [WAC 2002].

3.1.3 Teste de Invasão

O teste de invasão é um teste de segurança em que se tenta contornar e transpassar as características de segurança de um sistema baseando-se nas falhas existentes no seu projeto e implementação. O objetivo desse teste é identificar métodos de ganhar acesso a um sistema utilizando-se ferramentas e técnicas desenvolvidas por atacantes.

Um teste de invasão pode ser desenvolvido para simular um ataque externo ou interno a rede testada. Em um teste externo de invasão, a maior dificuldade encontrada é o *firewall* [CHA 2000], já que ele limita a quantidade e tipo de tráfego que é permitido entrar na rede interna a partir da Internet. Dependendo do tipo de protocolos permitido pelo *firewall*, os ataques iniciais geralmente são focados em protocolos comumente permitidos como HTTP e FTP. O objetivo inicial desse teste é testar o próprio *firewall* ou tentar transpassá-lo a fim de ter acesso à rede interna [HAE 97].

O teste de invasão interna é bastante similar ao externo com a diferença que agora se está atrás do *firewall* e possui-se um certo nível de acesso à rede. E, a partir desse momento, está-se testando também outros sistemas de segurança como os componentes de um sistema de detecção de intrusão, por exemplo.

O teste de invasão interna pode ser uma continuação de um teste de invasão externa, ou seja, conseguiu-se burlar o *firewall* e comprometeu-se uma das estações da rede interna, ou pode partir de uma conta de um empregado comum ou até mesmo de um administrador, dependendo das metas do teste. O objetivo agora é obter o maior

nível de acesso possível à rede e aos recursos da estação através de uma escalada de privilégios [SCA 2001].

Um teste de invasão consiste basicamente de quatro etapas, figura 3.1:

1. planejamento: fase de definição do teste, regras e objetivos;
2. descoberta: fase de obtenção de dados a fim de se identificar possíveis alvos e vulnerabilidades;
3. ataque: fase em que possíveis vulnerabilidades são testadas, verificadas e confirmadas;
4. relatório: fase que ocorre em paralelo às outras, visa a gerar um documento com as fragilidades e vulnerabilidades identificadas em uma rede e sugerir soluções e medidas para os problemas encontrados.

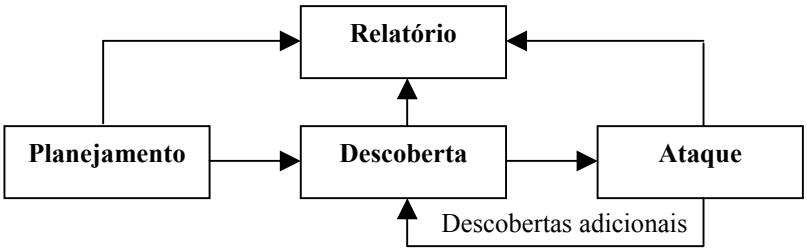

FIGURA 3.1 - Etapas de um Teste de Invasão [WAC 2002]

A etapa representada pelo ataque a um alvo é a mais complexa das fases de um teste de invasão, pois exige grande perícia e conhecimento por parte do avaliador ou atacante além de ser de difícil automatização. Pode-se dividir a fase de ataque em quatro subfases, figura 3.2:

- acesso ao alvo: subfase em que se tenta acessar o alvo e explorar uma de suas vulnerabilidades;
- escalada de privilégios: subfase em que já se possui acesso ao alvo e tenta-se obter um maior privilégio de utilização dos seus recursos;
- procura de novos alvos: subfase que ocorre junto a escalada de privilégio e procura por novos alvos;
- instalação de programas: subfase final de um ataque que visa a garantir o domínio do alvo invadido a partir da instalação de programas que permitam um maior acesso e controle do alvo.

FIGURA 3.2 - Subfases de um ataque

O teste de invasão é um teste muito eficiente e realista que verifica e demonstra as vulnerabilidades conhecidas existentes em uma rede, contudo requer uma grande preparação e conhecimento técnico. A sua implementação é difícil tanto por questões técnicas, pois pode provocar a perda de desempenho da rede ou mesmo a parada de aplicações críticas para a organização durante a sua realização, quanto legais, visto que certos procedimentos necessitam de permissões especiais para serem realizados.

3.1.4 Análise de *Logs*

Apesar de não poder ser considerado um teste propriamente dito, a revisão de *logs* de sistemas é uma tarefa crucial para avaliação de segurança de uma rede, pois permite ter uma visão dinâmica das atividades que estão acontecendo em uma organização e compará-la com o que é considerado normal e/ou padrão.

Vários sistemas em uma rede utilizam-se de registros para armazenar as atividades que estão desenvolvendo ou que estão monitorando, tais como: *firewalls*, IDSes, servidores, banco de dados, etc. A partir da análise desses *logs*, é possível identificar desvios da política de segurança da empresa, mau funcionamento e/ou configuração de mecanismos de segurança e até detectar possíveis intrusões.

Assim, a análise periódica dos *logs* de uma organização é bastante importante, principalmente junto com a aplicação de testes de segurança, pois esses geram muitas informações que são armazenadas na forma de registros nos sistemas sendo avaliados. A auditoria manual desses registros de sistemas pode ser complexa e vagarosa, por isso é aconselhada a utilização de ferramentas que automatizem essa atividade, contudo deve-se cuidar para filtrar apenas o que não é desejado e passar todos os demais registros, caso contrário, corre-se o risco de filtrar o que se deseja e qualquer outro evento anômalo.

3.2 Aplicação das Técnicas de Avaliação de Segurança

Nessa seção, as técnicas para avaliação de segurança vistas anteriormente são aplicadas a dois tipos de sistemas de segurança existentes para redes: *firewalls* e sistemas de detecção de intrusão. Esses dois mecanismos não são os únicos existentes, mas são aqui abordados por serem os mais estudados e utilizados atualmente. Primeiramente, é feita uma breve descrição desses equipamentos e, em um segundo momento, são descritos alguns modos possíveis de se avaliar tais mecanismos.

3.2.1 *Firewall*

No mundo real, um *firewall* (ou porta corta-fogo), é projetado para evitar que o fogo se propague em um edifício de um andar para outro. Teoricamente, em computação, um *firewall* possui um significado similar: ele previne que os perigos da Internet se propaguem para dentro de uma rede interna. Os objetivos de um *firewall* são [CHA 2000]:

- restringir o acesso a rede local;
- prevenir que intrusos obtenham informações sobre a rede local;
- evitar que dados sigilosos deixem a rede local;

Devido aos seus objetivos, *firewalls* são dispositivos que ficam normalmente localizados nas bordas de uma rede capazes de monitorar todo o tráfego de informação entre a rede local e a Internet. Assim, todos os dados que passam pela rede são verificados e os fluxos de comunicação indesejados podem ser filtrados.

Os *firewalls* são sistemas que podem aumentar consideravelmente a segurança de uma rede, limitando a exposição de uma rede local aos perigos da Internet, ao mesmo tempo em que permitem uma monitoração e um registro eficiente das atividades existentes entre a rede local e a Internet. Contudo um *firewall* não é a solução perfeita e possui muitas limitações como:

- um *firewall* não pode proteger a rede de usuários maliciosos;

- um *firewall* não pode proteger a rede de conexões que não passem por ele;
- um *firewall* não pode proteger a rede de ataques totalmente novos;
- um *firewall* não pode proteger totalmente uma rede contra vírus;
- um *firewall* não se configura de forma automática.

Os *firewalls* podem ser classificados basicamente em dois tipos: os baseados em filtro de pacotes e os baseados em aplicação (*proxy*). Um filtro de pacotes roteia os pacotes entre a rede local e a Internet, mas faz isso seletivamente, ou seja, é possível bloquear certos tipos de pacotes conforme a política de segurança da empresa e, conseqüentemente, impedir certos tipos de comunicação ou fluxo de dados. Um *proxy*, por sua vez, é um dispositivo especializado que intercepta a requisição de um serviço Internet por parte de um usuário, como FTP ou Telnet, e o redireciona ao servidor destino. Desta forma, o *proxy* pode avaliar as requisições do usuário e decidir quais são permitidas e quais são proibidas.

Os filtros de pacotes, figura 3.3, analisam os pacotes segundo algumas informações contidas nos datagramas IP, tais como: origem do datagrama, destino do datagrama, protocolo e tamanho do pacote. Alguns *firewalls* ainda podem verificar o campo de dados do datagrama IP para verificar a correta formatação dos dados transportados ou para procurar alguma informação específica como o nome do *site*, usuário, etc. Adicionalmente, dados como interface de origem do pacote, número de pacotes originados ou destinados a uma determinada máquina podem ser utilizados pelo filtro de pacotes para determinar a atitude que deve ser tomada.

Uma vez analisado o pacote a ser roteado, uma das seguintes atitudes pode ser tomada pelo *firewall*:

- enviar o pacote ao seu destino;
- descartar o pacote, sem notificar ninguém sobre o descarte;
- rejeitar o pacote: recusar a enviar o pacote ao seu destino e retornar um erro a origem do pacote;
- logar informações sobre o pacote;
- enviar um alarme, notificando a existência de um determinado pacote.

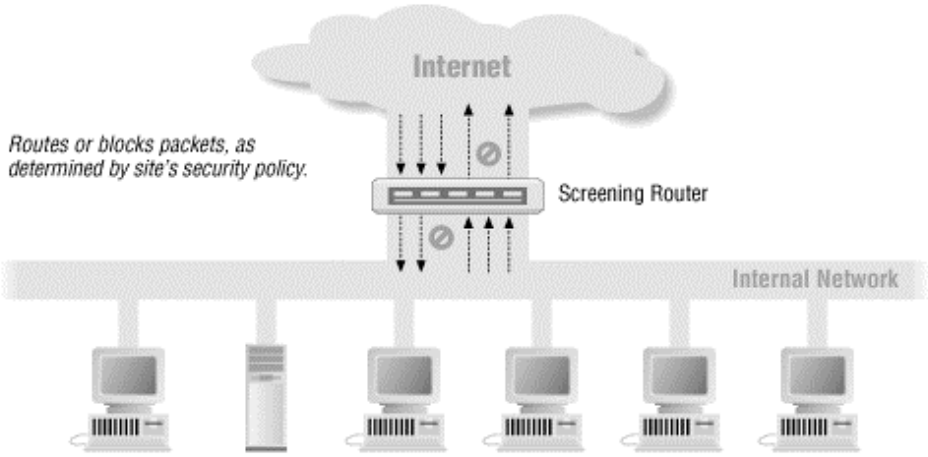

FIGURA 3.3 - Filtro de Pacotes [CHA 2000]

Os filtros de pacotes podem manter um histórico dos pacotes recebidos, neste caso, sendo chamados de *stateful packet filters*, ou podem ser *stateless packet filters*, ou seja, não armazenam informações sobre pacotes anteriores; toda a filtragem é feita com

base somente no pacote atual. As vantagens de *firewalls* baseados em filtro de pacotes é que são extremamente eficientes e facilmente encontrados em dispositivos em *hardware* ou *software*. Contudo este tipo de *firewall* não é perfeito, pois podem diminuir a performance de roteamento e nem todas as políticas de segurança podem ser implementadas utilizando esse tipo de tecnologia. As maiores dificuldades encontradas com este tipo de *firewall* são a dificuldade de configuração e dificuldade de teste das regras de filtragem.

Enquanto os filtros de pacotes atuam no nível de rede, coletando informações dos pacotes que trafegam na rede, os *firewalls proxies* agem no nível de aplicação como *gateways* de serviços. Por isso, são também conhecidos como *application-level gateways* ou *gateways* do nível de aplicação. A grande vantagem deste tipo de *firewall* está na sua transparência; o usuário tem a noção de estar se comunicando diretamente com o servidor na Internet. Ao invés disso, cada uma das partes da comunicação se comunica com o *proxy*, figura 3.4.

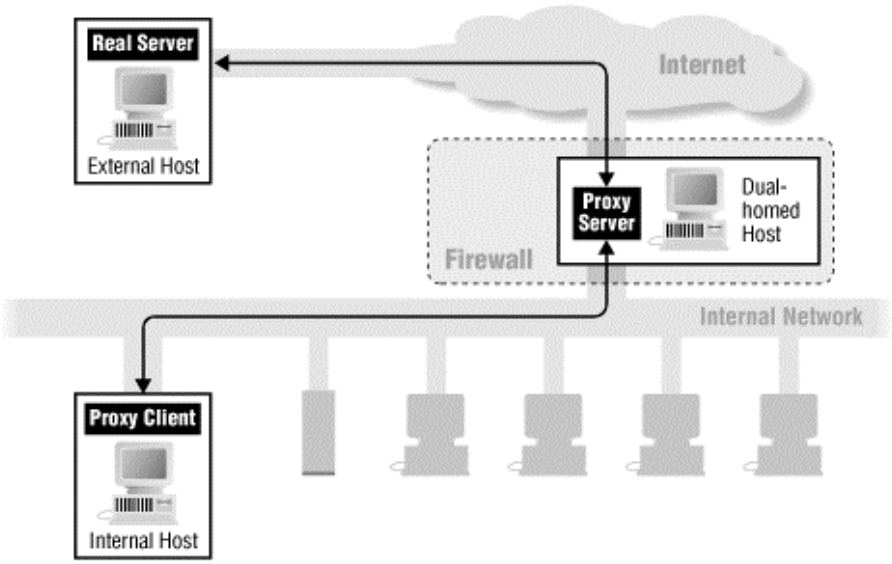

FIGURA 3.4 - *Proxy* [CHA 2000]

O fato de o *proxy* estar no nível de aplicação faz com que ele entenda as requisições dos clientes e as respectivas respostas do servidor, podendo, desta forma, realizar uma melhor avaliação da comunicação existente e, conseqüentemente, um controle mais eficiente do que é permitido e do que é proibido. Além disso, ele pode basear suas decisões com base no usuário que requisitou o serviço. Ou seja, pode-se possuir diferentes políticas de segurança para cada usuário da rede, o que dificilmente é possível através do uso de filtros de pacotes.

As vantagens dos *firewalls* baseados em *proxy* são:

- bons *logs*: uma vez que entende a aplicação o *proxy* pode realizar *logs* mais efetivos;
- uso de *cache*: pode-se aumentar a performance, tendo-se uma cópia dos dados mais solicitados pelos usuários;
- filtro inteligente: pode-se procurar por conexões específicas. Detecção de vírus mais eficiente;
- autenticação de usuário: como o *proxy* participa ativamente das conexões existentes, pode-se fazer autenticação de usuários e tomar decisões dependendo do usuário;

- proteção contra pacotes mal formados: como o *proxy* fica entre o servidor e o cliente, ele pode proteger o cliente de pacotes mal formados.

As desvantagens deste tipo de *firewall* são:

- necessidade de um *proxy* para cada tipo de protocolo. *Proxies* para protocolos antigos como FTP e Telnet são fáceis de se encontrar, mas o mesmo não ocorre para os protocolos mais novos;
- necessidade de modificações nos programas clientes. Por causa disso, nem sempre aplicações *proxy* funcionam tão bem como aplicações sem *proxy*.

3.2.2 Avaliação de um *firewall*

Apesar de não existir nenhum tipo de padronização ou critério de avaliação que possa ser utilizado para verificar as características de um *firewall*, é possível enumerar alguns tipos de abordagens possíveis e algumas características desejáveis desta tecnologia. Quanto às abordagens existentes para verificar-se o correto funcionamento de um *firewall*, pode-se avaliá-lo através das informações obtidas do seu vendedor, de análises feitas no seu projeto, das análises de seus *logs* ou através de testes de invasão [HAE 97].

Os dados disponibilizados pelos vendedores desse tipo de tecnologia de segurança informam sobre o nível de confiança que pode ser esperada do *firewall*. Apesar do vendedor ser provavelmente a pessoa que mais conhece o seu produto, não se pode confiar totalmente nesse tipo de informação, pois nem sempre ele será objetivo ou especificará as fraquezas existentes no seu produto, ainda mais num mercado competitivo como o atual.

A segunda possibilidade é obter-se informações das análises feitas durante o projeto do *firewall*. As análises que podem ser realizadas durante a fase de projeto, são as baseadas em medidas (abordagem analítica) e/ou as baseadas em simulação. Essas abordagens consistem, respectivamente, em uma análise do código fonte e em uma analise dos dados de simulações. Esse tipo de informação, além de demorar muito tempo para ser obtida e possuir uma série de limitações como visto anteriormente, também requer bastante conhecimento de implementação e funcionamento de um *firewall*. Na maioria das vezes, pouca informação é conseguida, e muitas interações do *software* com o sistema operacional, *hardware* e outros aplicativos são deixados de fora. Além disso, qualquer modificação do *software*, após a fase de projeto, pode não ser considerada.

Outra abordagem possível se dá através da análise dos *logs* produzidos por um *firewall*. Neste caso, deve-se instalar um *firewall*, configurá-lo conforme a política de segurança desejada e deixá-lo rodando. Após algum tempo, analisam-se os *logs* do *firewall*. E, através desses *logs*, pode-se detectar e corrigir possíveis erros de configuração. Contudo essa abordagem possui dois problemas básicos. O primeiro é que se pode detectar um erro de configuração tarde demais, ou seja, o sistema já foi invadido. O segundo é que nem todas as tentativas de invasões podem ter sido feitas e conseqüentemente não estarão presentes no *log*, ou pior, o sistema pode ter sido invadido e o *log* alterado.

Até o momento nenhuma das abordagens vistas permite uma boa análise das características e do correto funcionamento das tecnologias utilizadas por um *firewall*. Todas as técnicas possuem problemas, não são confiáveis, são muito demoradas ou possuem resultados incompletos. Essas técnicas podem até ser utilizadas, mas não

devem ser os únicos métodos de análise. Pelo contrário, esses métodos de avaliação devem ser usados paralelamente como fonte de dados adicionais.

Outra forma de realizar a avaliação de um *firewall* é através de testes de invasão. Esse é o método mais eficaz e também o mais utilizado. Através dessa técnica, tenta-se invadir o *firewall*, bem como transpassá-lo. A fim de realizar tais testes, usam-se vulnerabilidades conhecidas, erros de configuração e deficiências nas políticas de segurança. Assim, injetando-se falhas no *firewall*, obtêm-se informações importantes sobre o funcionamento deste e de sua correta configuração. O mais interessante nesse tipo de técnica é que se testa o *firewall* a partir do ponto de vista de um invasor ou *hacker*.

Num teste de invasão, visa-se primeiro a testar a correta implementação das políticas de segurança. Assim, os seguintes pontos devem ser testados:

- controle de acesso: somente podem ser acessados arquivos permitidos;
- *logs*: os *logs* são gerados corretamente;
- alarmes: atividades ilícitas geram os devidos alarmes;
- serviços disponíveis: somente os serviços permitidos são acessados através da Internet.

Outro ponto importante a ser verificado é se toda a comunicação existente entre a Internet e a rede local é realizada através do *firewall*, ou seja, não existe nenhum outro ponto de comunicação entre a Internet e a rede local que não seja através do *firewall*.

As fases existentes para se estruturar um teste de invasão a um *firewall* são basicamente três: obtenção de dados, ataques da Internet e ataques da rede local. A fase de obtenção de dados se preocupa em conseguir a maior quantidade de informações possíveis sobre o alvo a ser atacado, no caso o *firewall*. A coleta de dados pode ser indireta, através de informações obtidas em sites, DNS, etc, ou direta através do mapeamento de rede. A diferença entre elas está no fato de que a coleta indireta não pode ser detectada ou logada uma vez que se baseiam em informações disponíveis publicamente.

A segunda fase do teste consiste em atacar o *firewall* da Internet baseando-se nas informações obtidas durante a coleta de dados. O tipo de ataque a ser utilizado para o teste vai depender do tipo de *firewall* sendo utilizado. Nos *firewalls* baseados em filtros de pacotes, deve-se testar:

- ataques de *IP spoofing*;
- *IP source routing*;
- acesso a serviços;
- ataques de DoS.

Os *firewalls* baseados em *proxy* requerem uma abordagem diferente. A maior parte dos testes nesse tipo de *firewall* refere-se a possíveis erros de configuração do *proxy* ou a erros na implementação das políticas de segurança.

Os testes de invasão devem ser realizados juntamente com a análise dos *logs* do *firewall*, pois a injeção de falhas geradas por este método acelera a ocorrência de erros (intrusões) e, conseqüentemente, permite que uma análise dos *logs* detecte possíveis erros de configuração ou, até mesmo, erros do próprio *firewall*. Os *logs* de um *firewall* seriam equivalentes ao coletor de dados de um sistema de injeção de falhas visto no capítulo anterior, neles estariam todas as informações armazenadas sobre as falhas injetadas. Deve-se aproveitar todas as técnicas existentes para avaliação de *firewalls* e

correlacioná-las a fim de se obter uma maior quantidade de dados sobre a segurança oferecida por esse mecanismo.

A última fase de um teste de invasão consiste em ataques na rede local. Na maior parte das vezes, esses testes não têm como objetivo atacar o *firewall* propriamente; o mais freqüente é atacar-se o sistema operacional no qual o *firewall* está rodando. Nesse tipo de ataque, existe uma série de boas ferramentas que podem ser utilizadas como SATAN, Nessus, etc. Essas ferramentas são conhecidas como *scanners* de vulnerabilidades e têm por objetivo verificar as vulnerabilidades de uma determinada estação de trabalho. No caso, o *firewall* deve rodar em um computador cuja segurança individual esteja reforçada, pois a segurança de toda a rede pode depender dele.

Como pôde ser visto, existem diversas maneiras para se obter informações, com suas respectivas vantagens e desvantagens. Isto não é diferente com os testes de invasão que visam injetar falhas no *firewall*, a fim de invadi-lo ou enganá-lo. As dificuldades iniciam com a falta de dados sobre esse tipo de técnica, já que é uma abordagem relativamente nova e também por ser um campo que gera um bom número de serviços para as empresas ou pessoas que as realizam. Outra dificuldade é a inexistência de boas ferramentas para a realização de tais testes. A maioria dos *scanners* de vulnerabilidades procura por deficiências na segurança do sistema operacional e não na segurança da rede.

É importante salientar que nenhum teste de segurança é conclusivo, ou seja, o fato de um *firewall* ser aprovado num teste de invasão significa apenas que as análises feitas não descobriram nenhum problema e que ele é imune aos ataques que foram realizados. Realizar todos os testes conhecidos é muito complexo e custoso, assim como novos ataques podem surgir para os quais o *firewall* pode não estar preparado.

3.2.3 Sistemas de detecção de intrusão

Sistemas de detecção de intrusão (IDS) são sistemas que visam a identificar intrusões. Uma intrusão é caracterizada pelo uso incorreto ou não autorizado de recursos computacionais tanto por parte de usuários autorizados como por parte de pessoas estranhas (invasores). Assim, um IDS tem por objetivo monitorar a rede a procura de atividades de intrusão e de ataques e responder a elas em tempo real.

Ao contrário da tecnologia de *firewalls*, que tenta evitar que intrusões ocorram, os IDSes visam principalmente a detectar tentativas de intrusão no sistema. Baseia-se na idéia de que é praticamente impossível proteger um sistema de todas as sua falhas, e preocupam-se em identificar e notificar, de preferência em tempo real, as tentativas de invasão e os danos causados por estes.

A arquitetura de um sistema de detecção pode ser definida através de um conjunto de quatro componentes conforme o *Common Intrusion Detection Framework* (CIDF) [CAM 2001]. São eles um gerador de eventos, um analisador de eventos, um sistema para armazenar dados coletados e um sistema de contramedidas mostrados na figura 3.5

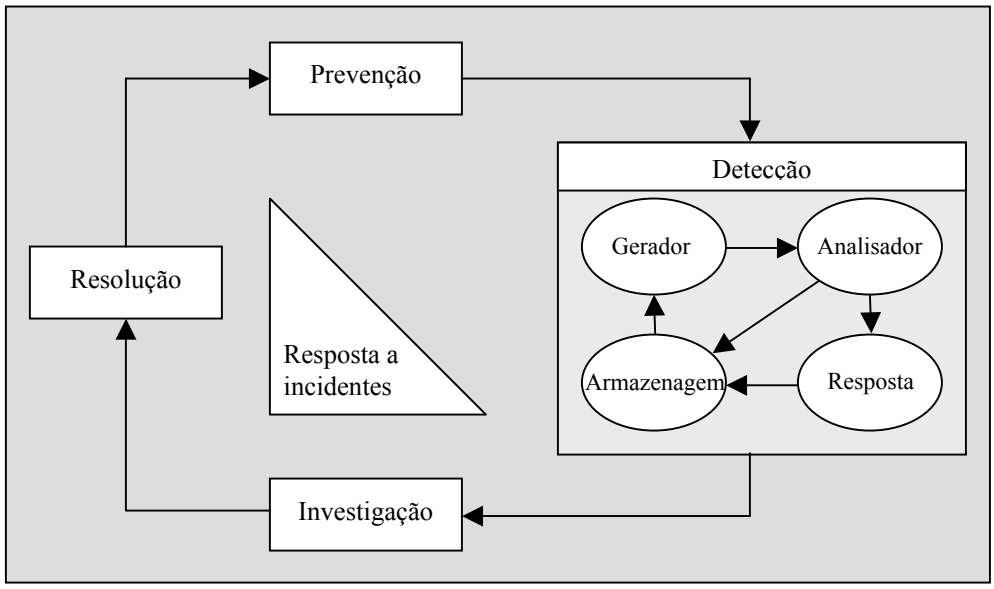

FIGURA 3.5 - *Common Intrusion Detection Framework* (CIDF)

Nessa figura estão identificadas as diversas fases existentes no gerenciamento de segurança: prevenção, detecção, investigação e resolução. Elas devem estar sempre ocorrendo em um sistema de segurança para garantir que falhas sejam identificadas, corrigidas e que falhas semelhantes sejam prevenidas. Um sistema de detecção de intrusão seria responsável pela fase de detecção e pela geração, análise, armazenamento e resposta a supostas intrusões.

O gerador de eventos consiste em um sensor que é responsável pela coleta de eventos. Um evento pode ser algo complexo ou a simples ocorrência de um determinado protocolo na rede. Os dados de entradas desse sensor podem ser originários de qualquer parte do sistema desde que possam conter evidências de uma intrusão. Exemplos são: pacotes da rede, arquivos de *log*, etc. O gerador de eventos é um componente essencial aos IDSes, pois sem eles não há informações para se tirar conclusões sobre possíveis ataques.

O analisador de eventos recebe informações de um ou mais sensores e/ou analisadores. É responsável por extrair informações relevantes dos dados coletados bem como determinar se algum tipo de intrusão ocorreu ou está em andamento. Esse componente deve retornar como saída à indicação de que uma intrusão ocorreu, se está foi a sua conclusão, e deve incluir as evidências do fato.

Tanto o analisador como o gerador de eventos produzem grande quantidade de informação que deve ser guardada caso seja necessário para uso futuro. Todos esses dados são armazenados junto a um banco de dados do próprio IDS para que possam ser analisados ou consultados futuramente.

O sistema de contramedidas é responsável por responder a possíveis ataques reconfigurando *firewalls* ou outros componentes de segurança, ou ainda, sugerindo ações a serem tomadas pelo gerente da rede. Nem todos os IDS possuem esse tipo de componente, a maioria é desenvolvida apenas para disparar alarmes. Esse componente permite a um IDS evitar maiores danos após um ataque inicial ter sido detectado.

Um outro componente que pode ser adicionado aos quatro componentes básicos de um sistema de detecção de intrusão seria a interface do mesmo com os seus usuários.

Esse componente permitiria a comunicação entre o gerente da rede e o IDS, suas saídas, configurações, controles, etc.

Além de suas divisões conceituais apresentadas acima através do CIDF, o tipo de dados gerados e analisados por um IDS pode variar significativamente conforme a abordagem de monitoração realizada pelo IDS. Os sistemas de detecção de intrusão podem ser classificados em uma das seguintes categorias baseado no tipo de dados examinados por eles: aplicação, estação, rede e integrado [BAC 99].

Um IDS baseado em aplicação coleta eventos gerados no nível de aplicação, geralmente, em forma de arquivos de *logs* gerados por banco de dados, *softwares*, *web servers* ou *firewalls*. Esse tipo de monitoração permite coletar dados com granularidade mais fina, ou seja, pode-se definir que tipo de aplicações quer-se monitorar e quais dados são importantes. Contudo a vulnerabilidade de tais aplicações pode influenciar na confiabilidade dos dados obtidos.

O IDS que coleta dados junto a uma estação ou a um sistema em particular pode obter informações tanto de *logs* do sistema como pode monitorar informações de acesso a arquivos e ações de usuários. Contudo os sensores desse tipo devem ser específicos para cada sistema, além de poderem gerar *overhead* na máquina em que estão executando. As vulnerabilidades do sistema também podem influenciar nos dados obtidos.

Há IDSes que coletam informações da própria rede. Tais dados são obtidos através do uso de *sniffers* de rede ou até mesmo através de agentes integrados a *hardwares* de rede. Esse tipo de sensor permite que se detecte ataques a rede e ataques distribuídos, além de se obter informações sobre máquinas e aplicações operando no ambiente monitorado. Contudo nem todos os dados que passam pela rede podem ser processados por este tipo de sensor dependendo da velocidade da rede. Outro problema é a complexidade de se monitorar as modernas redes baseadas em *switches*, onde cada segmento do *switch* possui uma única estação e está isolado dos demais segmentos.

Alguns sistemas de detecção combinam as diversas formas de monitoração descritas acima. Nesse tipo de IDS, é possível saber tudo o que está ocorrendo nos diversos níveis da rede e a evolução de possíveis ataques; porém, apesar de várias tentativas, ainda não existe um padrão de comunicação entre os diversos componentes de um IDS. Assim, é praticamente impossível integrar componentes de diferentes vendedores.

Outra possível diferenciação entre os diversos sistemas de detecção de intrusão refere-se à técnica utilizada pelos analisadores de eventos para a identificação de ataques. Existem basicamente três técnicas:

- análise de Assinaturas: é baseado nos vestígios deixados por ataques. Consiste em analisar os eventos e compará-los contra um banco de dados de assinaturas de ataques conhecidos;
- análise Estatística: consiste em monitorar o sistema a procura de desvios na conduta normal de programas e usuários. Não é muito comumente encontrado em IDSes por ser mais complexo de implementar e, conseqüentemente, não tão eficiente quanto à análise por assinatura. Contudo possui a vantagem de detectar ataques desconhecidos, e ataques mais complexos que ocorrem durante longos períodos;
- análise de Integridade: consiste na procura de alterações nas características e atributos de arquivos, diretórios e objeto de dados. Utiliza mecanismos de

criptografia chamados de Hash [SCH 96] ou resumo que podem identificar alterações mínimas em arquivos. Qualquer ataque que modifique arquivos é detectado, mesmo que as análises de assinatura e estatística não o façam. Infelizmente, esse tipo de análise não é realizado em tempo real, e sim, em modo *batch*.

3.2.4 Avaliação de um sistema de detecção de intrusão

Devido a sua importância para a segurança de organizações, é vital que os sistemas de detecção de intrusão funcionem de acordo com as expectativas e forneçam informações confiáveis para o gerente da rede. Um IDS cuja implementação não seja perfeita pode prover informações incorretas para o gerente bem como passar uma falsa sensação de segurança.

As implicações, advindas de falhas nesse tipo de sistema de segurança, são muito críticas e não é pouco provável que os próprios IDSes sejam alvo de ataques. Um intruso que desconfie da existência de algum IDS na rede que deseja invadir, possivelmente, irá atacar primeiro o IDS a fim de inutilizá-lo ou de forçá-lo a fornecer informações incorretas ao gerente da rede.

Cada componente existente num sistema de detecção de intrusão possui uma única função de segurança e pode ser atacado por diferentes razões. Como único elemento para coletar dados, os geradores de eventos funcionam como se fossem os olhos e ouvidos de um IDS. Um ataque a estes dispositivos faria com que não fosse mais possível para o IDS identificar o que está ocorrendo no sistema monitorado por ele.

Outro problema são as redes cada vez mais rápidas; IDSes baseados na monitoração de redes podem não ser capazes de coletar todos os dados que circulam por elas. De fato, os atuais produtos somente são capazes de capturar e analisar 100% dos dados sem nenhuma perda em redes de até 65Mbit/s [BAC 99]. Assim, em redes mais velozes, pode-se não detectar um ataque em andamento devido à perda de pacotes. Ataques que burlem a detecção por parte do analisador de eventos ou que apaguem os dados armazenados pelo IDS, bem como as outras partes de um sistema de detecção de intrusão, também são possíveis. E, portanto, um bom sistema de detecção deve ser capaz de identificar tais ataques e evitá-los.

Contudo ainda existem outros problemas em um IDS que não são gerados por subversão a ataques, e sim, por erros dele próprio. São os chamados erros de identificação de ataques e são classificados em: falso positivo e falso negativo. Um erro falso positivo ocorre quando o IDS classifica uma ação como anômala, ou seja, uma possível intrusão, quando na verdade é uma ação legítima. Não é um erro muito importante, mas deve ser minimizado ao máximo. Caso este tipo de erro ocorra muitas vezes no sistema, pode ocorrer de o gerente da rede começar a ignorar os alertas de intrusão e quando ocorrer uma intrusão real essa também ser ignorada pelo usuário do sistema. Os erros de falso negativos são mais perigosos e sérios. Ocorrem quando uma ação de intrusão acontece e o IDS a classifica como uma ação normal do sistema. Ao permitir que todas as ações ocorram, esse tipo de erro causa a falsa sensação de segurança, além de diminuir o estado de segurança da organização em relação ao estado anterior a instalação do IDS.

As abordagens existentes para verificar o correto funcionamento de um sistema de detecção de intrusão podem ser as mesmas utilizadas para a verificação de *firewalls*.

Contudo, como foi visto anteriormente, as informações obtidas através do seu vendedor, de análises feitas do seu projeto e das análises de seus *logs* apresentam deficiências, e os dados obtidos a partir dessas análises devem ser utilizados como um complemento.

O método mais utilizado em testes para verificação de IDS também é o teste de invasão. Mas, no caso dos sistemas de detecção de intrusão, os objetivos são outros. Independentemente dos mecanismos em que se baseiam e de como funcionam, é desejável que todos os IDS possuam as seguintes características [CRO 95]:

- rodar continuamente;
- ser tolerante a falhas; deve sobreviver a quedas do sistema e não perder sua base de dados;
- resistir a subversões, ou seja, o sistema deve ser capaz de monitorar a ele próprio;
- gerar pouco *overhead*;
- ser difícil de burlar.

Todos os requisitos acima devem ser cumpridos pelos sistemas de detecção de intrusão. Caso o sistema fosse facilmente enganado, vários ataques poderiam não ser detectados. É desejável que o sistema consuma poucos recursos, senão poderia ser impossível ou impraticável de utilizá-lo em alguns ambientes. Além disso, o sistema deve ser resistente e tolerante a falhas, ou seja, ele deve continuar funcionando mesmo em situações adversas ou estressantes.

Num teste de invasão direcionado a IDSes, deve-se testar todos os seus componentes. O teste deve abranger o gerador de eventos, o analisador de eventos, o armazenamento de dados e o sistema de contramedidas. Existem basicamente três tipos de testes que podem ser realizados [PUK 96]: teste de detecção de intrusão, teste de uso de recursos e teste de estresse.

No teste de detecção de intrusão, o objetivo principal é verificar a habilidade de um sistema de detecção de distinguir entre uma atividade normal da rede e uma atividade de intrusão. Nesse caso, deve-se rodar uma série de ataques conhecidos contra um IDS e verificar se ele é capaz de reconhece-los. É importante salientar que nesse teste deseja-se apenas verificar a capacidade de detecção do sistema, logo deve ser evitada ao máximo a existência de comunicações não relacionadas com o ataque. A avaliação dos resultados desse teste pode ser feita através dos *logs* gerado pelo IDS. Quantas simulações de ataques foram classificadas como ataque e quantas foram classificadas como atividade normal do sistema. Outra característica que pode ser observada é o número de falsos negativos e o número de falsos positivos que foram gerados. Todas essas informações indicam quão bem um sistema de detecção de intrusão foi no teste de detecção de intrusão.

O segundo teste visa a mensurar quanto de recursos do sistema um IDS utiliza. Os resultados desse teste podem verificar a viabilidade de uso ou não de um sistema de detecção. O teste consiste em executar alguns ataques simulados e verificar quanto de CPU e disco o *software* utiliza durante a detecção de intrusão e na geração de *logs* dos ataques. Os testes devem rodar por um período determinado a fim de calcular a razão entre espaço em disco e tempo de CPU utilizados pelo tempo de monitoração de um ataque. Os resultados desse experimento podem ser utilizados para definir os requisitos mínimos de um IDS quando em execução em uma rede real.

O último experimento objetiva observar o funcionamento dessa tecnologia em situações de uso intenso de uma rede. Por exemplo, um IDS pode ser capaz de detectar

certos ataques em condições normais de uso da rede, mas pode falhar em outras situações como durante um pico de utilização da rede. A fim de realizar essa verificação deve-se simular comunicações normais de usuários na rede durante a simulação dos ataques. Assim, deseja-se verificar até que nível de ruído (comunicações normais) o IDS ainda é capaz de detectar intrusões.

O teste deve iniciar com um número de comunicações consideradas normais e ir aumentando progressivamente. A cada aumento de sessões monitoradas deve-se executar uma série de ataques e verificar a taxa de detecção obtida. Os resultados desse teste podem ser utilizados para definir o limite máximo de sessões que um IDS pode monitorar sem perder a capacidade de detecção de intrusão e o limite máximo de sessões monitoradas que o IDS suporta sem acabar com os recursos do sistema em que ele está executando.

Apesar do grande investimento existente em pesquisa e desenvolvimento, a tecnologia de detecção de intrusão ainda é bastante recente e sua eficiência é limitada. Por isso, seu uso é importante para acrescentar um nível a mais na segurança de uma rede, contudo a segurança de uma rede não deve estar baseada apenas em um IDS.

A avaliação desses sistemas não é uma tarefa trivial e há uma falta de produtos para a verificação confiável deles. Da mesma maneira que no teste de *firewalls*, as ferramentas existentes não foram projetadas para a realização de tais testes, a maioria dos *scanners* de segurança é baseado na segurança do sistema operacional e não em segurança de rede [MCH 2000]. Logo, existe uma carência de boas ferramentas para testes de sistemas de segurança em rede.

Todas as avaliações realizadas até hoje indicam que os sistemas de detecção de intrusão possuem uma avaliação apenas razoável para identificar ataques conhecidos, e deficiente para reconhecer novos tipos de ataques. Em 1999, a IBM de Zurique realizou testes com dois sistemas comerciais, o RealSecure 3.0.x e o NetRanger 2.1.2. Foram utilizados ataques de *scripts* e ferramentas existentes na base de dados deles e os quais eles deveriam reconhecer. O RealSecure detectou 30 dos 42 ataques executados, enquanto o NetRanger detectou 18 dos 32. Além disso, ambos os sistemas tiveram uma alta taxa de falsos alarmes [MCH 2000]. O porquê de tão baixo índice de detecção, mesmo tendo os ataques em suas bases de dados, é outra dificuldade existente na avaliação de sistemas de detecção de intrusão. Um IDS pode ser afetado pelas várias condições do sistema. Assim, a detecção ou não de um ataque pode ser influenciada pelo nível de atividade do sistema. Um ataque pode ser normalmente detectado por um IDS, contudo o mesmo ataque pode não ser detectado quando a carga da rede e/ou do computador estiver alta.

4 Ferramentas para Injeção de Falhas em Sistemas de Segurança de Rede

Como foi visto, existem diversos métodos que podem ser utilizados para avaliar os sistemas de segurança de uma rede de computadores. Dos métodos enumerados nesse trabalho, normalmente, quatro são utilizados para avaliar mecanismos de segurança em rede, como *firewalls* e sistemas de detecção de intrusão; são eles: mapeamento de rede, varredura de vulnerabilidade, teste de invasão e análise de logs.

Todos possuem uma característica em comum: a falta de programas especificamente desenvolvidos para injeção de falhas em sistemas de segurança em rede. A maioria das ferramentas encontrada é inadequada para a realização desses testes, pois foram projetadas com outras finalidades [MCH 2000].

Uma pesquisa, realizada para o levantamento de ferramentas disponíveis para injeção de falhas em sistemas de segurança em rede, observou a existência de basicamente dois tipos de programas para esse fim. Existem as ferramentas para geração de pacotes e os programas de varredura de vulnerabilidades.

Os aplicativos para geração de pacotes, também conhecidos como injetores de pacotes, são ferramentas que permitem a criação e envio de pacotes de protocolos da pilha TCP/IP para uma determinada estação. Dessa forma, é possível criar qualquer tipo de pacote, forjando-se comunicações e origens de dados com a finalidade de testar-se sistemas como *firewalls* e sistemas de detecção de intrusão.

Outras ferramentas encontradas são os *scanners* de vulnerabilidades. Essas ferramentas são desenvolvidas com o objetivo de automatizar a procura por falhas em serviços e estações existentes em uma determinada rede. Apesar de não serem aplicativos que realizem testes em rede, são bastante usados para testar os mecanismos de detecção de um IDS por serem utilizados por intrusos que buscam vulnerabilidades em um determinado alvo.

A seguir é feita uma análise desses dois tipos de ferramentas enumerando suas vantagens e desvantagens visando a adequação dessas ferramentas junto às técnicas de avaliação de segurança em rede. E, no final do capítulo, é apresentada uma tabela resumo das características dessas ferramentas.

4.1 Ferramentas para Injeção de Pacotes

As ferramentas para injeção de pacotes permitem que pacotes de protocolos da pilha TCP/IP sejam criados e injetados em uma rede. Os tipos de protocolos suportados e os dados que podem ser configurados variam de aplicativo para aplicativo. De modo geral, os protocolos suportados são o IP, TCP, UDP e ICMP.

Foram vistas algumas das ferramentas existentes na Internet para a geração e envio de pacotes de protocolos da pilha TCP/IP como hping [SAN 2001], nemesis [GRI 2000] e sendip [RIC 2000]. Apesar de existirem inúmeros aplicativos destinados a esse fim, de modo geral todos possuem as mesmas características.

Esse tipo de programa poderia ser utilizado para testar os protocolos aceitos por um *firewall*, pois implementa diversos tipos de protocolos. Contudo o modo como são implementados e as funcionalidades que disponibilizam para os usuários não permitem que isso seja viável.

O problema, atualmente, em utilizar esse tipo de software para injetar falhas em sistemas de segurança em rede é que eles não fornecem as características necessárias para tais fins. As ferramentas de injeção de pacotes de modo geral:

- são em modo texto;
- não enviam seqüências de pacotes;
- possuem poucos recursos para monitorar respostas da máquina destino;
- não permitem a implementação de testes sofisticados.

A desvantagem dessas ferramentas serem em modo texto é o fato de não fornecerem uma interface muito amigável para utilizar. Como exemplo, pode-se citar que cada ferramenta de injeção de pacotes possui uma interface de utilização diferente, o que dificulta a migração entre os diversos programas. É até desejável que a ferramenta possua uma interface em modo texto para ser mais fácil utilizá-la em conjunto com outros programas, mas a existência de uma interface gráfica dos diversos protocolos implementados tornaria a sua utilização muito mais fácil e intuitiva.

Seria desejável a possibilidade de enviar seqüências de pacotes ao invés de um único pacote; pois, assim, permite-se a realização de simulações de fluxo de comunicação entre diversas máquinas. Adicionalmente, o envio de seqüência de pacotes também permite que um maior número de ataques seja simulado, aumentando-se a flexibilidade de tais ferramentas.

Além disso, apesar de, nos sistemas de segurança de rede, grande parte da coleta de dados ter que ser realizada pelo próprio sistema de segurança a fim de testar seu mecanismo de *log*, é desejável que a própria ferramenta de injeção forneça algum tipo de *feedback* sobre as ações que estão sendo tomadas pelo computador alvo. Dessa forma, quanto maior os recursos de monitoração disponibilizados pelo injetor de pacotes melhor, contudo a maioria das ferramentas de injeção de pacotes não retorna nenhuma informação sobre os pacotes enviados, com exceção da ferramenta hping.

As ferramentas de injeção de pacotes por atuarem diretamente sobre a rede são o tipo de aplicativo mais adequado para injetar falhas em uma rede. Contudo pode-se notar que existe uma série de melhorias que podem ser feitas nessas ferramentas para melhor adequá-las ao teste de sistemas de segurança em rede. As atuais características desses aplicativos não permitem que eles sejam muito utilizados para a injeção de falhas, mas, caso os problemas anteriormente mencionados sejam solucionados, a implementação de testes mais sofisticados seria possível, e o uso de injetores de pacotes para avaliação de *firewalls* ou sistemas de detecção de intrusão tornar-se-ia viável.

4.2 Ferramentas para Varredura de Vulnerabilidade

As ferramentas para varredura de vulnerabilidade foram desenvolvidas para automatizar a busca por falhas em serviços e em estações de trabalho existentes em uma rede de computadores. Podem ser utilizadas como ferramentas administrativas para o gerenciamento de segurança, e também como ferramentas de ataque por intrusos à procura de falha que possam ser exploradas para invadir uma estação. Entre os aplicativos encontrados na Internet para esse fim pode-se citar: Nessus [DER 2000], SAINT [WOR 2000] e SARA [TOD 1999].

Esse tipo de aplicativo primeiramente realiza uma varredura na estação alvo a fim de verificar os serviços ativos, as portas abertas e o tipo de sistema operacional existente nela. Após a varredura inicial, e com base nos seus resultados obtidos é feita uma série de testes a procura de falhas conhecidas que possam existir nos serviços ou no

sistema operacional. Ao final dos testes, é gerado um relatório das características das estações e dos problemas encontrados.

Os programas para varredura de vulnerabilidades realizam a maioria dos seus testes no nível de aplicação, procurando falhas em serviços, aplicativos, sistemas operacionais e representam um grande risco à segurança de uma rede quando utilizado por pessoas não autorizadas. Assim, as ferramentas de segurança de rede, como os IDS, devem ser capazes de detectar a ação de tais aplicativos.

As ferramentas para teste de vulnerabilidades possuem algumas das características desejáveis em uma ferramenta para injeção de falhas em sistemas de segurança. Elas possuem interface gráfica bem desenvolvida que aumenta a facilidade de uso do aplicativo, mas também mantém uma interface em modo texto caso esta seja necessária. Outra vantagem, a qual pode ser tomada como a principal desses programas, é que eles permitem o envio de seqüência de pacotes e também a realização de testes sofisticados.

Contudo a maioria delas não possui um mecanismo de monitoração das respostas do destino acessível aos usuários. Essas ferramentas monitoram as respostas da estação sendo testada, mas o usuário não tem acesso direto as respostas. O único contato é através do relatório de vulnerabilidades, após a realização dos testes. Elas também não permitem uma maior iteração dos usuários na realização dos testes de vulnerabilidades, ou seja, não é possível ao usuário criar um teste a ser executado, com exceção do Nessus [DER 2000], devido a sua linguagem *script* NASL.

De modo geral, esse tipo de aplicativo seria o que há de mais próximo de um injetor de falhas de sistemas de segurança em rede, a não ser por um detalhe: o nível das falhas testadas. As falhas que esses programas injetam são, em sua maioria, falhas do nível de aplicação, enquanto que o que é desejado é a injeção de falhas diretamente no nível de rede. Além disso, elas testam vulnerabilidades existentes em serviços e estações de uma rede e não em sistemas de segurança como *firewalls* ou sistemas de detecção de intrusão.

Mesmo assim, eles são bastante utilizados para verificar a capacidade de um IDS em detectar o uso desse tipo de ferramenta. O *scanner* de vulnerabilidade ideal é aquele que preenche os seguintes requisitos:

- possua um banco de dados de vulnerabilidades sempre atualizado;
- possua um relatório de vulnerabilidade preciso e conciso, evitando ao máximo a ocorrência de falsos positivos ou falsos negativos;
- possua mecanismos para análise entre relatórios de modo a realizar comparações e analisar possíveis tendências;
- forneça informações relevantes e concisas dos problemas encontrados e das maneiras de corrigi-los.

Uma pesquisa feita pela *Network Computing* [FOR 2001] entre as diversas ferramentas de varredura de vulnerabilidades existentes (públicas e privadas) demonstrou que nenhum dos programas existentes preencheu todos os requisitos acima. A análise das ferramentas foi baseada na capacidade de detecção de falhas existentes em diversas plataformas, na facilidade de utilização e na qualidade dos relatórios gerados por elas.

Das 17 vulnerabilidades mais críticas existentes, nenhum dos produtos testados foi capaz de detectar todas elas. O programa que obteve o melhor resultado foi o Nessus Security Scanner, o qual detectou 15 das 17 vulnerabilidades. A ferramenta SARA e a

SAINT detectaram, respectivamente, 10 e 9 das 17 vulnerabilidades. Isso serve para demonstrar que não é possível se basear apenas nos resultados de um aplicativo para se garantir a segurança de uma máquina ou sistema.

Fonte: FORRISTAL. Network Computing

TABELA 4.1 - Características dos *Scanners* de Vulnerabilidades

	Nessus Security Scanner	**SARA**	**World Wide Digital Security SAINT**
Preço	Gratuito	Gratuito	Gratuito (gerador de reporte a partir de $100)
Plataforma	Unix	Unix	Unix
Atualização automática da base de assinaturas	● (Web)	○	○
Procura por vulnerabilidade na estação	○	○	○
Referência CVE	●	●	●
Correção automática de vulnerabilidades	○	○	○
Código aberto	●	●	●
Linha de comando	●	●	●
Integração com banco de dados	○	○	○
Permite customização de testes **● sim ○ não**	● (NASL)	●	●

Fonte: FORRISTAL. Network Computing

TABELA 4.2 - Resultado da Detecção de Vulnerabilidades

	Nessus Security Scanner	**SARA**	**World Wide Digital Security SAINT**
Hewlett-Packard HP-UX 10.20			
Sendmail buffer overflow	○	●	●
SMTP mail relaying	●	●	○
SMTP EXPN/VRFY	●	○	●
FTP /pub/ directory writable	●	○	○
Microsoft Windows NT 4.0			
Coldfusion sample scripts	●	○	⊙
IIS sample scripts	●	⊙	⊙
RDS vulnerability	●	●	●
NULL session allowed	●	○	○
Novel NetWare 5.1			
Guessable SNMP string	●	○	●
Red Hat Linux 5.2			
wu-ftpd buffer overflow	○	●	●
NFS root (/) export	●	○	○
Guest account / no password	●	●	●
Bind NXT buffer overflow	●	⊙	●
Sun Microsystem Solaris 2.6			
rpc.cmsd buffer overflow	●	●	○

rpc.sadmind buffer overflow	●	●	○
rpc.ttdbserver buffer overflow	●	●	○
Finger service vulnerabilities	●	●	●
Total	**15**	**10**	**9**
● sim ○ não ⊙ identificação parcial			

4.3 Comparação entre as ferramentas de injeção de falhas em sistemas de segurança de rede

Nenhuma das ferramentas analisadas foi desenvolvida explicitamente para a injeção de falhas em sistema de segurança de rede, por isso nenhuma delas apresenta todas as características necessárias para realizar tal função. Um resumo das características que esses aplicativos possuem pode ser visto nas tabelas abaixo, um estudo mais amplo pode ser visto em [WAN 2001].

TABELA 4.3 - Resumo de Características das Ferramentas de Injeção de Falhas

Ferramenta	**Interface**		**Injeção de Falhas**		**Protocolos Suportados**				
	Texto	Gráfico	Rede	Aplicação	ip	tcp	udp	Icmp	Outros
Hping2	✓		✓		✓	✓	✓	✓	
Sendip	✓		✓		✓	✓	✓	✓	✓
Nemesis	✓		✓		✓	✓	✓	✓	✓
SAINT	✓	✓		✓	✓	✓	✓	✓	✓
SARA	✓	✓		✓	✓	✓	✓	✓	✓
Nessus	✓	✓		✓	✓	✓	✓	✓	✓

Ferramenta	**Seqüenciamento de Pacotes**		**Linguagem de Programação**		**Monitoração de Respostas**	
	Sim	Não	Sim	Não	Sim	Não
Hping2		✓		✓	✓	
Sendip		✓		✓		✓
Nemesis		✓		✓		✓
SAINT	✓			✓	✓	
SARA	✓			✓	✓	
Nessus	✓		✓		✓	

A partir do estudo e análise dessas ferramentas, foi possível especificar as características desejáveis em uma ferramenta para avaliação de mecanismos de segurança em rede. O modelo dessa ferramenta é apresentado no capítulo seguinte e, em um capítulo posterior, é descrita a implementação de um protótipo.

5 Modelo de um Sistema de Injeção de Falhas em Redes

Como pôde ser visto no capítulo anterior, existem dois tipos de ferramentas que podem ser utilizadas para injeção de falhas em segurança de rede. Os injetores de pacotes capazes de injetar falhas no nível de rede, mas que possuem poucas funcionalidades. E os *scanners* de vulnerabilidades, ferramentas automatizadas de mais alto nível e de fácil utilização que possuem maiores funcionalidades.

A ferramenta ideal para injetar falhas em sistemas de segurança de rede está em um meio termo entre esses dois tipos de aplicativos. É um injetor de falhas do nível de rede do mesmo modo que os geradores de pacotes, mas com uma maior flexibilidade de utilização como os *scanners* de vulnerabilidades. Nesse caso, essa ferramenta deve poder ser utilizada tanto em modo texto como em modo gráfico e deve ser capaz de gerar e enviar pacotes de protocolos do nível de rede, transporte e aplicação. Além disso, ela deve possuir uma linguagem de programação que permita enviar seqüência de pacotes conforme as respostas monitoradas da máquina alvo.

Segundo Hsueh [HSU 97], um sistema de injeção de falhas deve possuir os seguintes elementos:

- injetor de falhas;
- biblioteca de falhas;
- gerador de *workload*;
- biblioteca de *workload*;
- controlador;
- monitor;
- coletor de dados;
- analisador de dados.

O sistema proposto, por ser um injetor de falhas, deveria realizar todas as atividades de um sistema de injeção de falhas, contudo como realiza testes em mecanismos de segurança de rede algumas dessas atribuições podem ser simplificadas. O modelo proposto visa a implementar esses elementos através de três módulos, figura 5.1:

- módulo de gerência;
- módulo de injeção e monitoração;
- módulo de teste.

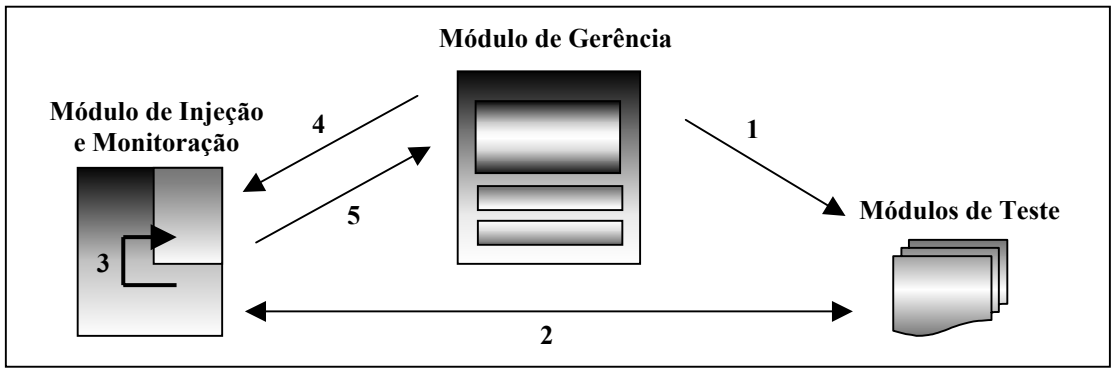

FIGURA 5.1 - Arquitetura Proposta

O módulo de gerência é responsável pela inicialização, finalização, figura 5.1 (1), e monitoração, figura 5.1 (4, 5), dos testes em execução e pela interface gráfica com o usuário do sistema; implementando, assim, algumas das funções destinadas ao controlador e ao monitor de um sistema de injeção de falhas.

Outra funcionalidade importante desempenhada pelo módulo de gerência é a de edição de pacotes. Nenhuma das ferramentas analisadas possui mecanismo que permita a edição de pacotes capturados na rede ou a criação de pacotes específicos para realização de testes na rede. O módulo de gerência, através de sua interface gráfica, deve permitir que pacotes sejam criados e editados para serem utilizados na avaliação de sistemas de segurança em rede.

O módulo de injeção e monitoração, como o próprio nome sugere, é responsável pelo injetor de falhas e pelo coletor de dados. O injetor de falhas propriamente dito é representado pela geração e envio de pacotes de protocolos da pilha TCP/IP, enquanto o coletor de dados é representado pela monitoração da rede e observação das respostas geradas pela estação sendo testada. Essas duas funções são implementadas por esse módulo e disponibilizadas para os usuários do sistema a fim de que estes as utilizem para o desenvolvimento de seus testes (*plug-ins*).

O módulo de injeção e monitoração deve fornecer as funções básicas para a injeção de falhas como funções de envio de pacotes *ethernet*, ARP, IP, UDP, TCP e ICMP, funções para monitoração da rede e funções avançadas para teste de segurança tais como *dnsspoof*, *tcpkill*, *arpspoof* e *scanning*. Desta forma, visa-se fornecer ao usuário uma base para a realização de testes simples através das funções de envio de pacotes da pilha TCP/IP e também uma base mínima para a realização de testes mais sofisticados através das funções de ataque como *dnsspoof*, *tcpkill*, etc.

Além das funções de teste, esse módulo implementa algumas funções específicas para o módulo de gerenciamento: comunicação, gerenciamento e estatísticas, figura 5.1 (3, 4, 5). As funções de comunicação, gerenciamento e estatística são importantes para a troca de informações entre o módulo de gerência e os módulos de testes, além de permitir ao usuário ter informações sobre os testes durante a sua execução e obter um resumo de cada teste ao seu final.

Uma característica importante em uma ferramenta para injeção de falhas em sistema de segurança em rede é que os usuários desse tipo de aplicativo sejam capazes de desenvolver seus próprios testes, não sendo obrigados a depender dos testes fornecidos juntos com a ferramenta. Por isso, a existência de algum tipo de linguagem para a criação de testes é necessária. As características desejáveis em uma linguagem desse tipo seriam:

- simplicidade;
- eficiência;
- funcionalidade.

A linguagem deve ser de simples aprendizado não apresentando maiores dificuldades para um programador iniciante, e não simples no sentido de suas funcionalidades. Deve ser eficiente a fim de conseguir realizar os testes sem que a tradução ou compilação da linguagem represente muito *overhead* na execução dos testes. E deve ser funcional, ou seja, as características anteriores não devem restringir a funcionalidade da linguagem, além de ser capaz de realizar e especificar testes em rede o mais variado possível.

Assim, a biblioteca de falhas e algumas funções do controlador e do monitor do sistema de injeção de falhas são de responsabilidade dos módulos de teste em execução, ou seja, quem deve criar e determinar a lógica do teste é o usuário da ferramenta, conforme o tipo de falhas que deseja injetar. Um conjunto de modelos de ataques e pacotes pode ser definido e fornecido junto com a ferramenta, contudo o usuário também é capaz de criar os seus próprios ataques e pacotes de testes.

Através do módulo de testes e de sua linguagem para especificação de testes, permite-se que esse modelo proposto atinja uma maior flexibilidade e expansibilidade que as ferramentas de geração de pacotes, aproximando-se das funcionalidades de um *scanner* de vulnerabilidades. Toda a comunicação entre os módulos de teste e a ferramenta é feita através das funções do módulo de injeção e monitoração disponibilizado ao usuário, figura 5.1 (2, 3, 5). A única interação entre os testes e o módulo de gerenciamento é durante a carga do mesmo e no momento de sua finalização, figura 5.1 (1).

Deve-se observar que o usuário final deste tipo de ferramenta não pode ser um usuário leigo, como acontece com os aplicativos de *scanner* de vulnerabilidades. O usuário deste tipo de ferramenta deve possuir um bom conhecimento de redes e protocolos a fim de poder utilizar tal sistema. Nesse caso, o perfil final é similar ao dos usuários de aplicativos geradores de pacotes.

A necessidade de um gerador de *workload* é discutível neste tipo de aplicativo, uma vez que ele visa testar um equipamento em condições normais de funcionamento. Ou seja, o *workload* seria o da rede em que o sistema de segurança estiver executando. Contudo é possível, a partir do injetor de falhas, criar tráfego e comunicações fictícias na rede a fim de aumentar a carga e, dessa forma, testar o sistema alvo com diferentes níveis de carga e tipos de comunicação. No caso de desejar-se criar um *workload* a ser injetado por essa ferramenta, a biblioteca de *workloads* funcionaria da mesma maneira que a biblioteca de falhas. Ou seja, quem deve criá-la deve ser o próprio usuário, existindo a opção de se fornecer modelos de *workloads*.

O coletor de dados e o analisador teriam suas funções compartilhadas entre o injetor de falhas e o sistema de segurança de rede sendo testado. Isso deve-se ao fato que um dos objetivos da injeção de falhas é verificar a capacidade de detecção, do correto registro das atividades e da correta ativação de alarmes e contramedidas do sistema de segurança em rede sendo testado. Assim, a coleta de dados do injetor de falhas restringe-se as respostas do equipamento testado e do tráfego existente na rede, enquanto que a análise é feita pelo usuário do injetor através das funções fornecidas pelo módulo de injeção e monitoração. Todos os demais dados devem ser coletados e analisados pelo equipamento testado e a correta funcionalidade dele depende dos testes sendo feitos, da sua política de segurança e da sua correta configuração.

O injetor de falhas em sistemas de segurança de rede poderia simular a realização de ataques como ataques de *IP spoofing*; *IP source routing* e ataques de negação de serviço, entre outros. Além de poder realizar varreduras e solicitações simuladas de serviços, existe a opção de realizá-las com diferentes níveis de utilização da rede. Todos esses testes são muito importantes para a avaliação da eficiência e do correto funcionamento de equipamentos de segurança em rede como *firewall*, sistemas de detecção de intrusão entre outros.

6 Implementação do Protótipo

Neste capítulo é descrita a implementação de um protótipo do sistema de injeção de falhas em rede especificado anteriormente. Essa implementação tem por objetivo validar o modelo proposto e avaliar suas potencialidades. Inicialmente, são apresentadas as características e funcionalidade dos módulos de gerência, injeção e monitoração e teste implementados, e, posteriormente, é descrita a construção de um módulo de teste do *TCP Kill*.

6.1 Módulo de Gerenciamento

O módulo de gerenciamento foi desenvolvido em linguagem C para a plataforma GNU/Linux e possui uma interface gráfica desenvolvida utilizando-se a biblioteca GTK (GIMP Toolkit). Através desse módulo é possível fazer o gerenciamento dos pacotes da pilha TCP/IP a serem injetados em uma rede e dos módulos de teste a serem executados pela ferramenta.

Esse módulo possui toda uma interface visando a edição de pacotes TCP/IP. Através dessa interface é possível abrir, criar, editar e enviar seqüências de pacotes da pilha TCP/IP, figura 6.1 (1). Atualmente, permite-se o envio de pacotes *ethernet*, ARP, IP, UDP, TCP e ICMP.

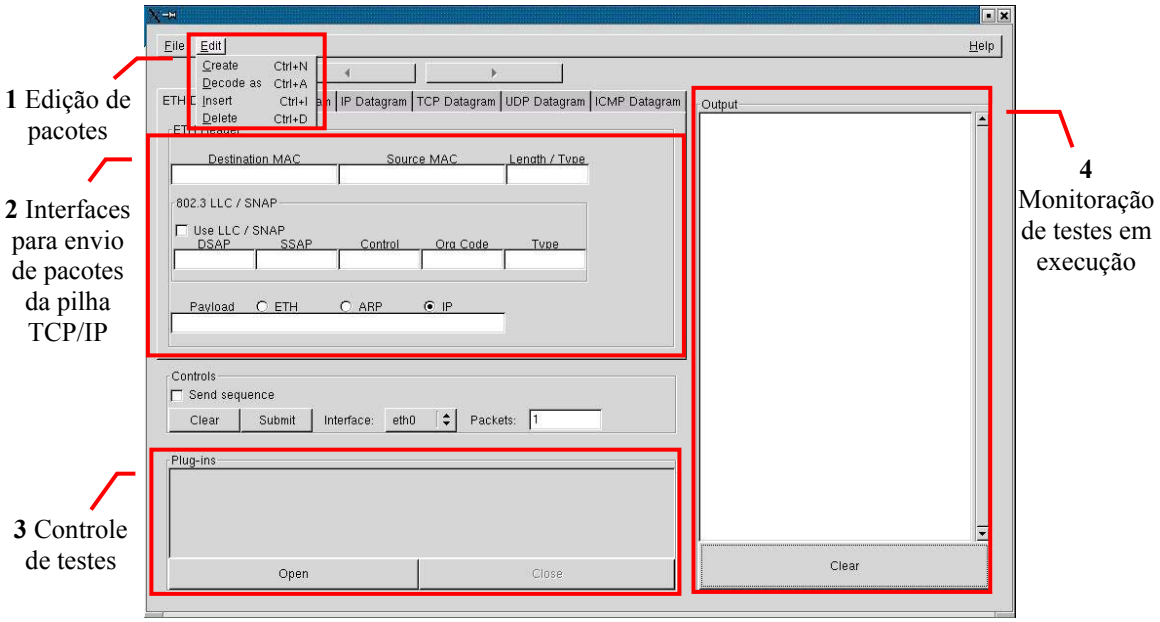

FIGURA 6.1 - Módulo de Gerenciamento

O injetor de falhas foi desenvolvido de modo a permitir o máximo de mobilidade e praticidade de uso ao usuário. Todos os cabeçalhos de protocolos implementados estão disponíveis em uma estrutura de *tags* o que facilita o deslocamento entre os diversos protocolos, figura 6.1 (2). A maioria dos campos de cada protocolo deve ser preenchido em decimal, com exceção de alguns campos como o de *checksum* e *payload* que devem ser em hexadecimal.

Não é feito nenhum tipo de consistência dos dados entrados pelo usuário, tanto de tipo de dados como de tamanho. Assim, qualquer que seja o dado entrado, o programa tentará enviá-lo. Essa característica é essencial por permitir criar os mais diversos tipos de pacotes mesmo que não estejam corretos e explorar possíveis

vulnerabilidades em sistemas e protocolos. Todos os pacotes trabalhados pela interface são salvos no formato da Libpcap, assim permitindo uma maior compatibilidade com outros sistemas e uma maior facilidade de uso da ferramenta.

Como não é realizado nenhum teste de consistência dos dados entrados pelo usuário, alguns erros podem ser gerados, sejam eles intencionais ou não. Por exemplo, pode-se criar um pacote ICMP e salvá-lo como se fosse um pacote TCP. O problema em fazer isso ocorre no momento em que se tenta abrir e decodificar o pacote novamente. Qualquer programa que suporte o formato da Libpcap abre o pacote como se fosse um pacote TCP, apesar de suas inconsistências. Assim, o usuário não poderia editar novamente seu pacote ICMP, pois ele se tornou, agora, um pacote TCP. A solução encontrada foi disponibilizar uma função que permite ao usuário especificar o tipo de decodificação que deve ser feita em um pacote, independente dos dados existentes nele, figura 6.2.

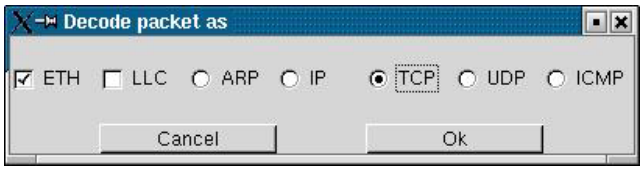

FIGURA 6.2 – Função para Decodicação de Pacote

Esta função permite a escolha opcional de um cabeçalho *ethernet* e *logical link control* (LLC), e permite a escolha do protocolo de mais alto nível. Na figura 6.2, os protocolos escolhidos foram o *ethernet* e o TCP, então o pacote é decodificado tendo cabeçalhos *ethernet*, IP (necessário pelo TCP) e o TCP propriamente dito.

Além disso, uma série de funcionalidades para a edição de seqüências de pacotes também é disponibilizada, como criação, inserção e exclusão de pacotes. A ferramenta permite que sejam criados novos pacotes ou seqüência de pacotes através de dados da interface gráfica ou de pacotes previamente salvos em arquivos. A criação pode ser feita utilizando-se de arquivos salvos em disco, figura 6.3, ou pode ser feita através da interface gráfica utilizando-se para isso das funções de inserção e exclusão de pacotes de uma seqüência de pacotes aberta, figura 6.1 (1).

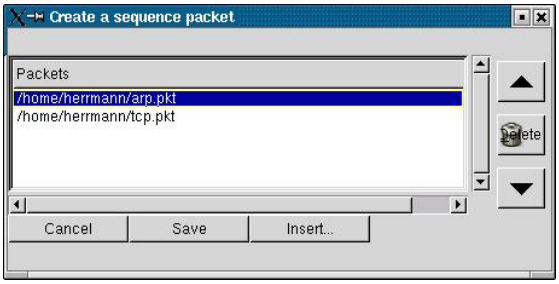

FIGURA 6.3 - Criação de uma Seqüência de Pacotes

Na figura 6.3, está-se criando uma seqüência de pacotes que consiste em enviar um pacote ARP (arp.pkt) seguido de um pacote TCP (tcp.pkt). Existe a opção de inserir mais pacotes através do botão de *insert* e de reorganizar a ordem dos pacotes dentro da seqüência através dos botões de navegação e de *delete*. Observe que os pacotes arp.pkt ou tcp.pkt não, necessariamente, se constituem apenas de um pacote ARP e TCP, respectivamente, eles próprios podem ser uma seqüência de pacotes previamente criada.

Outra funcionalidade do módulo de gerenciamento é o controle e execução dos módulos de teste, figura 6.1 (3). Ao abrir um módulo de teste (*plug-in*), figura 6.4, é

possível realizar três tipos de operações: iniciar o *plug-in*, parar o *plug-in* e pedir informações sobre o mesmo.

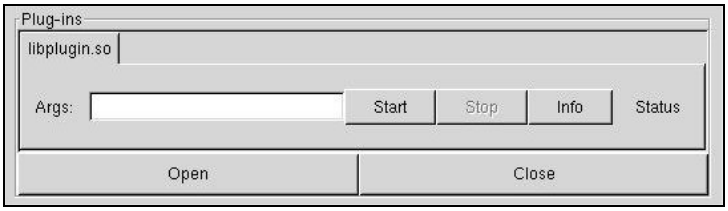

FIGURA 6.4 - Gerenciamento de Testes

Ao iniciar um *plug-in* é criada uma *thread* específica para a realização de seus testes, passando para o usuário o controle da *thread* e sendo passados como parâmetros para a mesma os dados de entrada existentes no campo Args (figura 6.4). Após o início do *plug-in*, pode ser executada a função de parada, a qual tenta finalizar o teste em execução. Entretanto, como os testes são implementados pelo usuário, não há garantias que o mesmo seja finalizado corretamente. Assim, caso não seja possível parar a execução do teste após um determinado período de tempo, a *thread* do *plug-in* é incondicionalmente finalizada. A última função refere-se à informação sobre um determinado teste, a qual obtém do *plug-in* informações sobre o seu uso e funcionamento.

Através do módulo de gerenciamento são realizados também o controle e a monitoração dos testes em execução pelo usuário, seja através do próprio módulo de gerência, seja pela utilização de módulos de teste (*plug-ins*), como pode ser visto na figura 6.1 (4). A monitoração consiste basicamente em informações sobre pacotes abertos, salvos ou enviados e sobre informações enviadas pelo *plug-in* ao módulo de gerenciamento.

6.2 Módulo de Injeção e Monitoração

O módulo de injeção e monitoração, como especificado anteriormente, fornece as funções básicas para injeção de pacotes e monitoração da rede, bem como funções mais sofisticadas para testes de sistemas de segurança. Com o intuito de alcançar esse objetivo, esse módulo foi implementado em linguagem C na forma de uma biblioteca a fim de ser facilmente utilizado tanto pelo módulo de gerenciamento como pelos módulos de teste desenvolvidos.

O módulo de injeção e monitoração fornece atualmente uma API (*Application Program Interface*) que permite realizar quatro tipos de operações:

- criar pacotes da pilha TCP/IP;
- enviar pacotes;
- monitorar pacotes na rede;
- enviar mensagens para o módulo de gerenciamento.

A API disponibilizada permite a criação de qualquer tipo de pacote, contudo fornece uma maior facilidade para a criação de pacotes *ethernet*, ARP, IP, TCP, UDP e ICMP, figura 6.5. Além de funções específicas para a criação de pacotes, também é fornecida a estrutura desses protocolos a fim de permitir que o usuário crie "manualmente" os seus pacotes. A seguir são vistas as funções para criação de pacotes da pilha TCP/IP em maiores detalhes.

```
/* convert string to hexa */
char *strtohex (char *payload, int *length_payload)

/* create an ethernet frame and return it */
char *frame_eth (int id, unsigned char *dmac, unsigned char *smac, unsigned
char *type, char *payload, int *length_payload)

/* create an logical link control frame and return it */
char *frame_llc (int id, unsigned char *dsap, unsigned char *ssap, unsigned
char *control, ..., char *payload, int *length_payload)

/* create an ARP frame and return it */
char *frame_arp (int id, unsigned char *hardware, unsigned char *protocol,
unsigned char *hlen, unsigned char *plen, ..., int *length)

/* create an IP frame and return it */
char *frame_ip (int id, unsigned char *version, unsigned char *ihl, unsigned
char *tos, unsigned char *total_length, unsigned char *identification, ...,
unsigned char *payload, int *length_payload)

/* create a frame TCP and return it */
char *frame_tcp (int id, unsigned char *ip_source, unsigned char
*ip_destination, unsigned char *source, unsigned char *destination,
unsigned char *sequence, ..., unsigned char *options, unsigned char *payload,
int *length_payload)

/* create a frame UDP and return it */
char *frame_udp (int id, unsigned char *ip_source, unsigned char
*ip_destination, unsigned char *source, ..., unsigned char *udpchecksum,
unsigned char *payload, int *length_payload)

/* create a frame icmp and return it */
char *frame_icmp (int id, unsigned char *type, unsigned char *code, unsigned
char *icmpchecksum, ..., unsigned char *payload, int *length_payload)
```

FIGURA 6.5 - Funções para Criação de Pacotes

As funções para criação de pacotes foram desenvolvidas de modo que o usuário possa criar *frames* de protocolos independentemente e assim adicioná-los a outros protocolos como se fossem seus *payloads*. Assim, para se criar um pacote TCP SYN de início de conexão, por exemplo, as seguintes funções podem ser chamadas:

1. strtohex para criar o *payload* do TCP;
2. frame_tcp para criar o pacote TCP, passando como *payload* a saída da função anterior;
3. frame_ip para criar o pacote IP, passando como *payload* a saída da função frame_tcp.

Outra vantagem dessa abordagem é a maior liberdade que o usuário possui para criação de seus pacotes sem nenhum tipo de verificação por parte da biblioteca. Assim, pode-se criar pacotes *ethernet* e adicioná-los como *payload* de um pacote TCP, por exemplo. Essa funcionalidade permite ao usuário explorar possíveis deficiências de protocolos e gerar testes bem específicos em equipamentos de rede.

Todas as funções de criação de pacotes possuem como parâmetros de entrada de um modo geral:

- um inteiro de identificação (*int id*);
- *strings* de texto para os campos do protocolo;
- um inteiro para cálculo do tamanho do pacote (*int *length_payload*).

O inteiro de identificação serve para informar quem está chamando a função. A princípio quem pode chamar essas funções são o módulo de gerenciamento e os módulos de testes. A identificação de chamada é importante para manter informações sobre a criação de pacotes por parte dos módulos de teste e, posteriormente, poder utilizar tais informações para gerar estatísticas, por exemplo.

Os *strings* de texto são utilizados para a criação de um *frame* em específico. Não há a necessidade de passar valores para todos os campos de um *frame*, mas alguns em especial são importantes: *length*, *checksum*, *payload* e *length_payload*. Esses campos são importantes pois seus valores podem influenciar de algum modo a função sendo chamada.

Os campos de *length* e *checksum* de um *frame* são os únicos campos que são calculados pelas funções de criação de pacotes, caso o usuário não especifique nenhum valor para eles. Isso permite que o usuário não precise preocupar-se em calcular o tamanho de um *frame* ou o valor de sua soma de verificação, ao mesmo tempo que não impede que o usuário entre com valores inconsistentes para a realização de testes. Todos os demais campos são preenchidos com os valores recebidos pela função, quaisquer que sejam eles.

Os campos de *payload* e *length_payload*, por sua vez, requerem um certo cuidado quanto a sua utilização, pois o incorreto uso desses campos pode invalidar o pacote sendo criado. O campo *payload* deve ser sempre fornecido através da função strtohex, caso exista um *payload*, pois seus valores devem ser informados em hexadecimal e não em formato *string* como os demais campos. Enquanto que o valor de *length_payload*, deve referenciar um inteiro global para todo o pacote sendo criado, ou seja, deve ser sempre o mesmo inteiro no decorrer das chamadas para a criação de um pacote, pois ele guarda a informação do tamanho do pacote no decorrer de sua criação desde o *payload* (strtohex) até o frame *ethernet* (frame_eth), por exemplo.

A API fornecida para o envio de pacotes consiste de quatro funções, figura 6.6:

- sendeth: envia *frames ethernet*;
- sendarp: envia *frames* ARP;
- sendip: envia *frames* IP
- sendfile: envia pacotes previamente salvos.

```
/* send an eth frame */
int sendeth (int id, const void *packet, size_t length, const char *dest_mac,
int interface)

/* send an arp packet */
int sendarp (int id, const void *packet, size_t length, const char *dest_mac,
int interface)

/* send an ip packet */
int sendip (int id, const void *packet, size_t length, const char *dest_ip)

/* send a libpcap file */
int sendfile (int id, char *file, int interface)
```

FIGURA 6.6 - Funções para Envio de Pacotes

Existem três funções para o envio de pacotes visando atingir diferentes níveis de detalhes, assim a função a ser utilizada vai depender do nível de detalhe especificado em um pacote. Além disso, é importante salientar que dependendo do sistema

operacional sendo utilizado podem existir certas verificações nos pacotes sendo enviados dependendo do nível de detalhe sendo utilizado. Por exemplo, em certos sistemas operacionais Linux, caso envie-se um pacote especificado até o nível de rede há uma verificação do cabeçalho IP, corrigindo-se erros de *checksum*. Assim, caso queira-se enviar um pacote com o *checksum* IP errado, deve-se criar um pacote até o nível de enlace.

Portanto, dependendo do quanto for especificado, deve-se utilizar uma das funções da figura 6.6. Caso tenha-se especificado apenas até o protocolo IP, pode-se utilizar a função sendip, se a especificação for do protocolo ARP ou até o protocolo *ethernet*, pode-se utilizar a função sendarp e sendeth, respectivamente. Nessas funções também é necessário informar o inteiro de identificação, para fins de controle do número e tipo de pacotes sendo enviados pelos módulos, e o tamanho total do pacote, obtido através da variável *length_payload* das funções de criação de pacotes.

Existe também a opção de enviar pacotes ou seqüência de pacotes previamente salvos. Esta função simplesmente abre um arquivo no formato Libpcap e envia todos os pacotes existentes no mesmo.

Todas as funções de envio são bloqueantes, assim caso seja enviado uma seqüência de pacotes muito grande através da função sendfile o módulo de teste que a chamou ficará bloqueado até o termino do envio de todos os pacotes existentes no arquivo.

Além das funções de criação e envio de pacotes existem mais duas classes de funções fornecidas pela API do módulo de injeção e monitoração que são uma função para monitoração de pacotes da rede e uma função para comunicação entre os módulos de testes e o módulo de gerenciamento, figura 6.7.

```
/* wait for a especific packet */
char *waitpacket (int id, char *interface, char *filter, int *length)

/* write a message to management program */
int writemsg (int id, char *msg)
```

FIGURA 6.7 - Funções de Monitoração e Comunicação

A função waitpacket é implementada utilizando-se funções de captura da biblioteca Libpcap, fornecendo uma abstração de mais alto nível para o usuário da ferramenta. Essa abstração permite que o usuário não se preocupe com vários detalhes de implementação, no caso específico, o usuário apenas necessita chamar uma única função, enquanto que seria necessária a chamada de várias funções da Libpcap. Essa função é bloqueante e escuta a rede a procura de pacotes que satisfaçam o filtro (*filter*) especificado pelo usuário. Quando um pacote preenche os requisitos definidos, o mesmo é retornado ao usuário para verificação ou tratamento.

A função writemsg, por sua vez, permite que o usuário da ferramenta mande mensagens a serem exibidas pelo módulo de gerenciamento, sejam elas para informar sobre a execução do módulo de teste ou sobre a ocorrência de eventos importantes. Ao ser chamada esta função escreve a mensagem recebida em um canal de comunicação criado entre o módulo de gerenciamento e o módulo de injeção e monitoração.

Esse canal, no caso um pipe, figura 6.8, é criado sempre pelo módulo de gerenciamento ao iniciar sua execução chamando-se uma função do módulo de injeção

e monitoração. Na chamada, é passado um número randômico que identifica quem criou o pipe. Ele é novamente necessário no término da execução do módulo de gerenciamento, quando se deseja encerrar o canal de comunicação.

Caso a chamada para criação do pipe seja bem sucedida, a função retorna uma referência para o pipe de leitura, assim permitindo que o módulo de gerenciamento receba informações do módulo de injeção e monitoração. A comunicação inversa entre o módulo de gerenciamento e o módulo de injeção e monitoração é feita através de chamadas de funções fornecidas pela API.

O número randômico é utilizado para identificar o módulo que criou o canal de comunicação. No caso, o pipe somente pode ser fechado quando o número passado na função de criação do pipe confere com o número passado para sua chamada de encerramento, ou seja, somente pode ser encerrado por quem o criou.

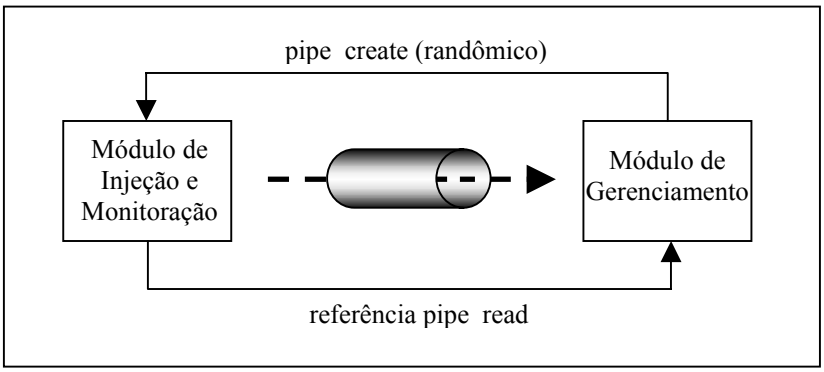

FIGURA 6.8 - Comunicação entre Módulo de Gerenciamento e Módulo de Injeção e Monitoração

6.3 Módulo de Teste

O módulo de teste, como visto anteriormente, é responsável pela implementação da biblioteca de falhas e por algumas funções do controlador e monitor do sistema de injeção de falhas. Toda a lógica dos testes em execução é determinada pelo usuário através desse módulo conforme o tipo de falha que deseja injetar. Por isso é importante que a ferramenta disponibilize uma linguagem de programação simples, eficiente e funcional que permita ao usuário flexibilidade para definir e controlar a realização de testes.

Existem duas soluções básicas para este problema. Pode-se optar por definir uma nova linguagem, nesse caso tentando otimizá-la para a execução de testes de mecanismos de segurança, através da definição de estruturas e funções específicas para esses fins, ou pode-se optar pela utilização de uma linguagem já existente com características, desvantagens bem conhecidas e toda uma infra-estrutura disponibilizada.

Entre as alternativas expostas optou-se por utilizar uma linguagem de programação já existente à criação de uma linguagem própria da ferramenta. A criação de uma linguagem própria acarretaria a necessidade de mecanismos na ferramenta para o controle, compilação e execução da linguagem, além de toda uma definição de estruturas, funções de controle e operações da própria linguagem. Além disso, a criação de uma nova linguagem não garantiria uma melhor definição de testes, nem uma melhor performance de execução dos mesmos, podendo até gerar muitas limitações em relação às linguagens existentes atualmente.

Assim, os módulos de testes são construídos utilizando-se a linguagem C e disponibilizados na forma de uma biblioteca dinâmica a ser carregada pela ferramenta. Para tanto, algumas funções padrões devem ser definidas nos módulos de teste para serem chamadas pelo módulo de gerenciamento no momento de sua execução, figura 6.9.

```
void _start  (unsigned char id, char *input)
void _stop   (unsigned char id)
char *_info  (unsigned char id)
```

FIGURA 6.9 - Funções Básicas de um Módulo de Teste

A função _start é chamada pelo módulo de gerenciamento ao iniciar a execução de um módulo de teste. São passados como parâmetros um identificador para o módulo de teste (id) e uma *string* de valores passados pelo usuário, figura 6.10 (1). Uma *thread* é criada especialmente na chamada dessa função, dessa maneira o *plug-in* do usuário pode executar em paralelo ao módulo de gerenciamento permitindo também que mais de um teste seja executado simultaneamente.

A função _stop, por sua vez, informa ao módulo de teste em execução que o mesmo deve ser encerrado. Nesse caso, o *plug-in* deve tomar as atitudes necessárias para a correta finalização dos testes em andamento. E a última função que um teste deve implementar é a de informação (_info) que consiste em fornecer um resumo explicando o teste que o *plug-in* implementa.

É importante notar que os testes são módulos independentes ao módulo de gerenciamento, ou seja, o módulo de gerenciamento não possui controle sobre os testes em execução e sobre a correta implementação dos mesmos. A única forma de controle existente é sobre as *threads* criadas para a execução dos testes, nesse caso, a única atitude possível é cancelar alguma *thread* criada, e, conseqüentemente, finalizar a execução de algum teste em andamento de forma abrupta.

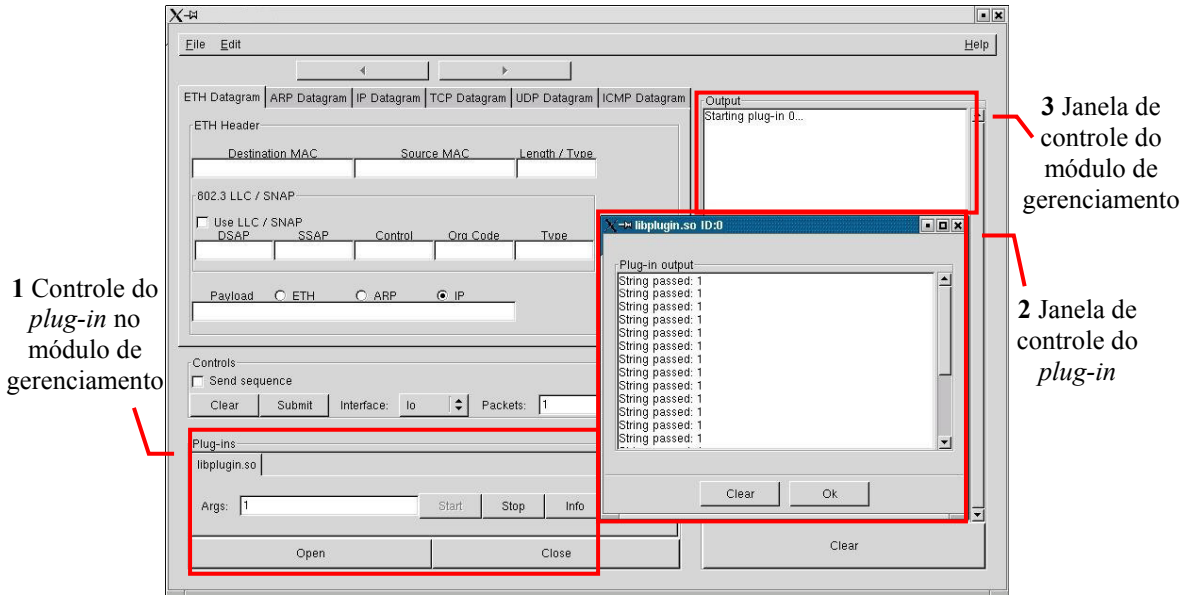

FIGURA 6.10 - *Plug-in* em Execução

Na figura 6.10, é possível visualizar a execução de um simples módulo de teste. Esse módulo simplesmente recebe uma *string* como parâmetro de entrada e a escreve na

janela de controle do teste. Ao inicializar um módulo de teste, é criada uma janela de controle onde são mostradas as saídas geradas pelo módulo, caso essa janela seja fechada, todas as saídas do módulo são redirecionadas para a janela de monitoração do módulo de gerenciamento.

```
void _main (int id, char *str)
{
  char msg[80];

  sprintf (msg, "String passed: %s\n", str);
  writemsg (id, msg);
}
void _start (unsigned char id, char *input)
{
  do
  {
    _main (id, input);
    sleep (1);
  } while (i == 0);
}
void _stop (unsigned char id)
{
  i = 1;
}
char *_info (unsigned char id)
{
  strcpy (info, "This plug-in write the input to the default output while i =
0!\n");
  return info;
}
```

FIGURA 6.11 - Funções do Módulo de Teste de Exemplo

Na figura 6.11, pode ser visto o código fonte das funções do módulo de teste de exemplo. Pode-se verificar que o controle de execução do *plug-in* é feito através de uma variável global. Quando a função para finalizar a execução do teste é chamada apenas se faz uma alteração nessa variável. Além disso, a função que escreve os parâmetros de entrada na janela de controle de teste é a função writemsg, disponibilizada pela API da ferramenta. Esse é um exemplo bem simples que serve para ilustrar o funcionamento de um módulo de teste e sua relação com os demais módulos.

6.4 Exemplo de Teste: *TCP Kill*

Nas seções anteriores, teve-se uma idéia geral da implementação do protótipo desenvolvido e das funcionalidades de cada módulo. Agora, deseja-se juntar todo esse conhecimento para o desenvolvimento de um módulo de teste um pouco mais sofisticado. O teste a ser desenvolvido visa verificar a capacidade de detecção de um IDS quanto a um ataque específico, no caso um *tcp kill*.

6.4.1 Descrição do Ataque

Esse ataque é classificado como um ataque de negação de serviço que consiste em finalizar todas as conexões TCP existentes em uma rede local, com exceção das conexões do próprio atacante. O *tcp kill* é um ataque bem simples de ser implementado e aproveita-se de algumas fragilidades do protocolo TCP:

- fraca autenticação;

- fraca integridade;
- encerramento de conexão.

A autenticação de um pacote TCP é feita através do endereço IP de origem do pacote e na coerência do pacote verificado através de uma máquina de estados de uma conexão TCP, enquanto que a integridade é verificada através de uma soma de verificação (CRC-32). Ambos os métodos são facilmente forjados, além disso um atacante utiliza-se da opção de *reset* para finalizar uma conexão TCP, uma opção válida que deve ser utilizada em três circunstâncias [POS 81]:

1. quando uma conexão não existe, um pacote de *reset* deve ser enviado em resposta a qualquer segmento TCP, exceto a outro *reset*;
2. se a conexão não está em nenhum estado sincronizado e um pacote TCP recebido confirma um segmento ainda não enviado;
3. caso um pacote recebido possua um nível de segurança ou precedência diferente ao da conexão existente.

Em todos os casos, um pacote de TCP RST somente deve ser aceito se o seu número de seqüência estiver dentro da janela de pacotes esperados para a conexão. No caso específico de inicialização de uma conexão, um RST é aceito se o seu número de seqüência estiver dentro da janela e o campo de ACK confirmar o pacote SYN.

Assim, esse ataque pode ser divido em duas fases. A primeira consiste em um atacante monitorar a rede local a espera de conexões TCP existentes e armazenar algumas informações importantes para uso futuro, como:

- IP origem;
- IP destino;
- número de seqüência dos pacotes.

E a segunda fase consiste em forjar pacotes TCP RST com as informações obtidas através da monitoração da rede local.

6.4.2 Implementação do Módulo de Teste tcp_kill

Como visto anteriormente o módulo de teste deve possuir três funções obrigatoriamente: _start, _stop e _info. A existência dessas funções é essencial, pois são elas que são chamadas pelo módulo de gerenciamento na inicialização, na finalização e na consulta de informações de um módulo de testes. Além disso, é necessária a inclusão do cabeçalho das funções disponibilizadas pelo módulo de injeção e monitoração a fim de que se possa utilizar as mesmas em um módulo de teste.

Então primeiramente vai-se declarar a função _info por ser a mais simples, figura 6.12. A função baseia-se na copia de uma *string* de explicação do módulo para uma variável global a qual é retornada quando a função é chamada.

```
char *_info (unsigned char id)
{
  strcpy (info, "This plug-in kills all tcp connections in the local
network.\n");
  return info;
}
```

FIGURA 6.12 - Função _info do Módulo de Teste tcp_kill

A função _start é o núcleo do módulo de teste e necessita realizar uma série de testes e guardar algumas informações sobre as conexões existentes, como visto anteriormente, por isso algumas variáveis auxiliares são necessárias, figura 6.13.

```
#include "libnsa.h"

int i = 0;
char info [150];
char msg[150];

int length = 0;
int length_tcp = 0;
int size = 0;
int frame_length = 0;

char *packet;
char *packet_to_send;
char *payload;

char ip_source[20];
char ip_dest[20];
char tcp_ack[20];
char tcp_sport[20];
char tcp_dport[20];

struct eth_hdr *ethernet;
struct ip_hdr *ip;
struct tcp_hdr *tcp;
struct in_addr ip_addr;
```

FIGURA 6.13 - Variáveis Auxiliares do Módulo de Teste tcp_kill

A primeira atitude da função _start após ser chamada é informar que iniciou a sua execução para o módulo de gerenciamento e iniciar um *loop* a espera de pacotes TCP existentes na rede local, figura 6.14. Para tanto, são chamadas as funções writemsg e waitpacket da API disponibilizada pelo módulo de injeção e monitoração. Pode-se observar através da função waitpacket que a interface monitorada é a eth0 e os pacotes que devem ser retornados são apenas os TCP.

```
void _start (unsigned char id, char *input)
{
  writemsg (id, "Starting tcp kill...\n");
  do
  {
    writemsg (id, "Waiting for a tcp packet...\n");
    /* Wait for a TCP packet */
    packet =  waitpacket (id, "eth0", "tcp", &length);
```

FIGURA 6.14 - *Loop* Principal do Módulo de Teste tcp_kill

A função waitpacket é bloqueante e somente retorna quando um pacote TCP for obtido. A partir desse momento, deve-se navegar entre os cabeçalhos capturados na rede a procura de informações importantes para o ataque. Assim, são copiados os endereços IP de origem e destino do cabeçalho IP e o número de identificação de seqüência, porta origem e porta destino do cabeçalho TCP para variáveis auxiliares, figura 6.15.

```
if (packet != NULL)
{
 ethernet = (struct eth_hdr *) packet;
 if (ntohs (ethernet->type) == 0x0800)
 {
  frame_length = sizeof (struct eth_hdr);
  ip = (struct ip_hdr *) (packet + frame_length);
 }
 if (ntohs (ethernet->type) <= 0x600)
 {
  frame_length = sizeof (struct eth_hdr) + sizeof (struct llc_hdr);
  ip = (struct ip_hdr *) (packet + frame_length);
 }

 /* copy ip source and destination */
 ip_addr.s_addr = ip->saddr;
 sprintf (ip_source, "%s", inet_ntoa (ip_addr));
 ip_addr.s_addr = ip->daddr;
 sprintf (ip_dest, "%s", inet_ntoa (ip_addr));

 frame_length += (ip->ihl * 4);
 tcp = (struct tcp_hdr *) (packet + frame_length);

 /* copy tcp ack, source port and destination port */
 sprintf (tcp_ack, "%lu", ntohl (tcp->ack_seq));
 sprintf (tcp_sport, "%u", ntohs (tcp->source));
 sprintf (tcp_dport, "%u", ntohs (tcp->dest));
```

FIGURA 6.15 - Obtenção de Dados Importantes para o Ataque

Após a coleta de dados do pacote capturado, inicia-se a segunda fase do ataque que consiste em forjar um pacote TCP RST para finalizar a conexão existente. Nessa segunda fase, são utilizadas as informações coletadas e as funções de construção de pacotes da API, tais como frame_tcp e frame_ip. A figura 6.16 mostra a utilização dessas funções bem como dos dados copiados anteriormente. Inicialmente, o cabeçalho TCP é criado, pois posteriormente ele é necessário para a criação do IP.

Na função frame_tcp são informados os endereços IP de origem e destino além dos demais dados necessários para a criação do segmento TCP para que a própria função calcule o *checksum* do cabeçalho. Também é passado o endereço da variável auxiliar length_tcp para que se possa calcular o tamanho final do pacote sendo montado. Essa mesma variável é passada como parâmetro para função frame_ip junto com o cabeçalho TCP, no caso a variável payload, figura 6.16.

Após o envio do pacote com o TCP RST é enviada uma mensagem ao módulo de gerenciamento informando os dados de identificação da conexão encerrada. E, no final do *loop*, é feita a liberação da memória alocada pelas funções utilizadas para evitar possíveis problemas de falta de memória, adicionalmente a variável que guarda o tamanho dos pacotes deve ser zerada.

```
   /* create tcp frame RST */
   payload = frame_tcp (id, ip_dest, ip_source, tcp_dport, tcp_sport, tcp_ack,
"0", "0", "", FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, "0", "",
"0", "", "", &length_tcp);
```

```
  /* create ip frame */
   packet_to_send = frame_ip (id, "4", "5", "0", "", "0", TRUE, FALSE, "0",
"64", "6", "", ip_dest, ip_source, "", payload, &length_tcp);

   /* send packet */
   size = send_ip (id, packet_to_send, length_tcp, ip_source);

   /* write message to manager module */
   sprintf (msg, "Kill tcp connection, sip: %s dip: %s sport: %s dport: %s\n",
ip_source, ip_dest, tcp_sport, tcp_dport);
   writemsg (id, msg);

   /* free memory */
   free (packet);
   free (payload);
   free (packet_to_send);
  }
  length_tcp = 0;

 } while (i == 0);
}
```

FIGURA 6.16 - Criação do Pacote TCP RST

Como pode ser observado o *plug-in* desenvolvido é bem simples e não faz uma série de verificações que seriam interessantes. Por exemplo, ele não verifica se o pacote TCP é um pacote de início ou término de conexão nem se o pacote é originado ou destinado à máquina do próprio atacante, ou seja, alguns pacotes são gerados desnecessariamente, devido ao atual estado da conexão (início ou término) e as conexões da própria máquina do atacante são alvos do ataque. Contudo esses testes são simples verificações que devem ser feitos no pacote capturado e que não são importantes para o entendimento do ataque ou da criação desse *plug-in*.

```
void _stop (unsigned char id)
{
  i = 1;
  writemsg (id, "Leaving tcp kill...\n");}
```

FIGURA 6.17 - Função _stop do Módulo de Teste tcp_kill

A última função necessária para este módulo de teste é a _stop. E como pode ser visto na figura 6.17 sua implementação consiste em alterar uma simples variável do *plug-in* que faz encerrar o *loop* e enviar uma mensagem ao módulo de gerenciamento informando o término da execução do teste. Outras maneiras podem ser utilizadas para encerrar a execução de um teste, mas optou-se pela simplicidade. Além disso, nada impede que outras funções auxiliares do usuário sejam implementadas em um módulo de teste, não sendo necessário se restringir às funções obrigatórias explicadas.

E, finalmente, o *plug-in* deve ser compilado como uma biblioteca dinâmica. Em um sistema operacional Linux, seriam necessários dois comandos, um para criar um objeto da biblioteca e um segundo para criar a biblioteca dinâmica propriamente dita, ligando-a as demais bibliotecas das quais é dependente, figura 6.18.

```
# Create shared library's object file, tcp_kill.o.

gcc -fPIC -Wall -g -c tcp_kill.c

# Create shared library.
# Use -lc and –lnsa to link it against C library
# and against the proposed program library, since tcp_kill
# depends on these libraries.

gcc -g -shared -Wl -o tcp_kill.so tcp_kill.o -lc –lnsa
```

FIGURA 6.18 - Criação do Módulo de Teste tcp_kill

7 Conclusão

Atualmente, a Internet está sendo utilizada com objetivos bem diferentes daqueles que eram os seus iniciais. A enorme expansão que a rede teve desde de sua criação e os serviços para os quais ela está sendo utilizada como comércio eletrônico, transações bancárias não estavam previstas. Assim, a rede teve que se adaptar as novas exigências e necessidades.

A preocupação com segurança aumentou, visto que inúmeros dados sensíveis circulam pela rede, e novas tecnologias foram criadas para sanar as dificuldades enfrentadas pela Internet, tentando evitar possíveis invasões e ataques. O correto projeto e funcionamento desses novos sistemas é fundamental para a segurança de uma rede. Por isso, técnicas e ferramentas específicas para a avaliação de sistemas de segurança em rede são extremamente importantes.

Existem diversas técnicas tanto para a injeção de falhas como para a avaliação de segurança de sistemas computacionais. A injeção de falhas pode ser realizada durante todo o ciclo de vida de um sistema, visando testar os mecanismos de tolerância a falhas implementados pelo sistema. Contudo, mesmo que o sistema tenha sido testado durante todo o seu desenvolvimento, ainda podem existir problemas que não são inerentes ao sistema projetado e que surgem na sua interação com o sistema operacional, com os protocolos que utiliza ou com outros programas. Por isso, há a necessidade de testes para a avaliação de segurança desses sistemas e, conseqüentemente, de ferramentas específicas para realizar essas avaliações.

Este trabalho propôs e implementou uma ferramenta específica para injetar falhas em sistemas de segurança em rede a fim de testá-los quanto a sua eficiência e o seu correto funcionamento. O injetor de falhas em sistemas de segurança em rede foi projetado com o objetivo de possuir uma granularidade de ataque pequena injetando ataques no nível de rede, similar aos injetores de pacotes, podendo assim testar problemas em protocolos e sistemas utilizados na rede. E, tendo recursos e funcionalidades similares aos programas de varredura de vulnerabilidades através da utilização de *plug-ins*, permitir que diversos tipos de ataque mais sofisticados possam ser realizados.

As inovações dessa ferramenta em relação as analisadas são sua interface gráfica para criação de pacotes, suas facilidades para edição de pacotes e sua capacidade de expansão através do uso de *plug-ins*. Nenhuma das ferramentas estudadas permite a criação de pacotes através de uma interface gráfica, característica essa que torna muito mais intuitivo a criação dos mesmos. Além disso, a edição de pacotes torna a criação de seqüência de pacotes para ataques e testes muito mais simples e eficiente. E, finalmente, o uso de *plug-ins* permite que a ferramenta possua uma maior flexibilidade e expansibilidade. A tabela a seguir realiza uma pequena comparação entre as ferramentas vistas no capítulo 4 e a ferramenta proposta descrita nos capítulos 5 e 6.

TABELA 7.1 - Comparação entre as Ferramentas Existentes

Ferramenta	Interface		Injeção de Falha		Protocolos Suportados		Seqüência de Pacotes		Linguagem de Programação		Monitoração de Respostas	
	Texto	Gráfico	Rede	Aplicação	TCP/ IP	Outros	Sim	Não	Sim	Não	Sim	Não
Injetores de Pacote	✓		✓		✓*			✓		✓		✓*
Scanners de Vulnerabilidade	✓	✓	✓	✓	✓	✓	✓			✓*	✓	
Ferramenta Proposta		✓	✓	✓	✓	✓	✓		✓		✓	

* de um modo geral.

A ferramenta ainda não está com todas as funcionalidades implementadas, assim, como trabalho futuro, têm-se a finalização de todos os seus recursos projetados. Adicionalmente, outros trabalhos futuros visam a realização de testes com a ferramenta, a fim de verificar os seguintes aspectos:

- aplicabilidade: a ferramenta é adequada a realização de testes de segurança em rede;
- facilidade de uso: a ferramenta é de fácil utilização, ou seja, não requer muito aprendizado para ser utilizada, exceção feita aos conceitos básicos de rede e programação;
- eficiência: a ferramenta é rápida e eficiente na execução de testes.

A partir dos resultados obtidos, modificações podem ser efetuadas para melhorar a ferramenta e melhor adequá-la ao teste de sistemas de segurança. Algumas características adicionais já estão especificadas como a criação de bibliotecas de teste e a criação de interfaces de mais alto nível para o protocolo DNS, visto que é bastante utilizado em vários testes, e para a seleção e ativação de testes existentes.

Referências

[ABN 2001] ABNT. **NBR ISO/IEC 17799**: Tecnologia da Informação: código de prática para a gestão da segurança da informação. Rio de Janeiro, 2001

[BAC 99] BACE, R. **An Introduction to Intrusion Detection & Assessment**. Scotts Valley: Infidel, 1999.

[CAM 2001] CAMPELLO, R.; WEBER, R. F. Sistemas de Detecção de Intrusão. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 19., 2001, Florianópolis. **Livro Texto dos Minicursos**. Florianópolis: UFSC, 2001. p. 1-43.

[CER 88] COMPUTER EMERGENCY RESPONSE TEAM. **CERT/CC.** 1988. Disponível em: <www.cert.org>. Acesso em: 30 mar. 2003.

[CHA 2000] CHAPMAN, D. B.; ZWICKY, E. D. **Building Internet *Firewalls***. 2nd ed. Sebastopol: O'Reilly & Associates, 2000.

[CRO 95] CROSBIE, M. **Active Defense of a Computer System using Autonomous Agents**. [S. l.]: COAST Group, 1995.

[DER 2000] DERAISON, R. **Nessus Security Scanner**. Disponível em: < http://www.nessus.org/>. Acesso em: 10 nov. 2002

[FOR 2001] FORRISTAL, J.; SHIPLEY, G. Vulnerability Assessment Scanners. **Network Computing**, [S.l.], p. 51-60, Jan. 2001. Disponível em: <http://www.nwc.com/1201/1201f1b1.html >. Acesso em: 10 nov. 2002.

[GRI 2000] GRIMES, M. **Nemesis Packet Injection**. 2000. Disponível em: <http:// www.packetfactory.net/Projects/nemesis/>. Acesso em: 10 nov. 2002.

[HAE 97] HAENI, R. E. ***Firewall* Penetration Testing**. 1997. Disponível em: <http://www.seas.gwu.edu/~reto/papers/>. Acesso em: 28 dez. 2001.

[HSU 97] HSUEH, M.; TSAI, T. K.; IYER, R. K. Fault Injection Techniques and Tools. **Computer**, New York, v. 30, n. 04, p.75-82, Apr. 1997.

[LAP 98] LAPRIE, J. Dependability of Computer Systems: from Concepts to Limits. In: IFIP INTERNATIONAL WORKSHOP ON DEPENDABLE COMPUTING AND ITS APPLICATIONS, DCIA, 1998, Johannesburg, South Africa. **Proceedings...** Johannesburg: University of the Witwatersrand, 1998.

[MCH 2000] McHUGH, J.; CHRISTIE, A.; ALLEN, J. Defending Yourself: The Role of Intrusion Detection Systems. **IEEE Software Computer**, New York, v. 17, n. 5, p. 42-51, Sept./Oct. 2000.

[POS 81] POSTEL, J. **Transmission Control Protocol (TCP)**: RFC 793. [S.l.]: IETF, Sept. 1981. Disponível em: <http://www.ietf.org>. Acesso em: 30 mar. 2003.

[PRA 96] PRADHAN, D. K. **Fault-Tolerant Computer System Design.** Upper Saddle River: Prentice-Hall, 1996.

[PUK 96] PUKETZA, N. J. et al. A Methodology for Testing Intrusion Detection Systems. **IEEE Transactions on Software Engineering**, New York, v. 22, n. 10, p. 719-729, Oct. 1996.

[RIC 2000] RICKETTS M. **SendIP**. 2000. Disponível em: <http:// www.earth.li/projectpurple/progs/sendip.html>. Acesso em: 10 nov. 2002.

[SAN 2001] SANFILIPPO, S. **Hping**. 2001. Disponível em: <http://www.hping.org>. Acesso em: 10 nov. 2002.

[SCA 2001] SCAMBRAY, J.; MCCLURE, S.; KURTZ, G. **Hacking Exposed**. 2nd ed. Berkeley: Osborne, 2001.

[SCH 96] SCHNEIER, B. **Applied Cryptography**. New York: John Wiley & Sons, 1996.

[STO 2001] STONEBUMER, G.; GOGUEN, A.; FERINGA, A. **NIST Special Publication 800-30**: Risk Management Guide for Information Technology Systems. Washington, Oct. 2001.

[TOD 99] TODD, B. **Security Auditor's Research Assistant**. 1999. Disponível em: <http://www-arc.com/sara/index.shtml>. Acesso em: 10 nov. 2002.

[WAC 2002] WACK, J.; TRACEY, M. **NIST Special Publication 800-42**: DRAFT Guideline on Network Security Testing. Washigton, Feb. 2002.

[WAN 2001] WANNER, P. **Técnicas de Injeção de Falhas e Avaliação de Segurança de Rede**. 2001. 47p. Trabalho Individual (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

[WOR 2000] WORLD WIDE DIGITAL SECURITY, INC. **Security Administrator`s Integrated Network Tool**. 2000 Disponível em: <http://www.wwdsi.com/saint/>. Acesso em: 10 nov. 2002.