Bruno Ribeiro dos Santos

# DETECÇÃO DE INTRUSOS UTILIZANDO O SNORT

Monografia apresentada ao Departamento de Computação da Universidade Federal de Lavras, como parte das exigências do curso de Pós-Graduação Latu Sensu em Administração de Rede Linux, para obtenção do título de especialista em Administração em Rede Linux.

Orientador:
Prof. Joaquim Quinteiro Uchôa

**LAVRAS**
**MINAS GERAIS – BRASIL**
**2005**

Bruno Ribeiro dos Santos

# DETECÇÃO DE INTRUSOS UTILIZANDO O SNORT

Monografia apresentada ao Departamento de Computação da Universidade Federal de Lavras, como parte das exigências do curso de Pós-Graduação Latu Sensu em Administração de Rede Linux, para obtenção do título de especialista em Administração em Rede Linux.

APROVADA em ____de _____de _____

_____
Prof. Herlon Ayres Camargo (Msc)

_____
Prof. Simone Markenson Camargo (Msc)

_____
Prof. Joaquim Quinteiro Uchôa (Msc)
(Orientador)

**LAVRAS**
**MINAS GERAIS – BRASIL**
**2005**

# Resumo

Com o surgimento da Internet, o aumento das transações eletrônicas através da mesma e os valores agregado a informação, a busca por melhores estratégias de segurança tem aumentado consideravelmente.

Por causa do valor da informação que trafega pela rede, a comunicação segura através desta tornou-se muito valiosa. Ataques podem causar prejuízos e até mesmo manchar a imagem de uma empresa. Existem diversas técnicas e métodos de proteção à informação entre eles pode-se citar, *firewalls*, *backups*, *antivírus*, sistemas de detecção de invasão entre outros.

Neste combate pela informação, seja para protegê-la ou obtê-la, o administrador deve estar preparado para vencê-lo. Conhecer as técnicas de invasão e as ferramentas de proteção é vital para um bom administrador.

Neste trabalho, faz-se uma proposta e implementa-se um servidor SDI, através da solução de código aberto *Snort*.

# Agradecimentos

A Deus, pela sua presença constante em minha vida. “TUDO POSSO NAQUELE QUE ME FORTALECE.”

Ao meu orientador, Joaquim Quinteiro Uchôa, pelo valioso e grandioso apoio, contribuição e paciência dispensados ao longo deste trabalho.

Aos meus pais e familiares, pois sempre me incentivam e apóiam.

Aos meus colegas de Curso, pelo companheirismo e apoio.

Aos demais professores do curso Pós-Graduação em Administração de Rede Linux – ARL/UFLA, por compartilharem seus conhecimentos.

E a todos aqueles que, de algum modo, me ajudaram.

# Sumário

# Lista de figuras

# Lista de Tabelas

# 1 Introdução

Atualmente, a Segurança da Informação é uma das maiores preocupações dos administradores de redes, visto que, as constantes trocas de informações e as inúmeras transações financeiras utilizando meios eletrônicos, como por exemplo, a Internet, tem crescido vertiginosamente nos últimos anos.

As pessoas sempre protegem os portões onde guardam seus tesouros e bens; não fazê-lo, geralmente, resulta em ser roubado. Os gerentes de informática têm percebido que é igualmente importante proteger as redes de comunicação contra intrusos e sabotadores, tanto interno como externo a rede. Este documento tem por objetivo implementar um servidor SDI[1](Sistemas de Detecção de Invasão) com a utilização do *Snort*.

Por causa do valor da informação que trafega pela rede, a comunicação segura através da rede tornou-se muito valiosa. Devido ao aumento das transações eletrônicas, diferentes tipos de ataques estão surgindo, tais como: *Sniffer*, *IP Spoofing* (Representação de *IP*), *Denial of Services* (Negação de Serviços), decifradores de senhas e chaves, vírus etc. Para combater estes ataques, algumas soluções de segurança estão sendo implementadas, como: criptografia, autenticação, checagem de integridade, não-repúdio, assinaturas digitais e certificados, gerenciamento de chaves e ocultação de endereço, *firewall*, SDI etc.

> "Atualmente, o administrador de rede carrega uma cruz. Não basta apenas aplicar *patchs* de correções, manter o *software* atualizado, ou implementar regras de *firewall*. Há a necessidade de algo que alerte em tempo real as tentativas de invasão [18]."

1 Intrusion Detection System (IDS)

Para ser mais especifico, detecção de invasão significa detectar o uso não autorizado ou ataque em um sistema ou em uma rede.

Os SDIs tem seu modo de operação análogo a de um antivírus. Ele inspeciona o conteúdo do tráfego da rede ou de arquivos locais para procurar e desviar possíveis ataques, exatamente como um pacote de antivírus inspeciona o conteúdo de arquivos para procurar assinaturas de vírus ou possíveis ações maldosas.

A existência de um *firewall* bem configurado na rede não diminui a importância de um SDI, isto porque, o *firewall* realiza uma inspeção de pacote limitada para determinar o acesso em sua rede. O SDI inspeciona o pacote inteiro para captar a presença de conteúdo danoso e alertá-lo de sua existência.

Um dos SDIs mais famosos no mundo do *software* livre é o *Snort*. O *Snort* é um sistema de detecção de intruso baseado em assinaturas. Ele atua comparando o tráfego de rede com as assinaturas de anomalias, detectando ataques através dessas.

Há outras alternativas quanto a SDI livre, entre elas pode-se citar o *Shadow*. O *Shadow* é constituído de pelo menos dois módulos, um sensor e um analizador. O sensor coleta o tráfego de rede e o analizador examina os pacotes em busca de anomalias e mostra os resultados para o administrador.

Análogo ao *Snort*, o *Shadow* utiliza a biblioteca *Libpcap*, que é parte do aplicativo *Tcpdump*, para capturar os pacotes. Ele utiliza um conjunto de *scripts Perl* para apresentar os resultados via servidor *web*. Segundo os desenvolvedores a chave do uso eficiente do *Shadow* está na definição inteligente dos filtros do *Tcpdump*, baseado no ambiente de rede. Maiores informações sobre o *Shadow* podem ser obtidas em http://www.nswc.navy.mil/ISSEC/CID/Install3-MS.htm.

Tendo em vista a preocupação de manter um ambiente seguro, procura-se neste documento apresentar o *Snort* como uma solução completa e avançada

enquanto SDI.

Este documento está organizado como se segue: o segundo capítulo apresenta os conceitos relacionados à segurança da informação, destacando sua importância em um ambiente computacional. Neste capítulo aborda-se os ataques mais comuns, e define-se o que é detecção de invasão, destacando a importância dos SDIs e o que pode ser realizados com eles.

O terceiro capítulo apresenta o *Snort* e seu funcionamento. Explica-se os componentes que o compõe, bem como o fluxo de pacotes dentro de sua arquitetura.

O quarto capítulo aborda a instalação e configuração do servidor SDI, detalhando os componentes que o forma e as configurações necessárias para seu funcionamento.

O quinto capítulo apresenta os testes realizados para validar o servidor, bem como os resultados obtidos.

E por último, no sexto capítulo, são apresentadas as considerações finais do trabalho, as perspectivas mais relevantes, além das impressões deixadas pelo sensor SDI.

# 2 Conceitos Relacionados à Segurança da Informação

## 2.1 Introdução

Muitos administradores de redes podem ter em mente a seguinte pergunta: Por que serei alvo de uma invasão? Essa pergunta não tem uma única resposta, os motivos são diversos.

Nenhuma rede, por menor que seja, pode descartar a possibilidade de uma tentativa de invasão. Esse fato ocorre, visto que, as invasões podem ter vários motivos. Um invasor pode simplesmente estar testando seus conhecimentos, não causando nenhum dano ao sistema, mas pode estar querendo obter vantagens através de informações capturadas, ou até mesmo causar prejuízos à empresa invadida.

No decorrer deste capítulo, procura-se de forma resumida apresentar os principais conceitos relacionados com a segurança da informação, fazendo uma trilogia das vulnerabilidades mais comuns. Será realizada uma breve introdução à invasão de computadores, definindo o que constitui uma, bem como ressaltar a importância dos SDIs.

## 2.2 Formas de ataques e invasões

Qualquer código ou programa é vulnerável à *bugs*[2], por serem projetados e escritos por seres humanos, sujeitos à falhas, toda sua produção também está.

Vulnerabilidades não se localizam somente no código fonte dos programas. Sistema mal configurado, onde o administrador utiliza as

2 Falhas exitentes em *software*, causadas por erros de desenvolvimento.

configurações padrões e sistema desatualizado, onde os *patch* de correções não são aplicados, geram muitas vulnerabilidades. A inexistência de uma política de segurança também é uma falha grave num ambiente de tecnologia da informação. Maiores informações sobre política de segurança podem ser obtidas em [28].

Qualquer sistema pode apresentar erros, e os mais graves são aqueles ligados a serviços de redes. Uma observação importante que se faz em relação à segurança é que se deve manter em execução somente os serviços necessários. Quanto menos código no ar, menos *bugs*, menos problemas de segurança. [2]

Um dos maiores receios atualmente é em relação à segurança do sistema operacional. Não existe sistema operacional imune a falhas e totalmente seguro. Existe é administrador consciente e capaz de implantar um sistema seguro através de sua correta implantação, fazendo-se uso das ferramentas de segurança disponibilizadas para este fim.

Em virtude dos fatos anteriormente relatados, existem algumas recomendações básicas para minimizar incidentes de segurança. São elas:

- Manter o sistema operacional e aplicativos atualizados, principalmente os serviços de rede disponíveis, fundamentalmente se a atualização for relativa a algum aspecto de segurança.
- Executar somente serviços necessários, desabilitando os desnecessários e removendo-os da instalação se possível.
- Usar senha com poderes administrativos somente quando a tarefa a ser executada exija tal privilégio.
- Prover segurança física é muito importante. Só deve ter acesso ao servidor pessoas autorizadas.
- Utilizar ferramentas de segurança. Se um servidor for invadido, e o invasor obtiver acesso privilegiado, não há forma de medir o estrago causado. Portanto, será necessário um tempo maior que o aceitável para apurar os

danos causados. É importante coletar os dados para uma posterior auditoria, e deve-se reinstalar o servidor.

A maioria das técnica de invasão de sistema ou de negação de serviço exploram problemas ocasionados por má configuração do sistema operacional ou dos serviços de redes. Existem várias várias formas de invasão. Apresenta-se nas seções a seguir algumas dessas formas.

### 2.2.1 *Probing*

*Probing* propriamente dito, não é uma técnica de invasão, mas sim uma forma de obter informações sobre a rede. As informações obtidas podem ser usados como base para uma possível invasão.

*Probing* pode ser utilizado para descobrir quais serviços estão disponíveis na rede, com as respectivas versões, os sistemas operacionais utilizados, possíveis estações de gerência de redes, SDI, *honeypots* etc. As técnica de realização de *probing* podem ser facilmente detectadas ou impedidas caso haja a correta implementação de mecanismo de segurança na rede.

### 2.2.2 *Trojan Horses*

*Trojan Horses* são programas de computadores que se propõem a realizar determinadas tarefas e sem que o usuário saiba, executa alguma outra, que normalmente afeta a segurança do sistema.

As tarefas obscuras do *trojan* vão desde permitir que terceiros acompanhem ou tomem conhecimento do que se digita no teclado, como dados de acesso (senhas e *logins*), até abrir portas de acesso, gerando uma ótima oportunidade para o invasor.

### 2.2.3 *Backdoors*

Normalmente um invasor procura garantir uma forma de retornar a um computador invadido, sem precisar utilizar-se dos métodos empregados na

realização da invasão. Na maioria dos casos, o intuito do invasor é poder retornar ao computador invadido sem ser notado. Portanto, o invasor instala um programa no computador que permitirá acesso a qualquer momento.

O nome dado ao programa que abre esta porta para futuros acessos é *backdoor*. O *backdoor* permitirá acesso ao sistema mesmo que a falha que originou o primeiro acesso seja corrigida, isso é claro se o *backdoor* não for removido.

Um *backdoor* pode ser escrito na forma de programa, *script*, ou até como uma série de procedimentos (criação de uma conta com acesso de administrador etc), entre outras maneiras.

### 2.2.4 *Buffer Overflow*

Um *buffer overflow* é resultado do armazenamento em um *buffer* uma quantidade maior de dados do que sua capacidade.

As ameaças de *buffer overflow* são consideradas ameaças críticas de segurança, apesar de ser uma falha bem conhecida e bastante séria, que se origina exclusivamente da falta de conhecimento do programador no desenvolvimento seguro de um *software*.

Este tipo de vulnerabilidade tem sido largamente utilizado para invasões remotas de computadores, onde o invasor consegue comprometer a estabilidade do sistema.

Existem diversos tipos de ataques *buffer overflow*, entre eles *stack smashing attacks*, ataques contra *buffers* que se encontram na pilha (*stack*), e *heap smashing attacks*, que são ataques contra *buffer* que se encontrão no *heap*. Técnicamente *buffer overflow* é um problema com a lógica interna do programa, mas a exploração dessa falha pode levar a sérios prejuízos.

O objetivo de uma exploração contra um programa privilegiado vulnerável a *buffer overflow* é conseguir acesso de tal forma que o invasor

consiga controlar o programa atacado, e se o programa possuir privilégios suficientes, ou seja, se ele possuir a funcionalidade *flag suid root*, controlar a máquina.

Na maioria das vezes, o invasor tenta subverter as funções de programas que são do superusuário, executando um código arbitrário que dê algum retorno aproveitável.

O princípio é estourar o *buffer* e sobrescrever parte da pilha, alterando o valor das variáveis locais, valores dos parâmetros e/ou o endereço de retorno. Altera-se o endereço de retorno da função para que ele aponte para a área em que o código que se deseja executar encontra-se armazenado (código malicioso dentro do próprio *buffer* estourado ou até algum trecho de código presente no programa vulnerável). Pode-se assim executar código arbitrário com os privilégios do usuário que executa o programa vulnerável. *Daemons* de sistema, ou aplicações que rodam com privilégios de super-usuário são portanto alvo preferencial. Maiores informações sobre *buffer overflow* podem ser encontradas em [1] e [11].

### 2.2.5 *Exploits*

*Exploits* são códigos de programas desenvolvidos especialmente para explorar falhas introduzidas em aplicativos por erros involuntários de programação.

Esses *exploits*, que podem ser preparados para atacar um sistema local ou remotamente, variam muito quanto à sua forma  e poder de ataque. Pelo fato de serem peças de código especialmente preparadas para explorar falhas muito específicas, geralmente há um diferente *exploit* para cada tipo de aplicativo, para cada tipo de falha ou para cada tipo de sistema operacional.

Os *exploits* podem existir como programas executáveis ou, quando usados remotamente, podem estar ocultos, por exemplo, dentro de uma mensagem de correio eletrônico.

### 2.2.6 *Password Crackers*

*Password Crackers* são programas desenvolvidos para quebrar senhas de usuários ou programas. Este tipo de programa pode ser utilizado para auxiliar o administrador à descobrir senhas fracas existentes no sistema, como também pode ser usado por um invasor para descobrir senhas de usuários, e usá-las sem que o proprietário perceba. Em geral, *Password Crackers* tentam descobrir a senha do usuário através de um processo de força bruta.

Nos sistema que utilizam senhas sombreadas (*shadow*), como o *Linux*, as senhas são armazenadas de forma codificada, ou seja, a senha armazenada é o *digest* de uma função *hash* que tem como semente a senha informada.

Nos sistemas **NIX*[3] mais recentes, as senhas sombreadas são armazenadas no arquivo */etc/shadow*, que só pode ser lido pelo usuário *root*. Para maiores informações sobre senhas sombreadas consulte [26].

Quando o usuário *loga* na máquina, o sistema executa a função *hash* sobre a senha e compara o *digest* com a senha sombreada armazenada no arquivo */etc/shadow*. O ataque de força bruta nada mais é que a aplicação dessa função *hash* sobre uma grande quantidade de palavras (dicionário), comparando o resultado desta aplicação com a senha sombreada até a comparação coincidir ou esgotar as palavras do dicionário.

Geralmente os *password crackers* são lentos, e a sua eficiência depende inteiramente da qualidade das senhas. Senhas difíceis para um *password crackers* são aquelas que possuem letras, números e outros tipos de caracteres.

A senha constitui a chave principal de entrada no sistema, a descoberta da senha por pessoas diferente da proprietária caracteriza um risco para o sistema. Em virtude disso, recomenda-se que o administrador do sistema force o usuário a utilizar senhas fortes, para que a segurança do sistema não seja comprometida.

3 Distribuições *Linux* e varantes *UNIX*

A maioria dos sistemas *NIX utilizam o *PAM* (*Plugglable Autentication Modules*) que é um conjunto de bibliotecas que permite autenticar usuários, gerenciar contas e controlar recursos dos usuários no sistema.

Com o *PAM* é possível configurar a autenticação individual de cada aplicativo. A maioria dos aplicativos que requer autenticação tem suporte ao *PAM*.

O *PAM* permite a configuração de vários níveis de acesso. Como o próprio nome diz o *PAM* é baseado em módulos, tendo cada módulo uma funcionalidade. Para maiores informações sobre o *PAM* consulte [16], [24] e [27].

Existe um módulo *PAM* chamado *pam_passwdqc* que controla a qualidade da senha, ou seja, com a implementação deste módulo impede-se o usuário de cadastrar senhas fracas. Recomenda-se a implementação desde módulo nos sistemas *Linux* que oferecem suporte ao *PAM.* Informações adicionais sobre este módulo *PAM* podem ser obtidas em [10].

### 2.2.7 *DoS* (*Denial of Service*)

Os ataques conhecidos como *denial of service* são caracterizados por uma tentativa explícita do atacante de impedir que um usuário legítimo utilize determinado serviço. Algumas estratégias utilizadas nesses ataques são:

- Inundar uma rede para impedir que usuários legítimos façam uso dela;
- Impedir ou romper a conexão entre duas máquinas visando impedir o acesso a um serviço;
- Impedir o acesso a um determinado serviço;
- Impedir ou negar um serviço a um sistema ou pessoa específica.

Ataques do tipo *DoS* geralmente não comprometem a privacidade dos dados. A finalidade de um ataque *DoS* é tirar um serviço, servidor, computador ou até mesmo uma rede do ar. Ataques do tipo *DoS* muitas vezes são utilizados

para “atrapalhar” ou desacreditar um serviço.

Qualquer maneira de tirar um computador, serviço ou rede do ar pode ser considerado um ataque *DoS*. Como exemplo, pode-se citar um ataque com o intuito de gerar milhares de *logs* até encher o disco ou partição. Se o administrador colocar os *logs* na mesma partição do sistema , tal ataque gerará um problema pois todo espaço do disco será ocupado.

Alguns administradores implementam o bloqueio da conta caso ocorra n tentativas de *login* incorretas. Neste caso, um atacante pode gerar n tentativas erradas de *login* até que o bloqueio da conta ocorra. Forçar o travamento de uma conta destas é considerado um ataque *DoS*, principalmente quando esta conta é a usada por algum serviço, pois se a conta é bloqueada, o serviço sairá do ar. Esse é um dos motivos pelo qual recomenda-se que conta usada para controlar serviços seja configurada para não logar, definindo seu *shell* como */dev/null*.

Ataques para tirar redes ou servidores do ar também são possíveis. Esses ataques consistem em enviar para um determinado serviços, pacotes *TCP* com uma sinalização de “urgência”, com o conteúdo do pacote composto por caracteres inválidos. Esse ataque *DoS* ficou conhecido como (*Out of Band Data*). Atualmente, a grande maioria das pilhas *TCP/IP* é protegida contra esses tipos de ataques ou variações. Se a quantidade de informação inválida for realmente muito grande, ainda existe a possibilidade de tirar do ar o computador, ou até mesmo inundar o *link*.

Para se obter uma grande quantidade de pacotes, ataques do tipo *DoS*, foram atualizados para ataques do tipo *DoS* distribuídos, conhecido com *DDoS* (*Distributed Denial of Service*). Como já relatado, o ataque *DoS* consiste em enviar milhares de pacotes ou requisições de serviços em um dado momento. Como gerar essas requisições de um único ponto é praticamente impossível, esse ataque é distribuído entre várias máquinas, ou seja, são várias máquinas gerando requisições para um único servidor.

Os ataques de *denial of service* surgiram explorando falhas de implementação em serviços e sistemas operacionais, como *Ping-of-dead* [5] e, em alguns casos, até mesmo a forma de funcionamento dos protocolos, como no caso do *SYN Flooding* [6] e do *UDP packet storm* [7]. Num segundo momento, surgiram os amplificadores de ataque, tais como os ataques *smurf* [8] utilizando técnicas de *IP Spoofing* [4]. Mas detalhes sobre *spoofing* serão vistos na seção 2.2.8.

Com o passar dos anos todas as técnicas de ataques conhecidas (*spoofing, flooding,* amplificadores de ataques etc.) acabaram por ser incorporadas nos ataques *DoS*, e esses, por sua vez, realizados de forma distribuída. Para se chegar a esse nível de contaminação, o conceito utilizado foi de vermes, através do qual uma vulnerabilidade explorada na máquina do usuário injeta um código malicioso que aguarda as ordens do atacante sobre o alvo a ser atingido e os ataques a serem realizados.

Contra esses tipos de ataques, existem poucas medidas, principalmente se o objetivo do atacante for realizar um ataque *DDoS* por inundação de banda. Contudo, um bom *firewall* pode dificultar bastante a eficácia de um ataque desse. Algumas regras básicas de filtragem em *firewall* para evitar ataques *DDoS*:

- Filtrar qualquer tráfego *ICMP* entrando ou saindo da rede;
- Filtrar qualquer tráfego entrando na rede em portas que não estão em uso;
- Filtrar qualquer tráfego saindo da rede, em computadores que não precisam emitir tal tráfego;

A regra básica é impedir tráfego não autorizado, não só entrando na rede, mas também, a partir dela, de forma que os computadores em sua rede interna não possam ser usados como agentes.

### 2.2.8 *Spoofing*

A técnica de *spoofing* consiste na falsificação do remetente da

mensagem de um pacote de transmissão de dados, para que o receptor o trate como se fosse de um outro utilizador. Em certos sistemas, e com a intenção de obter um melhor nível de segurança, o servidor de rede só deixa utilizar certos serviços a um número restrito e autenticado de clientes. O método encontrado para furar este esquema é o de falsificar o remetente dos pacotes de dados que viajam na rede.

A técnica de *spoofing* também é utilizada por invasores para dificultar a descoberta da origem da invasão, uma vez que, as informações de origem dos pacotes são forjadas.

Os principais e mais largamente utilizados tipos de *spoofing* são: *IP Spoofing*, *ARP Spoofing* e *DNS Spoofing*. Como a técnica de *spoofing* constitui a falsificação de alguma informação, o *IP Spoofing* "falsifica" o *IP*, fingindo ser outro diferente do original. A técnica de *ARP Spoofing* consiste na falsificação do *MAC address*, redirecionando o tráfego para o impostor. No *DNS Spoofing*, o invasor compromete o servidor *DNS* e explicitamente altera as tabelas de endereço de *IP – hostname*, direcionando uma consulta *DNS* para outro servidor que não seja o "proprietário" do nome.

## 2.3 Detecção de Invasão

Detecção de invasão, refere-se ao ato de descobrir uma entrada não autorizada em um computador ou em uma rede. Esse acesso não autorizado, ou invasão é uma tentativa de comprometer ou danificar de alguma forma outros dispositivos da rede.

Um SDI é equivalente a um alarme contra ladrões configurado para monitorar pontos de acesso, atividades hostis e invasores conhecidos.

O modo mais simples de definir um SDI seria descrevê-lo como uma ferramenta capaz de inspecionar o *payload* dos pacotes a procura de

correspondências de ataques. Além desta característica um SDI tem a capacidade de ler e interpretar o conteúdo de arquivos de *logs* de roteadores, *firewalls*, servidores e outros dispositivos de rede.

As correspondências de ataques são descobertas através da comparação dos dados do pacote com assinaturas de ataques armazenadas em banco de dados. Neste ponto, o SDI pode emitir alarmes ou alertas, adotar vários tipos de ações automáticas, variando desde a desativação dos *links* Internet ou de servidores específicos, até a ativação de rastreadores, e coletas de informações forenses.

Para ser mais específico, detecção de invasão significa detectar o uso não autorizado ou ataques em um sistema ou em uma rede. Um SDI é projetado e usado para detectar e, então, desviar ou deter tais ataques ou uso não autorizado de sistemas, redes e recursos relacionados.

Os SDIs podem ser divididos em baseados em rede, baseados em *host* e distribuídos. Os SDIs que monitoram *backbones* de rede e procuram assinaturas de ataque são chamados de SDIs baseados em rede, enquanto aqueles que operam em *hosts* defendendo e monitorando os sistemas operacionais e sistemas de arquivos contra sinais de invasão, são chamados de SDIs baseados em *host*. Os grupos de SDIs que funcionam como sensores remotos e se reportam a uma estação de gerenciamento central são conhecidos como SDI distribuídos (SDIDs).

O nome sistema de detecção de invasão evoca a visão de um dispositivo situado no perímetro de uma rede, alertando-o para a presença de invasores. Embora seja uma aplicação válida, de modo algum é a única. O SDI também pode desempenhar um papel importante em uma arquitetura de defesa aprofundada, protegendo serviços internos, além de atuar como uma defesa de perímetro. Muitas funções internas de sua rede podem ser monitoradas quanto a segurança e a compatibilidade. Pode-se citar:

- Monitoração do acesso a banco de dados;
- Monitoração da funções de DNS;
- Proteção do servidor de e-mail.

Os SDIs podem servir a muitos propósitos em uma arquitetura de defesa aprofundada. Além de identificar ataques e atividade suspeita, pode-se usar dados do SDI para identificar vulnerabilidades e fraquezas na segurança.

Eles podem impor um plano de segurança. Por exemplo, se o plano de segurança proíbe o uso de aplicativos de compartilhamento de arquivo, como *Kazaa*, *Gnutella*, ou serviços de troca de mensagens, como *IRC* ou *Instant Messenger*, pode-se configurar o SDI para detectar e relatar o uso desses *software.*

Os SDIs são uma fonte de evidências valiosas. Os *logs* de um SDI podem se tornar uma parte importante no tratamento de incidentes de computador. Os sistemas de detecção são usados para captar ataques internos, monitorar tráfego de saída, e podem ser usados também como ferramentas de gerenciamento de incidentes para rastrear um ataque.

Um SDI baseado em rede (SDIR) pode ser usado para registrar e correlacionar atividades de rede maldosas. O SDIR é furtivo e pode ser implementado para monitorar passivamente ou para reagir a uma invasão.

A configuração de um SDI deve apresentar algumas propriedades desejáveis, entre elas pode-se citar as seguintes:

- Funcionar continuamente sem supervisão humana. O sistema deve ser confiável o suficiente para permitir que processe em *background;*
- Ser tolerante a falhas no sentido de que precisa resistir a um impacto no sistema e não ter sua base de informações corrompidas;
- Impor o mínimo de sobrecarga ao sistema;
- Observar desvios do comportamento normal;
- Ser facilmente adaptável ao sistema em questão. Cada sistema tem um

padrão de uso diferente, e o mecanismo de defesa deve se adaptar a essas condições;

- Lidar com a mudança do sistema, seja pela correção do sistema existente, seja pela adição de novos aplicativos ao mesmo. O perfil do sistema mudará e o SDI deve seguir esta mudança;
- Apresentar alarmes condizentes com o tráfego analisado.

O último ítem levanta a questão dos tipos de erro prováveis de acontecer no sistema. Este podem ser categorizados em falsos positivos, falsos negativos e erros de subversão.

Os falsos positivos acontecem quando o SDI gera uma alerta quando não devia. Basicamente, um falso positivo é um alarme falso. No lado oposto, pode-se obter falsos negativos. Em outras palavras, alguém compromete um sistema monitorado por um SDI e o SDI não detecta isso. Falsos negativos podem aparentar uma falsa idéia de segurança.

Erros de subversão são mais complexos e parecidos com falsos negativos. Um intruso pode usar o conhecimento sobre a operação interna do SDI para alterar o seu estado, possivelmente permitindo comportamento anômalo do sistema. O intruso pode então violar a segurança.

## 2.4 Importância dos SDIs

Nenhuma rede por menor que seja pode ser deixada sem proteção. Qualquer rede pode ser alvo de ataques, não importando seu tamanho. A capacidade de saber quando um invasor está envolvido em uma exploração ou outra atividade maldosa pode significar a diferença entre estar ou não comprometido.

Uma importante ferramenta para aumentar o nível de segurança de uma rede é o SDI. Os SDIs apresentam várias funcionalidades que auxiliam o administrador, entre elas apresenta-se as seguintes:

- Detectar o uso de *ICMP* e outros tipos de varreduras de exploração de rede que poderiam indicar um ataque iminente.
- Emitir alertas ao administrador do êxito de um comprometimento, oferecendo a oportunidade de implementar ações atenuantes, antes que um dano maior seja causado.
- Fornecer ao administrador de segurança uma janela que mostra o funcionamento interno da rede. Os SDIs fornecem o microscópio necessário para detectar os invasores. Sem a ajuda do SDI, um administrador de segurança fica vulnerável às explorações e só saberá da sua presença depois que um sistema falhar ou que um banco de dados estiver corrompido.
- Identificar vulnerabilidades e fraquezas em seus dispositivos de proteção de perímetro; por exemplo, *firewall* e roteadores.
- Reunir informações úteis para auditorias.
- Auxiliar a implantação de uma política de segurança, uma vez que, através de seus *logs* é possível descobrir quem está burlando as regras de segurança.
- Interromper ataques à rede, enquanto alerta os administradores quanto a sua presença.
- Detectar tentativas de *login* de administrador falhas e reconhecer programa de quebra de senhas.
- Alertar o administrador do sistema para possíveis brechas de segurança.

A existência de um *firewall* na rede não diminui em nada a importância de um SDI no plano de segurança. Um *firewall* pode ser configurado para detectar certos tipos de invasões, como uma tentativa de acessar a porta 27374 da entrada indireta tipo cavalo de tróia do *SubSeven*. Além disso, ele poderia ser configurado para gerar um alerta para qualquer tentativa de penetração em sua rede. No sentido mais restrito, essa seria uma função de um SDI. Entretanto, é pedir demais da tecnologia para que simplesmente determine o que pode e o que não pode entrar ou sair da sua rede, sem esperar que ela analise o conteúdo

interno de cada pacote. Um *firewall* deve ser uma parte integrante de sua defesa aprofundada, sendo sua principal função a de porteiro.

## 2.5 Cuidados necessários com um SDI

É um ledo engano pensar que instalando um SDI a rede estará segura e que um SDI é imbatível contra invasões. Um SDI também esta sujeito a ataques. Provavelmente o administrador de um SDI desejará acessar o sistema de forma remota (*SSH*) e provavelmente desejará armazenar os alertas em um banco de dados (*MySQL* ou *Postgres*). Além disso, provavelmente haverá o desejo de ver os alertas com uma interface elegante, que poderá exigir um servidor *Web* (*Apache*).

Com base nessas informações, haverá várias portas abertas no SDI: *SSH* (porta 22), *HTTP* (porta 80), *HTTPS* (porta 443) e possivelmente *MySQL* (porta 3306) ou *Postgres* (porta 5432). Agora, qualquer um com acesso a uma rede pode usar *NMAP* e *portscan* com um *sniffer* diretamente em sua interface não-promíscua.

Isso torna o SDI exatamente como qualquer outro aplicativo, portanto, deve-se ficar atendo aos anúncios de vulnerabilidades de segurança e nos anúncios de segurança do sistema operacional. Vale ressaltar também que o SDI é um *software* como outro qualquer, podendo conter vulnerabilidades. Em virtude disso, recomenda-se manter o *software* de SDI sempre atualizado bem como sua lista de assinaturas de ataques.

Isso é algo que deve ser tratado, além de certificar-se das atualizações dos aplicativos e da correta configuração e atualizações do *Kernel* do sistema. O administrador deve certifica-se de fazer o básico. Além das recomendações apresentadas na seção 2.2, as seguintes devem ser observadas na implementação de um SDI:

- **Manter a integridade do sistema.** O *Tripwire* é um aplicativo para ajudar a

garantir a integridade dos arquivos e diretórios críticos do sistema, identificando todas as alterações ocorridas neles. Há muitos outros aplicativos como o *Tripwire*, entre eles pode-se citar o *AIDE*. Para maiores informações sobre esses aplicativos consulte respectivamente [19] e [20].

- **Filtrar os serviços utilizados.** Utilizar um *firewall* bem configurado para filtrar as portas dos serviços utilizados e certificar-se de realizar a correta configuração dos serviços disponibilizados. *Iptables* é o *firewall* mais utilizado no mundo *Linux*, apesar de existir outras alternativas como *Ipchain.* Informações sobre eles podem ser obtidas respectivamente em [25] e [22].
- **Criptografar e usar autenticação de chave pública o máximo que puder.** Sempre que possível deve-se substituir programas que enviam informações em texto plano, por programa que criptografe os dados, principalmente se houver o envio de dados confidenciais como senhas de usuários.

É fundamental implementar as recomendações sugeridas. Um SDI deve ser o mais seguro possível, pois qualquer invasão em um SDI invalida todos os seus alertas de segurança, além de fornecer uma falsa sensação de segurança.

# 3 O *Snort* e seu Funcionamento

## 3.1 Introdução

Atualmente, o administrador de rede vive um dilema. Não basta apenas aplicar *patchs* de correções, manter os *softwares* atualizados, ou implementar regras de *firewall*. Há a necessidade de algo que alerte em tempo real as tentativas de invasão[18].

Para ser mais especifico, detecção de invasão significa detectar o uso não autorizado ou ataque em um sistema ou em uma rede.

Um dos SDIs mais famosos no mundo do *software* livre é o *Snort*. O *Snort* é um sistema de detecção de intruso baseado em assinaturas. Ele atua comparando o tráfego de rede com as assinaturas de anomalias, detectando ataques através dessas.

Este capítulo aborda aspectos relevantes na implementação de um sistema de detecção de intrusos baseado no *Snort* bem como sua forma de operação. Será apresentado sua arquitetura e o tratamento dado ao pacote de rede dentro desta. Pretende-se também apresentar como é criada e constituída uma assinatura do *Snort*.

## 3.2 O que é *Snort*

O *Snort* é um sistema de detecção de invasão baseado em rede (SDIR) de código fonte aberto, possuindo muitos recursos. Ele é *sniffer* que tem como diferencial a capacidade de inspecionar o *payload* do pacote, área que contém os dados do mesmo, fazendo o registros dos pacotes, além de detectar as invasões.

Fornecer funções de farejamento (*sniffer*) e registros de pacotes é a parte fundamental do *Snort*, sendo sua grande vantagem os recursos de detecção

de invasão, que fazem a correspondência do conteúdo de cada pacote com as regra[4] de invasão.

O *Snort* é um sistema avançado capaz de detectar quando um ataque está sendo realizado e, baseado nas características do ataque, alterar ou remodelar sua configuração de acordo com as necessidades, e até avisar ao administrador do ambiente sobre o ataque [2].

Considera-se o *Snort* um SDI *ligthweight*, ou seja, ele pode ser colocado em funcionamento com muito pouco esforço tanto no sentido computacional quanto no sentido configuração e suporte.

Um característica marcante do *Snort* é a facilidade de criação de assinaturas de ataques. Profissionais da área de segurança da informação sabem da velocidade que os eventos ocorrem no mundo virtual. A prontidão, exatidão e rapidez são assuntos essenciais em se tratando de segurança. Como o *Snort* é um sistema de código aberto e apresenta linguagem fácil de criação de assinaturas, permite a pronta criação e adição de uma nova assinatura de ataque por parte do administrador.

O *Snort* é uma ótima ferramenta, mas como todo aplicativo deve ser bem implementada. Devem ser aplicadas somente as assinaturas de ataques relevantes a realidade da rede. Por exemplo, não é necessário monitorar ataque a banco de dados *MySQL* se ele não se encontra em uso, o que ocasionaria apenas alertas irrelevantes. Além disso o banco de assinaturas deve estar constantemente atualizado evitando assim que ataques passem desapercebidos. O administrador tendo esses cuidados evita ou minimiza os falsos positivos e falsos negativos.

O profissional responsável pela rede deve certificar-se que o sistema onde o *Snort* esta operando é o mais seguro possível, ou seja, deve-se esta atendo a segurança do sistema operacional onde o *Snort* esta rodando. Uma

4 Uma regra é um conjunto de requisitos que geram uma ação do SDI.

invasão à máquina SDI invalida qualquer alerta do mesmo.

O *Snort* é um sistema que não requer grandes recursos de *hardware*. Os recursos de *hardware* necessário para o sistema operacional em uso atende ao *Snort*.

Recomenda-se duas interfaces de redes, uma para trabalhar em modo promíscuo, sendo o sensor do *Snort* e outra para conectividade de rede típica (*SSH*, serviços da *Web* etc.). As interfaces devem ser da velocidade máxima da rede, se a rede é 10/100 Mbps a interface deve ser de 100 Mbps.

O *Snort* foi projetado para ser executado em uma grande gama de sistemas entre eles *Linux*, *FreeBSD*, *NetBSD*, *OpenBSD*, *Solaris*, *MacOS*, *Windows*, entre outros.

Existem alguns programas opcionais que podem ser instalados para melhorar/facilitar a administração do sistema. Cita-se os aplicativos para o mundo *Linux*. Os *softwares* são:

- *MySQL*, para armazenamento dos alertas em banco de dados;
- *Guardian*, para leitura em tempo real dos *log* do *Snort*, e bloqueio via *firewall*;
- *Iptables*, para bloqueio de ataques;
- *Smbclient*, caso pretenda enviar mensagens *WinPopup*;
- *Apache*, para disponibilização de informações via *Web*;
- *PHP*, se houver *plug-ins* que o exija;
- *OpenSSH*, para acesso remoto;
- *Apache* com recursos *SSL* para monitoramento;
- *ACID* (*Analysis Console for Intrusion Detection*), para apresentar de forma mais inteligível os *logs* do *Snort* armazenados em banco de dados.

A arquitetura do *Snort* é composta de 4 componentes básicos: o farejador, o pré-processador, o mecanismo de detecção e os *plugins* de saída. Em sua forma mais básica o *Snort* é um farejador de pacotes. Entretanto, ele é

projetado para pegar pacotes e processá-los através do pré-processador e depois verificar esses pacotes com relação a uma série de regras. A figura 3.1 oferece uma visão de alto nível da arquitetura do *Snort*.

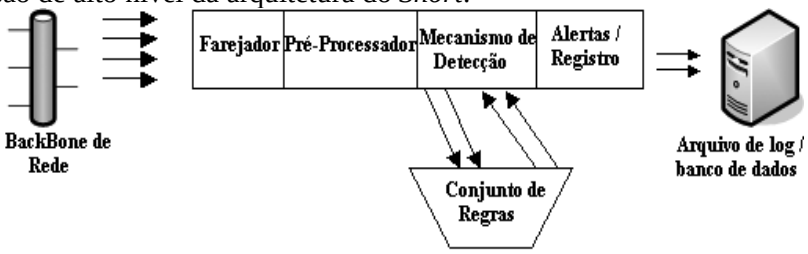


**Figura 3.1** – Arquitetura do *Snort*

O *Snort* em uma visão mais simples pode ser comparado a um separador de moedas mecânico.

1. Ele pega todas as moedas (pacotes da rede);
2. Depois, ele as envia para determinar se são moedas e como devem avançar (o pré-processador);
3. Em seguida, ele ordena as moedas de acordo com o tipo. Isso serve para o armazenamento de moedas de 50, 25, 10 e 5 centavos. (no Snort, esse é o mecanismo de detecção);
4. Finalmente, é tarefa do administrador decidir o que fazer com as moedas (no *Snort*, registrar, armazenar em banco de dados, alertar e/ou ativar regras dinâmicas).

## 3.3 Explorando os recursos do *Snort*

O *Snort* apresenta vários recursos que o torna muito poderoso, tais como: farejamento de pacotes, registro de pacotes e detecção de invasão. Ele utiliza uma linguagem de regras flexível para descrever o tráfego que ele deve analisar ou deixar passar, e também um mecanismo de detecção que utiliza uma arquitetura modular de *plugins*, que registram os ataques de diversas maneiras, dentre elas em banco de dados (*MySQL*, *PostgreSQL*, *Oracle* etc), em arquivo

binário no formato do *tcpdump*, em arquivo texto e no *syslog*.

O processo de detecção é baseado na análise dos pacotes que trafegam na rede, comparando-os com uma base de assinaturas de ataques conhecidos. Esta base de assinaturas é baseada num conjunto de regras, que são consultadas pelo *Snort* no momento da análise do pacote, sendo utilizadas de acordo com a definição no arquivo de configuração.

As regras já acompanham o *Snort*, sendo apenas necessário mantê-las atualizadas, entretanto, o administrador pode escrever suas próprias regras, assim como alterar as já existentes de forma a acrescentar mais funcionalidades a elas.

O *Snort* pode ser considerado um sensor, podendo monitorar diferentes pontos da rede ao mesmo tempo. Entre esses pontos pode-se citar os seguintes:

- Normalizar requisições *HTTP*;
- Detectar ataques do tipo *Unicode*;
- Detectar *Buffer Overflow*;
- Detectar *portscan*;
- Remontar os segmentos *TCP*;
- Ativar regras dinamicamente (regras podem ser ativadas por outras regras).

## 3.4 Componentes do *Snort*

A base do *Snort* está montada em cima da biblioteca *Libcap*. Esta provê funções de acesso a recursos de rede de baixo nível, como monitoramento do sistema, *debugging* de rede entre outros.

A figura 3.2 apresenta e ajuda a esclarecer os componentes que compõem o *Snort* e oferece uma visão de alto nível do processo do *Snort*.

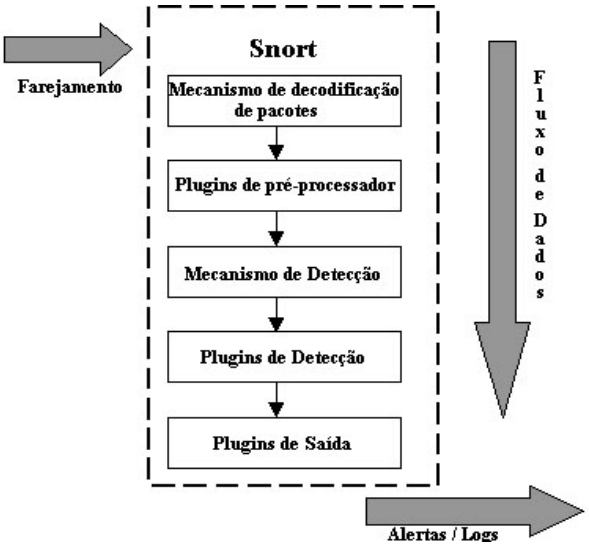


**Figura 3.2** – Componentes do *Snort*

A seguir apresenta-se os quatro componentes do *Snort*, bem como o seu papel dentro do processo de detecção de invasões:

1. **Mecanismo de captura / decodificador de pacotes** – Primeiro, o tráfego é obtido do *link* de rede, através da biblioteca *Libpcap*. Os pacotes passam pelo mecanismo de detecção, que montam a estrutura dos pacotes para os protocolos de enlace, os quais são ainda mais decodificados para os protocolos de nível mais alto, como as portas *TCP* e *UDP*.
2. ***Plugins* de pré-processadores** – Os pacotes passam então por um conjunto de pré-processadores. Eles são examinados e manipulados antes de serem enviados ao mecanismo de detecção. Cada pré-processador verifica se esse pacote é algo que ele deve examinar, alertar a respeito ou modificar.
3. **Mecanismo de detecção** – Em seguida, os pacotes são enviados para o mecanismo de detecção. O mecanismo de detecção verifica cada pacote em relação a várias opções listadas nos arquivos de regras do *Snort*. Cada uma das opções de palavra-chave da regra é vinculada a um *plugin* de detecção que pode realizar testes adicionais.

4. ***Plugins* de saída** – O *Snort* produz a saída dos alertas do mecanismo de detecção, dos pré-processadores ou do mecanismo de decodificação.

Apesar de poder decodificar outros tipos de protocolos, o *Snort* é focado na pilha de protocolos *TCP/IP*. Usando a pilha de protocolos *TCP/IP*, a figura 3.3 ilustra o nível de atuação dos componentes do *Snort*.

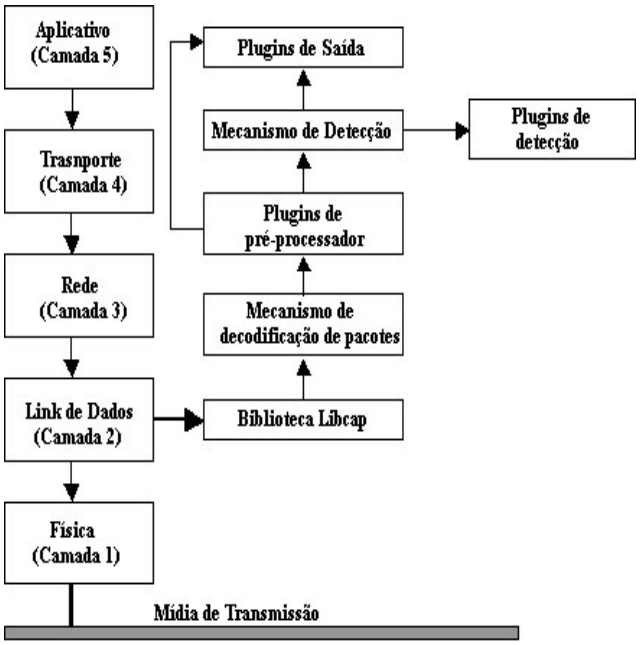


**Figura 3.3** – O *Snort* e o modelo *TCP/IP*

As camadas do TCP/IP são as seguinte:

- Camada de aplicação (Camada 5) – Por exemplo, protocolo *HTTP*, *SMTP;*
- Camada de transporte (Camada 4) – Por exemplo, protocolo *TCP* e *UDP;*
- Camada de rede (Camada 3) – Por exemplo, protocolo *IP* e *ICMP;*
- Camada de enlace (Camada 2) – Por exemplo, protocolo *Ethernet*, *Token Ring;*
- Camada física (Camada 1) – Por exemplo, placa de rede, cabos, conectores.

### 3.4.1 – Mecanismo de Captura

O *Snort* necessita de um mecanismo que capture o tráfego à medida que ele passe pela rede. O comportamento padrão de uma placa de rede é ignorar o tráfego que não é destinado para ela.

Precisa-se mudar este comportamento, fazendo com que a placa de rede verifique todo o tráfego do segmento de rede ao qual ela pertence. Para isso a placa deve operar em modo promíscuo.

O mecanismo que o *Snort* utiliza para colocar a placa de rede no modo de operação promíscuo é a biblioteca *LibPcap*. A biblioteca *LibPcap* foi escrita como parte de um programa maior chamado *TCPdump*.

Esta biblioteca permite que desenvolvedores escrevam códigos para receber pacotes da camada de *link* de dados em diferentes tipos de sistemas operacionais *UNIX like*, sem se preocupar com a idiossincrasia das placas de redes e *drivers*.

Basicamente, a biblioteca *LibPcap* pega pacotes diretamente da placa de rede, permitindo ao desenvolvedor escrever programas para decodificar, exibir ou registrar pacotes. Ela funciona de maneira semelhante a uma escuta telefônica, mas é usado para rede de dados, em vez de rede de voz. A biblioteca permite que um aplicativo ou um dispositivo de *hardware* se intrometa no tráfego da rede de dados[3].

Assim que os pacotes chegaram da placa de rede e foram passados para o mecanismo de decodificação do *Snort* pela biblioteca *Libpcap*, o *Snort* precisa decodificar os pacotes brutos da camada de *link* de dados. Dentro do código fonte do *Snort* existem funções implementadas para executar esta decodificação.

Tomando como exemplo uma rede *Ethernet* com tráfego *TCP/IP*, o pacote passa pelos seguintes passos de decodificação. Assim que o pacote chega é chamada a função que processa o pacote, esta função chama outra que decodifica estrutura *Ethernet*. Dentro da função que decodifica a estrutura

*Ethernet*, chama-se uma função que decodifica o protocolo *IP*, e finalmente esta função chama a que decodifica o protocolo *TCP*. Neste ponto o pacote *TCP* foi decodificado e é vinculado as estruturas de dados apropriadas. O *Snort* armazena os pacotes armazenados em ponteiros e estruturas de dados mantidos na memória.

### 3.4.2 Pré-processadores

Após a captura, os pacotes são encaminhados para os pré-processadores, ou seja, antes de serem enviados para o mecanismo de detecção, eles são passam primeiro pelos pré-processadores. A idéia principal por trás do pré-processador é fornecer uma estrutura para permitir alertar, eliminar e modificar os pacotes, antes que eles cheguem no mecanismo de detecção principal do *Snort*. A figura 3.4 ilustra o fluxo de pacotes após a captura.


**Figura 3.4** – Fluxo de Pacotes

Os recursos de detecção do *Snort* ocorrem através da correspondência de dados dos pacotes com padrões bem definidos. Esse padrões bem definidos ou regras, são uma evolução das assinaturas.

As assinaturas são basicamente especificações de ataques através de correspondência de número e *string* em relação a parte particular do pacote. Por exemplo, um pacote direcionado à porta 80, contendo *cmd.exe* geralmente é um bom sinal de um *hacker* atacando um servidor *Web* baseado no *Windows*. Um SDI pode detectar esse ataque, verificando o número da porta de destino, *flags TCP* e realizando uma correspondência de *string* simples em relação à parte de dados do segmento *TCP*. As regras são muito parecidas com isso.

Quando do desenvolvimento do *Snort*, existia uma demanda para ele ir além de um SDI de correspondência de regras. Uma das característica solicitadas era a detecção de anomalias de protocolo, onde o *Snort* poderia detectar que os dados de um pacote não obedecem a regras do protocolo a que pertence. Essa geralmente não é uma característica de um SDIR baseado em regra normal.

O *Snort* implementa recursos como a detecção de anomalias de protocolos através de pré-processadores. Os pré-processadores manipulam os dados dos pacotes após o decodificador do *Snort* ter analisado os campos do pacote, mas antes que o mecanismo de detecção comecem a fazer comparações com regras.

Há um custo na inclusão de pré-processadores. A velocidade do *Snort* é derivada de sua base de correspondência de regras simples. Haverá perda de velocidade sempre que um pré-processador for acrescentado. O grau de perda de performance esta diretamente ligado coma forma de implementação dos pré-processadores. No *Snort* essa implementação se dá por meio de *plugins* modulares, onde pode-se decidir exatamente quais pré-processadores serão ativados.

Como a implementação dos pré-processadores ocorre através de *plugins* pode-se deixar um(s) pré-processador fora da base de código. Cada *plugin* é implementado como trecho de código separado. Em seu próprio arquivo-fonte independente. Isso tem outras vantagens além da velocidade, pois garante independência de desenvolvimento, visto que, um colaborador pode desenvolver um *plugin* mais facilmente sem atrapalhar o desenvolvimento de outro.

O pré-processador recebe os pacotes capturados e os verificam em relação aos *plugins*. Os *plugins* verificam certos padrões dos pacotes. Detectado o padrão do pacote este é encaminhado para o mecanismo de detecção. O *Snort* possui *plugins* de *RPC*, *IIS*, *Telnet*, de fragmentação, entre outros.

Os *plugins* constituem uma importante característica do *Snort*. Eles podem ser desabilitados e ativados conforme a necessidade do administrador. Por exemplo, senão há tráfego *RPC*, torna-se desnecessário a ativação do *plugin* associado.

Os pré-processadores tornam mais fácil escrever regras, diminuem a presença de falso positivo / negativo e fornecem a um SDI de correspondência de regra a capacidade de superar seu modelo de detecção tradicionalmente simples, enquanto mantém o desempenho. Os principais objetivos para os quais os pré-processadores são usados são:

- Remontagem de pacotes;
- Decodificação de protocolos;
- Detecção não baseada em regras ou baseada em anomalias.

Basicamente, os pré-processadores são utilizados quando se quer detectar algo que a detecção baseada em regra direta não pode fazer sem ajuda.

#### 3.4.2.1 Remontagem de Pacotes

O *Snort* tem dois *plugins* de pré-processadores que ajudam na correspondência de regra, combinando dados espalhados em vários pacotes. Esses *plugins* são:

- *stream4*;
- *frag2*.

A detecção baseada em assinatura corresponde padrões bem definidos com os dados de cada pacote, um por vez. Neste caso ele pode ver os dados dos pacotes sem ajuda.

Remontando os fragmentos em pacotes completos com o *frag2*, pode-se garantir que um ataque não tenha êxito em usar fragmentação para se desviar da detecção.

Remontando cada fluxo em um ou mais pseudo-pacotes com o *stream4*, pode-se garantir que o mecanismo de assinatura de um pacote possa

corresponder padrões entre vários pacotes em uma sessão *TCP*.

Adicionando manutenção de estado com o *stream4*, pode-se fornecer alguma "inteligência" a essa correspondência de assinatura, sobre quais pacotes podem ser ignorados e onde um pacote está na conexão.

Os pré-processadores de remontagem de pacote ajudam a garantir que o *Snort* detecte ataques, mesmo quando os dados a serem correspondidos estejam em vários pacotes.

O *frag2* reconstrói um pacote a partir de todos os fragmentos que recebe e depois coloca o pacote reconstruído no caminho normal usado por um pacote que tenha acabado de deixar o decodificador.

O *stream4* não modifica nenhum dos pacotes que examina, em vez disso, ele constrói um "pacote úbero" de todos os dados e passa isso através dos outros pré-processadores e mecanismo de detecção [3].

### 3.4.2.2 Decodificação de Protocolos

A correspondência de padrão baseada em regras freqüentemente pode falhar em protocolos para os quais os dados podem ser representados de muitas maneiras diferentes. Por exemplo, os servidores *Web* aceitam muitas maneiras de escrever uma *URL*. O *IIS*, por exemplo, aceita caracteres de barra invertida "\" no lugar de barra normal "/" em *URL's*.

A detecção baseada  em regras geralmente fornece correspondência de *string* / *bytes* simples em relação a dados dentro de um pacote. Ela pode tratar de todas as versões diferentes de uma *URL* em dados *HTTP* sem ajuda ou pelo menos sem infinitos conjuntos de regras, mas com um grande número de regras para cobrir todas a s maneiras de representar uma *URL*.

O pré-processador *http_decode* fornece ao *Snort* a capacidade de tornar as *URL's* canônicas, antes de tentar corresponder padrões a eles. A correspondência de regra direta também pode ser frustrada por dados baseados em protocolos inseridos no meio de dados que, de outro modo, corresponderia a

um padrão [3].

Os pré-processadores *rpc_decode* e *telnet_negotiation* removem dados que poderiam ser estranhos ao dispositivo de correspondência padrão. O pré-processador *rpc_decode* consolida todos os fragmentos da mensagem em uma única mensagem *RPC*. O pré_processador *telnet_negotiation* remove seqüencias de negociação *Telnet* [3].

Os pré-processadores de decodificação de protocolo tornam a correspondência de *string* possível principalmente forçando os dados do pacote a serem algo menos ambíguo, para que eles possam ser mais facilmente correspondidos.

#### 3.4.2.3 Detecção não baseada em regras ou baseada em anomalias

A detecção baseada em regra funciona bem devido a sua simplicidade. Ela é muito determinística, tornando fácil configurar para a obtenção de menos falsos positivos, e é de fácil otimização. Entretanto, existem funções que simplesmente não podem ser obtidas sob esse modelo.

O *Snort* possui pré-processadores que realizam a detecção de ataque que não podem ser feita usando-se regras regulares ou detecção de anomalia de protocolos. Essa categoria de pré-processadores permitem que o *Snort* possa ser facilmente estendido para detectar ataques, praticamente de qualquer maneira que um desenvolvedor possa imaginar.

Alguns ataques simplesmente não podem ser detectados por correspondência de regra ou detecção de anomalia de protocolo. Por exemplo, pode ser difícil detectar de forma confiável varreduras de portas. Uma varredura de porta geralmente envolve várias investigações, normalmente em mais de uma porta ou em mais de uma máquina. Se não envolver, será muito difícil distinguir de uma tentativa normal de conexão. Um único pacote chegando a porta 80 de uma estação de trabalho, que não é um servidor *Web*, poderá ser somente um acesso à um endereço errado.

Por outro lado, 200 pacotes destinados à porta 80, enviados a cada um de seus endereços *IP*, chegando em ordem numérica pelo endereço *IP*, quase certamente é uma varredura de porta de uma pessoa curiosa. O que distingue um do outro é a combinação desses fatores:

- Número de *host* de destino;
- Número de portas de destino;
- Tempo durante o qual os pacotes foram enviados.

Não existe uma maneira de levar todos esses fatores em conta com a correspondência de regras direta, mesmo com a remontagem de fluxo. O *Snort* possui pré-processadores que detectam esses tipos de anomalias (tráfego suspeito), que para o exemplo citado é o *portscan*.

O *Snort* conta com outros pré-processadores para outros tipos de anomalias que envolvem vários fatores, entre eles pode-se citar o *Back Orifice detector* (Decodifica tráfego de rede do *Back Orifice*), *arpspoof (plugin* de detecção de uso incorreto de *ARP*), *fnord* (Analisador e detector de *shellcode* polimórfico) entre outros.

### 3.4.3 Mecanismo de Detecção

Após os pacotes serem capturados pela rede, eles são decodificados e colocados nas estruturada de dados, organizando, filtrando e decodificando o fluxo de dados.

O mecanismo de detecção pode ser considerado o coração do *Snort*, ou seja, é parte mais importante do sistema. Ele recebe os dados provenientes do pré-processador e verifica através de um conjunto de regras (assinaturas).

As regras do *Snort* são baseadas em texto e normalmente armazenadas em um diretório ou subdiretório da instalação do *Snort*. Os arquivos de regras são classificados em diferentes grupos; por exemplo, o arquivo *ftp.rules* contém uma coleção de ataques e explorações *FTP*.

Na inicialização o *Snort* lê todos os arquivos de regras e cria uma lista

encadeada tridimensional. O *Snort* usa essa lista para comparar pacotes para detecção[3] **.**

O *Snort* armazena as regras e suas opções na lista encadeada e depois procura na lista uma correspondência de cabeçalho da regra. Então, dentro da correspondência de cabeçalho, ele procura uma correspondência de padrão ou uma combinação, usando um *plugin* de detecção. A figura 3.5 ilustra a estrutura de armazenamento das regras.

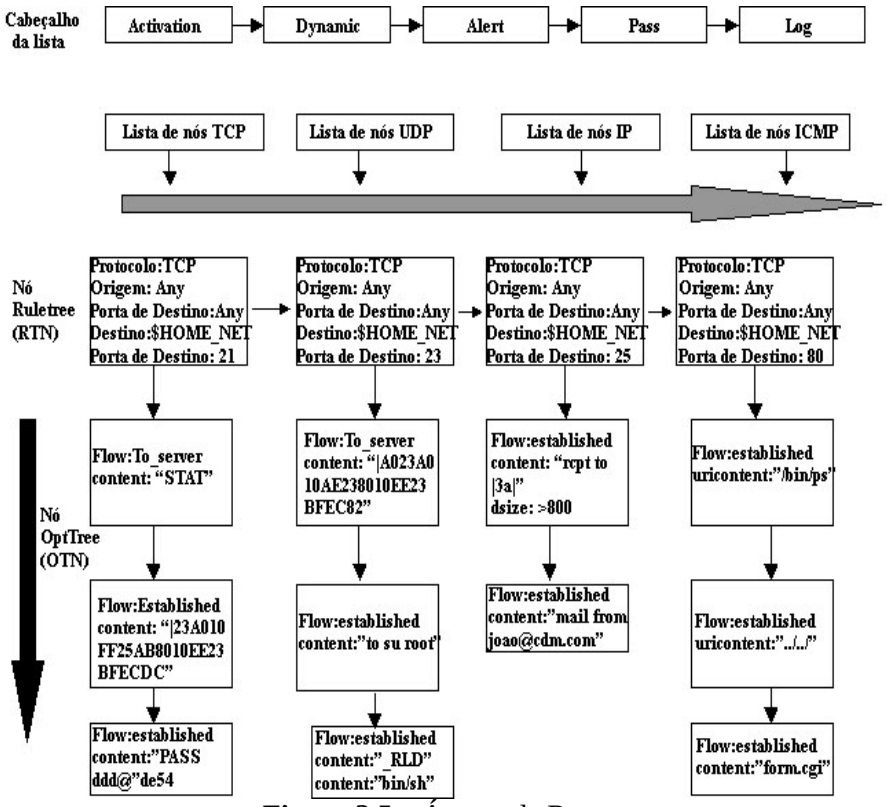


**Figura 3.5** – Árvore de Regras

Existem cinco encadeamentos de regras separados. Esses encadeamentos são os cabeçalhos da lista, que podem ter os seguintes valores:

- ***Activation*****:** Alertar e depois ativar outra regra dinâmica;

- ***Dynamic*:** Registra o tráfego quando chamado pela regra de ativação anterior;
- ***Alert*:** Gera um alerta e depois registra o pacote;
- ***Pass*:** Ignora esse pacote;
- ***Log*:** Registra o tráfego (Não Alerta).

Para cada um dos cinco encadeamento de regras, existem listas encadeadas separadas, divididas por protocolo. Esse nível da árvore é referido como *RTN* (*Rule Tree Nodes* – nó da árvore de regras). Os quatro protocolos suportados são[15]:

- ***TCP***: Protocolo *TCP*; por exemplo, *SNMP*, *HTTP*, *FTP*;
- ***UDP*** Protocolo *UDP*; por exemplo, pesquisa *DNS*;
- ***ICMP***: Protocolo *ICMP*; por exemplo, *ping*, *traceroute*;
- ***IP***: Protocolo *IP*; por exemplo, *IPSec*, *IGMP*.

Dentro de cada uma das listas encadeadas de protocolo estão as opções da regra, que são referidas como *OTN* (*Options Tree Nodes* – nós da árvore de opções). Um exemplo desses opções seria:

- ***Content***: Conteúdo verificado pelo algoritmo de correspondência de padrões;
- ***Flow***: *Link* para *plugin* de detecção.

Na inicialização o *Snort* lê os arquivos de regras e preenche as listas encadeadas. Uma vez as listas construídas, é necessário um método de navegação para procurar uma correspondência com os pacotes.

Quando o pacote chega no mecanismo de detecção, o *Snort* percorre os cabeçalhos de regra na seguinte ordem: *Activation*, *Dynamic*, *Alert*, *Pass* e *Log*. Dentro de cada cabeçalho de regras, os *RTN* e *OTN* serão verificados. Por exemplo, supondo a existência de um pacote *HTTP* que coincidirá com a regra *Alert*, os nós *Activation* e *Dynamic* serão verificados e nenhum correspondência será retornada.

Não encontrando nenhuma correspondência, o *Snort* passa para o nó

subseqüente, no caso *Alert*. O método de pesquisa é dependente do protocolo do pacote, e a busca inicia-se na lista de nós *TCP*. Percorre-se os *RTN's* da esquerda para a direita, tentando combinar o pacote através dos seguintes parâmetros:

- Endereço *IP* de origem;
- Endereço *IP* de destino;
- Porta de origem;
- Porta de destino.

Quando o *Snort* encontra uma correspondência, o algoritmo percorre as colunas debaixo, procurando uma correspondência dentro de cada um dos *OTN's*. Por exemplo, podem existir várias regras *HTTP*, portanto, o *Snort* verifica as opções de cada uma. Quando encontra a correspondência da exploração o *Snort* faz o seguinte:

Dentro do nó de opção existem dois ítens:

- Um ponteiro para um *plugin* de detecção, que é a terceira dimensão da lista encadeada, sendo as duas primeiras o *RTN* e os *OTN's*. O *plugin flow* verifica se o pacote corresponde a uma sessão estabelecida [3];
- O padrão que se procura. Usa-se o algoritmo de pesquisa rápida de *string Boyer-Moore.*

Encontrada a correspondência, o processo de busca sai da estrutura em árvore e retorna ao cabeçalho *Alert*. Neste momento será gerado o alerta de acordo com o formato especificado e o processo se encerra.

O *Snort* utiliza uma estratégia de saída rápida. Quando há uma correspondência de um pacote com alguma regra, o *Snort* não verifica mais esse pacote com nenhuma outra regra.

Como já relatado o mecanismo que o *Snort* utiliza para processar as regras, as buscam na seguinte ordem: *Activation*, *Dynamic*, *Alert*, *Pass* e *Log*. Pode ser interessante alterar esta ordem de busca, tornando a regra *Pass* a

primeira a ser procurada, para isso deve inicializar o *Snort* com o parâmetro -o. Deve-se ter cuidado em usar este parâmetro, pois uma regra errada pode deixar todo o tráfego passar e não alertar nada.

### 3.4.4 *Plugins* de Saída

Após a verificação do pacote pelo mecanismo de detecção e ocorrendo a correspondência do pacote com alguma regra algum procedimento deve ser adotado.

Os *plugins* de saída são os responsáveis por realizar esse procedimento, que pode ser gerar alertas ou tomar alguma medida em imediato. A figura 3.6 ilustra os procedimentos que podem ser adotados pelos *plugins* de saída.

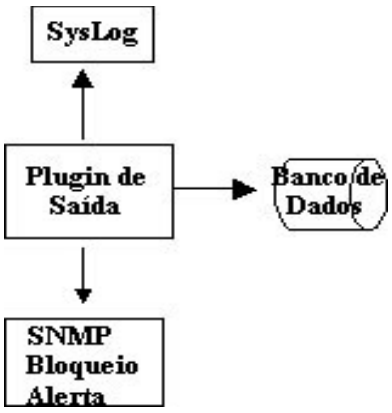


**Figura 3.6** – *Plugins* de Saída

Os *plugins* de saída podem interagir com *firewall*, podem enviar alertas via *e-mail*, *popups*, gravar em arquivos textos, *MySQL* entre outros[17]. Existem muitas ferramentas adicionais que podem ser usadas com o *Snort*, como *plugins* para *Perl*, *PHP*, servidores *Web*, *ACID*, *Swatch* etc. Após os *plugins* de saída o pacote passou por todas as etapas do *Snort* [17].

Entre os *plugins* de saída pode-se destacar os *plugins Snortsam* e o *Guardian*. Esses dois *plugins* tem a capacidade de gerar regras de *firewall*, bloqueando tentativas de invasão em tempo real.

O *Snortsam* fornece uma ligação entre a detecção de uma exploração e a configuração de um *firewall* para bloquear o endereço *IP* de origem do ataque. A configuração do *firewall* é realizada através da adição de alguns parâmetros nas assinaturas de ataques. Maiores informações sobre o *Snortsam* pode ser obtidas em [14].

O *Guardian* é uma ferramenta que tem a capacidade de ler os *logs* do *Snort* em tempo real e gerar regras de *firewall* bloqueando os *IP's* de origens dos ataques. Maiores informações sobre o *Guardian* consulte a seção 4.2.3**.**

## 3.5 Criação de Regras

Como já relatado neste documento as regras do *Snort* constituem o seu coração. Saber criá-las pode ser uma excelente arma para o administrador de redes. O *Snort* possui uma linguagem fácil na construção de novas regras que podem ser codificadas em uma única linha.

Como pode ser observado usa-se o termo regra e não assinatura. O termo assinatura refere-se a nada mais do que uma definição básica de ataque, parecido com uma pegada deixada na lama pelo sapato de alguém invadindo uma casa. Uma regra define a metodologia de ataque em termos de identificação do invasor, análogo à identificação do modo como o ladrão entrou na casa, na esperança de capturá-lo.

As regras do *Snort* são divididas em duas partes lógicas: cabeçalho e miolo. O cabeçalho contém a ação da regra, protocolo, endereço *IP*, máscara de rede e porta. O miolo contém mensagem de alerta e informações sobre que parte do pacote deverá ser examinada.

### 3.5.1 Cabeçalho

Os cabeçalhos das regras do *Snort* são considerados a parte principal da regra. Eles identificam o que deve ser feito quando a regra for sinalizada, qual protocolo deve ser usado e as informações de origem e destino, incluindo

postas, endereço *IP* e redes. O cabeçalho, ilustrado pela figura 3.7, é dividido em quatro categorias principais:

- Ação da Regra;
- Protocolo;
- Informações da Origem;
- Informações do Destino.


**Figura 3.7** – Regra do *Snort*

As ações que podem ser tomadas pelo *Snort* são as seguintes:

- ***Activate****:* É a mais poderosa do *Snort*. Ela gera um alerta e depois inicia a regra dinâmica especificada. Ela é muito útil para capturar ataques complexos.
- ***Dynamic:*** Essa ação permanece inativa até que uma regra *active* a dispare;
- ***Alert***: Registra o pacote e depois alerta o usuário da forma especificada no arquivo de configuração;
- ***Pass***: Ignora esse pacote, continuando a análise nos pacotes subseqüentes;
- ***Log***: Registra o pacote (Não Alerta).

Essas são as cinco ações pré-existentes no *Snort*, pode-se criar tipos de regras personalizados. Esses tipos de regra determinam como outros aplicativos produzem a saída dos dados em outros tipos de *plugins* de saída.

A regra a seguir propicia a criação de um arquivo texto de *log* quando uma anomalia de *hacker* definida é detectada.

```
ruletype hacker_log
{
        type log
        log_tcpdump: hacker.txt
}
```

Outro exemplo de regra, segue abaixo. Esta regra é para atualizar um banco de dados remoto.

```
ruletype update_database
{
        type alert output
        output database: log, mysql, user=bruno password=RTd23_2005 dbname=teste
host=192.168.200.8
}
```

Maiores informações sobre criação de tipos podem ser obtidas em [3].

O campo protocolo contém o protocolo a ser analisado. Atualmente a árvore de regra suporta os protocolos *TCP*, *UDP*, *ICMP* e *IP*. Ao designar o protocolo da regra, deve-se simplesmente incluir o tipo, após a ação da regra, separado por um espaço em branco.

A terceira parte do campo cabeçalho da regra é usado para atribuir endereços *IP* de origem e destino pertinentes a essa regra específica. O coringa "*any*" pode ser utilizado para definir qualquer endereço. Os endereços são representados da seguinte forma: *IP/CIDR* (*Classless InterDomain Routing*), por exemplo, 192.168.200.0/24. O operador de negação "!" pode ser usado em conjunto com um endereço *IP*, para negar o valor. Por exemplo, para criar uma regra que fosse válida para qualquer rede, exceto a rede 192.168.200.0, usaria-se a seguinte diretiva: ! 192.168.200.0/24

Pode-se especificar intervalos de endereço em um formato de lista,

usando colchetes para englobar intervalos de *IP's*, com cada elemento separado por vírgula, como observado a seguir:

[192.168.100.0/24, 192.168.200.0/24, 192,168,20.5]

O número da porta também pode ser especificado de várias formas incluindo o coringa *any*, uma única porta ou uma faixa de portas. O *Snort* não permite a listagem de portas específicas separadas por vírgula, entre colchetes, como no esquema *IP*. Entretanto, o *Snort* possibilita a inclusão de intervalos de portas com um número de porta mínimo e máximo predefinidos. Caso haja a omissão dos valores mínimos e máximos o *Snort* considerará os valores 0 e 65535, como os valores mínimos e máximos respectivamente. Os valores mínimos e máximos são separados pelo caracter dois pontos (:).O exemplo abaixo especifica as portas de 1 a 1024 da rede 192.168.200.

192.168.200.0/24 1:1024

Para especificar todas as portas acima de 1024 pode-se usar a seguinte notação:

192.168.200.0/24 1024:

Os operadores de direção dizem ao mecanismo do *Snort* a maneira correta de interpretar as regras. Atualmente, existem apenas dois operadores de direção: -> e <>. O operador -> diz ao *Snort* que ao lado esquerdo do operador está a origem e a direita o destino. A regra a seguir registra todos os pacotes *TCP* de qualquer origem com destino a rede 172.20. na porta 21 (*FTP*).

log tcp any any -> 172.20.0.0/16 21

O operador <> não impõe nem se preocupa com o lugar que o tráfego se inicia. O exemplo a seguir ilustra uma regra que registra todos os pacotes *TCP* transferidos em portas abaixo de 1024, entre as redes 192.168.200.0 e 192.168.100.0.

log tcp 192.168.200.0/24 :1024 <> 192.168.100.0/24 :1024

O *Snort* permite ao usuário definir regras *activate* e *dynamic*, criando

assim um encadeamento de regras. Quando uma regra *activate* é disparada, ela ativa a regra *dynamic* definida para começar a executar com base em alguns detalhes de configuração específicos dentro desta regra *dynamic*.

A seguir está uma regra um exemplo das regras *activate* e dynamic que são disparadas quando os dados estão sendo enviados para a porta 23, mais comumente usadas para *Telnet*.

Activate tcp any any -> any 23 (activates: 23; msg “Potencial Telnet Login “;) dynamic tcp any any -> any 23 (activated_by: 23; count 20;)

### 3.5.2 Miolo da Regra

O *Snort* não exige que o campo miolo tenha definições de regras completas, mas a sua inclusão é excelente para ampliar as definições das regras. A não obrigatoriedade do miolo não diminui sua importância, este pode ser considerado a alma do mecanismo de detecção.

O formato do miolo é dividido em seções separadas por ponto e vígula (;). Cada seção define uma opção, seguida do valor da opção desejada. O miolo da regra fica entre parênteses. Não é o objetivo deste autor enumerar todas as opções de assinaturas. Serão especificadas algumas. Caso o leitor queira conhecer outras opções basta acessar www.snort.org/docs ou consultar [3]:

- *nocase:* Não faz distinção entre maiúscula e minúscula;
- *content*: procura por um molde no conteúdo do pacote. A busca pode ser realizada por conteúdo *ASCII* ou binários, podendo negar uma expressão com o ponto de exclamação (!). A expressão a ser localizada deve vir entre aspas dupla. A seguir segue um exemplo de regra, que gera um alerta sempre que uma conexão *Telnet* for realizada com a palavra *root*, não fazendo distinção entre maiúscula e minúscula;

  alert tcp any any -> any 23 (content: “root”; nocase;)
- *msg:* imprime uma mensagem em alerta e registra o pacote;

- *regex*: habilita a utilização de expressões regulares dentro da opção *content*. Atualmente só estão disponíveis os caracteres coringa interrogação (?) e asterisco (*). A seguir segue um exemplo de uma regra que alerta para toda tentativa de *dot dot Attack*. Maiores informações sobre esta técnica de ataque pode ser obtida em [13] :

  alert tcp any any -> 200.20.167.20 80 (content: "../*../"; regex; msg: "Dot dot Attack";)
- *uricontent*: opção de conteúdo *URI* [5](*Uniform Resource Identifier).* Permite verificar a correspondência de conteúdo da regra na seção *URI* de um pedido. Um exemplo desta regra pode ser observado a seguir, onde é logada qualquer requisição *web* na porta 80 onde haja a expressão "*cgi-bin/phf*" na seção *URI*;

  log tcp any any -> any 80 (content: "Logging PHF"; uricontent: "/cgi-bin/phf";)
- *flow*: permite a definição da direção do pacote em relação aos fluxos de comunicação cliente-servidor. Defina *flow*: <opção>, onde opção é uma das instruções de opção da tabela 3.1. A seguir segue um exemplo de uma regra que alerta sobre um potencial envio do arquivo */etc/passwd* de um servidor.

  alert tcp 192.168.100.2 any -> any any (msg: "Potencial envio do arquivo / etc/passwd"; flow:from_server; content:"root::";)
- *sameip:* determina se o endereço *IP* de origem é igual ao endereço de destino. A seguir segue um exemplo dessa opção de regra;

  alert ip any any ->any any (msg : "Mesmo endereço IP de origem e destino"; *sameip*;)
- *session*: permite capturar dados em texto puro das sessões do protocolo e produz uma saída desses dados. Um exemplo desta opção pode ser observado abaixo. Essa regra gera um alerta e imprime a transmissão da sessão *FTP* inteira na saída padrão. Como a transmissão de *FTP* é em texto plano

5 URI especifica o computador, a localização dos pacotes e o protocolo usado para acessar esses pacotes

aparecerão usuário e senha.

alert tcp any any -> any 21 (content: “FTP Session Data”; session: printable;)

**Tabela 3.1- Opções do parâmetro *flow***

| Opção | Descrição |
|---|---|
| *to_server* | Pacotes enviados para o servidor |
| *from_server* | Pacote enviados do servidor |
| *to_client* | Pacotes enviados para o cliente |
| *from_client* | Pacotes enviados do cliente |
| *only_stream* | Pacotes reconstruídos ou dentro de um fluxo estabelecidos |
| *no_stream* | O oposto da opção anterior |
| *established* | Pacotes que fazem parte de uma conexão ou sessão *TCP* estabelecida |
| *stateless* | Independe do estado do pacote |

O *Snort* fornece suporte ao Filtro de Pacote *Berkeley* (*Berkeley Packe*t *Filter- BPF*). As regras *BPF* foram desenvolvidas para fornecer uma linguagem de alto nível para filtragem de pacotes. Ela permite ao usuário definir regras rápidas de análise de pacote, com base na origem, destino, protocolo e em outras informações de cabeçalho pertinentes. O esquema *BPF* é a base do aplicativo *TCPdump*. Maiores informações sobre este aplicativo pode ser obtidas em www.tcpdump.org.

Existem diversas formas de implementar uma regra do *Snort*. Podem existir numerosos métodos para identificar ataques danosos, mas independente da forma de implementação, existem algumas precaução a serem adotadas na definição de uma boa regra.

Para minimizar falsos positivos e negativos, é fundamental examinar o miolo das regras, especialmente, é importante examinar as assinaturas de ataque de conteúdo dentro das regras.

Além da definição do conteúdo das regras, é importante definir os eventos de ação adequados para suas regras. Basicamente existem duas escolhas

principais: registrar e alertar. Deve-se primeiro definir em qual categoria a regra se encaixa. Os questionamentos a seguir, ajuda a definir a categoria:

- O ataque afeta sistemas de missão crítica?
- O ataque dá acesso não autorizado a dados de missão crítica?
- O ataque compromete diretamente um sistema?

Se a resposta para qualquer uma dessas perguntas for positiva, a ação da regra deve ser *alert*. Se negativa, talvez seja necessário somente registrar os dados para posterior análise. As duas instâncias gerais para se usar registro de dados são:

- Os *logs* fornecem dados evidencias que podem ser usados para identificar ou processar um invasor;
- Os *logs* fornecem um meio adicional para informações de ataque de alto risco.

É extremamente importante testar todas as regras incluídas nos arquivos de definição de regras. Em geral, as regras devem ser testadas com relação a quatro métricas: eficiência, utilidade, precisão e singularidade.

Regras e conjuntos de regras incorretamente definidos podem estragar o desempenho de seu sistema e inundar seus arquivos de alerta e *log*. Em geral existem os seguintes métodos para testar as regras:

- Teste de esforço;
- Teste de regra individual do Snort;
- Teste de *Berkeley Packet Filter* (*BPF*).

Os teste de esforço de vulnerabilidade, ataque e pacote são alguns dos mais úteis a serem realizados em sensores *Snort*. O objetivo do teste de esforço é identificar limites, descobrindo o volume de dados que pode ser processado e analisado pelo sistema *Snort* implementado.

Existem diversas ferramentas para realização de teste de esforço, entre elas destaca-se, o *Nessus*, *Satan* e *Nmap* entre outros. Para informações sobre

essas ferramentas consulte respectivamente [21], [9] e www.insecure.org.

Para realização de teste de sintaxe individual de regra, utiliza-se o seguinte comando que analisa um arquivo de regras. A *flag* -i especifica a interface e a -n determina a interrupção após ter recebido um pacote.

```
snort -i eth0 -n 1 -c <arquivo de regra>
```

De modo análogo ao teste de regra de sintaxe individual, existe a possibilidade de testar individualmente as regras *BPF* com o utilitário *TCPdump*. Como o utilitário *TCPdump* é apenas um interpretador de regras, muita pouca funcionalidade de depuração está incorporada nele. O modo mais fácil de identificar erros em potencial é testar a regra quanto a sintaxe correta. O comando a seguir testa a sintaxe de um arquivo de regra *BPF*.

```
tcpdump -i eth0 -n -F <arquivo de regras BPF>
```

# 4 Instalação e Configuração de um Servidor SDI

## 4.1 Descrição do Servidor SDI

Para instalação de um servidor SDI utiliza-se o *Snort*, que é um dos SDIs mais conhecidos na atualidade no mundo do *software* livre. Com o intuito de acrescentar maiores funcionalidades ao sistema de detecção de invasão alguns outros aplicativos foram acoplados ao *Snort*, criando a proposta de SDI. Entre eles destacam-se o *Analysis Console Intrusion Database* (*ACID*) e o *Guardian*.

O *ACID* é uma ferramenta para navegação e análise de dados. Basicamente o *ACID* é um conjunto de *scripts PHP* que fornece uma interface entre um navegador *Web* e o banco de dados onde os dados do *Snort* estão armazenados.

O intuito do *ACID* é tornar a leitura dos *logs* do *Snort* muito mais agradável e interessante. Sua função é eliminar a dificuldade de interpretação dos *logs* do sistema, o que normalmente acarreta a não observação de informações importantes para a segurança do sistema.

Maiores informações sobre o ACID estão disponíveis em http://acidlab.sourceforge.net/.

O *Guardian* que constitui o outro aplicativo instalado no servidor, é um *plugin* para o *Snort. Ele* tem a capacidade de ler em tempo real os *logs* do *Snort e* bloquear as tentativa de ataque que estão sendo realizada em alguns dos sistemas monitorados.

Para que o *Guardian* consiga desempenhar suas funções torna-se necessário um *firewall*, que se encarregará de bloquear o invasor e mantê-lo sem conexão com o servidor alvo do ataque.

Atualmente o *Guardian* contém *scripts* compatíveis com o *Ipfw*[6], *Iptables* e *Ipchain*, mas nada impede o desenvolvimento de *scripts* compatíveis com outros *firewalls*. O *download* do *Guardian* pode ser realizado em http://www.snort.org/dl/contrib/other_tools/guardian/.

## 4.2 Implantação do Servidor SDI

A adição dos aplicativos ACID e *Guardian* ocasiona à necessidade da instalação de outros aplicativos na solução proposta. Por exemplo, o *ACID* necessita de um banco de dados para armazenar os alertas do *Snort*, permitindo assim sua apresentação de forma mais amigável ao usuário. Como relatado na seção anterior o *Guardian* também necessita de um *firewall*.

As necessidades de aplicativos adicionais não se resume somente ao servidor de banco de dados e ao *firewall*. Nas seções a seguir será abordado os requisitos de *softwares*, bem como as configurações adicionais necessárias para confecção do servidor SDI.

### 4.2.1 Requisitos de Sistema

O sistema operacional utilizado para instalação do servidor proposto foi o *Linux*, utilizando a distribuição *Fedora Core* 2. A distribuição utilizada não é tão importante, pois todos os componentes utilizados podem ser instalados em qualquer sistema *Unix-Like*.

O *Snort* para funcionar exige a biblioteca *Libcap*. A *Libpcap* é uma biblioteca de captura de pacotes para sistemas *Linux*. Sua função é colocar a interface de rede em modo promíscuo, ou seja, capturando pacotes destinada a ela, bem como os pacotes destinados a outros *hosts* da rede.

Recomenda-se a instalação da versão mais recente desta biblioteca, mesmo que seu sistema já possua alguma versão instalada. A atualização pode

6 Firewall padrão do FreeBSD

oferecer uma maior estabilidade e velocidade na execução do programa. A versão mais atual pode ser encontrada em www.tcpdump.org.

Outra recomendação que se faz é quanto a capacidade de armazenamento. O tamanho do disco rígido varia de acordo com o tamanho da rede e dos dados inspecionados, mas a princípio recomenda-se uma capacidade razoável de armazenamento, visto que os alertas do *Snort* obrigatoriamente devem ser armazenados.

Na instalação deste servidor SDI, o armazenamento de dados em banco de dados deixa de ser opcional e passa a ser obrigatório, visto que, opta-se por trabalhar com o *ACID* para monitoramento dos *logs*. Atualmente o *ACID* suporta o *PostgreeSQL* e o *MySQL*, mas é possível modificar os *scripts* existentes para que ele suporte outros bancos. Neste documento, optou-se por utilizar o *MySQL*.

Antes da instalação do *Snort*, torna-se necessário a instalação do *MySQL*, já que é necessário habilitar o suporte ao *MySQL* durante o processo de instalação. Este SGBD[7] bem como sua documentação podem ser encontrados em http://www.mysql.org.

Após a instalação do *MySQL* torna-se necessário inicializar o banco de dados "*mysql*" com suas tabelas de permissão. Para realização desta operação basta executar um *script* chamado *mysql_install_db*, existente no subdiretório *scripts* da instalação do *MySQL*.

Depois de instalar e inicializar o banco de dados *MySQL* é fortemente recomendável a configuração de uma senha de acesso ao banco para o usuário *root*. A configuração da senha é realizada com o comando `"SET PASSWORD FOR root@localhost=PASSWORD ('senha');"` após a conexão com o banco de dados realizada com o comando `mysql`. Agora o acesso ao banco de dados tem que ser realizado com o parâmetro -p (*mysql -p*), caso contrário

7 Sistema de Gerenciador de Banco de Dados

uma mensagem de erro será apresentada.

Neste momento já é possível realizar a instalação do *Snort*, não se esquecendo de habilitar o suporte ao *MySQL*. Para maiores informações sobre instalação do *Snort* consulte [3]. É recomendável estar atento às atualizações do mesmo, que estão sempre disponíveis em www.snort.org.

Após a instalação do *Snort* torna-se necessário criar o banco de dados onde serão armazenados os alertas do *Snort*. Existe um *script* chamado *create_mysql* incluído na distribuição do *Snort* e localizado no subdiretório *contrib* que quando executado, cria todas as tabelas necessárias. Este *script* é executado como segue:

```
# mysqladmin -u root -p create snort_db
#mysql -u root -p
mysql> connect snort_db
mysql> source create_mysql
```

Para finalizar a criação do banco de dados os comandos a seguir devem ser executados:

```
mysql> grant INSERT, SELECT on snort_db.* to snort@localhost;
mysql> grant INSERT, SELECT on snort_db.* to snort@"%";
mysql> connect  mysql
mysql> set password for 'snort'@'localhost' = password('senha');
mysql> set password for 'snort'@'%' = password ('senha');
mysql>flush privileges;
mysql> exit;
```

Se tudo ocorreu conforme o esperado, pode-se inicializar o *Snort*, mas não antes da realização de algumas modificações no seu arquivo de configuração (*snort.conf*). A configuração do *Snort* deve estar de acordo com a política de segurança da empresa.

Não é objetivo deste documento descrever cada ítem do arquivo de configuração. O próprio arquivo já está escrito de maneira bem intuitiva. A

leitura do arquivo com atenção é imprescindível para a utilização do mesmo. Entre as opções de configuração destaca-se as seguintes:

- Os endereços de rede interna e externa;
- O endereço na árvore de diretório onde estão localizadas as regras;
- Os recursos de remontagem de pacotes para detecção de ataques de fragmentação, *arpspoof*, *Telnet*, requisição *HTTP* e tráfego *RPC*, bem como a utilização correta de seus respectivos parâmetros. Estes parâmetros mal configurados podem causar um aumento desnecessário no uso de *CPU* e memória;
- O banco de dados a ser usado para armazenar os eventos considerados nocivos ao ambiente;
- Otimização das regras a serem usadas pelo Sistema de Detecção de Intruso, isto é, devemos eliminar aquelas que não se aplicam ao ambiente monitorado.

Os ítens anteriores são constituídos dos seguintes parâmetros do arquivo de configuração do *Snort* (*/etc/snort/snort.conf*):

**var HOME_NET any** – este parâmetro deve ser alterado para o *IP* da placa de rede ou da rede interna. Se há mais de uma placa de rede pode-se usar a seguinte notação: [10.1.1.0/16,192.168.200.0/24]

**var EXTERNAL_NET any** – este parâmetro define a rede externa. Um bom valor para ele é *any*.

**var RULE_PATH ../rules** – este parâmetro deve conter o caminho da árvore de diretório que contém as regras do *Snort*.

**preprocessor opções** – este parâmetro define os preprocessadores ativos no *Snort*, devem estar habilitados somente aqueles relevantes à rede.

**output opção** – este parâmetro define onde serão armazenados os *logs* do *Snort*. Nesta implementação de servidor SDI este parâmetro teve a seguinte configuração: `output database: log, mysql, user=snort password=senha dbname=snort_db host=localhost`

**include $RULE_PATH/regra** – é a parte final do arquivo de configuração do *Snort*. Este parâmetro define as regras habilitadas no *Snort*. Devem estar habilitadas somente as relevantes à política de segurança da empresa.

As figuras 4.1 e 4.2 ilustram trechos do arquivo de configuração do *Snort*, respectivamente os trechos referentes a pré-processadores e regras.

```
##################################################
# Step #2: Configure preprocessors
#
# General configuration for preprocessors is of
# the form
# preprocessor <name_of_processor>: <configuration_options>

# Configure Flow tracking module
# ------------------------------
#
# The Flow tracking module is meant to start unifying the
state #keeping mechanisms of snort into a single place. Right
now, only  #a portscan detector
# is implemented but in the long term,  many of the stateful
#subsystems of snort will be migrated over to becoming flow
#plugins. This must be enabled for flow-portscan to work
#correctly.
#
# See README.flow for additional information
#
preprocessor flow: stats_interval 0 hash 2
```

**Figura 4.1**- Trecho de configuração do *Snort* referente à preprocessadores

```
#==========================================
# Include all relevant rulesets here
#
# The following rulesets are disabled by default:
#
#   web-attacks, backdoor, shellcode, policy, porn, info, icmp-
#info, virus, chat, multimedia, and p2p
#
# These rules are either site policy specific or require tuning
#in order to not generate false positive alerts in most
enviornments.
#
# Please read the specific include file for more information
and
# README.alert_order for how rule ordering affects how alerts
#are triggered.
#==========================================
include $RULE_PATH/local.rules
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/scan.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
#include $RULE_PATH/telnet.rules
include $RULE_PATH/rpc.rules
include $RULE_PATH/rservices.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
#include $RULE_PATH/tftp.rules
include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-coldfusion.rules
#include $RULE_PATH/web-iis.rules
#include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-php.rules
include $RULE_PATH/sql.rules
include $RULE_PATH/x11.rules
include $RULE_PATH/icmp.rules
include $RULE_PATH/netbios.rules
include $RULE_PATH/misc.rules
include $RULE_PATH/attack-responses.rules
#include $RULE_PATH/oracle.rules
```

**Figura 4.2 -** Trecho de configuração do *Snort* referente à regras

Neste momento para testar o funcionamento do *Snort* pode-se inicializá-lo através do seguinte comando:

```
snort -c /etc/snort/snort.conf -D
```

Se tudo ocorreu bem deve existir a seguinte entrada no arquivo /var/log/messages:

```
Snort initialization completed successfully
```

Caso não exista a mensagem acima, algo de errado ocorreu. Para verificação do erro, o *Snort* deve ser inicializado em modo verbose com o comando abaixo:

```
snort -vde -c /etc/snort/snort.conf
```

### 4.2.2 Ferramenta de Análise de *Logs*

Após os passos da seção anterior terem sido executados, e se tudo deu certo, provavelmente o *Snort* estará sendo executado e armazenando seus *logs* no banco de dados *MySQL*. Como pode ser observado, ainda não existe nenhuma ferramenta implantada que facilite a interpretação dos *logs*.

Neste ambiente proposto a ferramenta escolhida foi o *ACID*. O *ACID* pode ser acessado por qualquer navegador disponível no mercado. Este aplicativo atualmente fornece os seguintes recursos:

- Um interface amigável para pesquisa e consulta em banco de dados. Essas consultas podem ser realizadas por diversos parâmetros entre eles pode-se destacar o endereço *IP*, protocolo, hora, regra etc.
- Um decodificador de pacotes que pode exibir informações de camada 3 (rede) e 4 (transporte) dos pacotes registrados;
- Recursos para gerenciamento de dados, onde pode ser realizado o agrupamento de alertas, exclusão, arquivamento ou exportação de alertas;
- Geração de informações estatísticas, como por exemplo gráficos.

A estrutura do *ACID* tem várias camadas físicas e é escalável. Pode-se usá-la em apenas um computador ou pode-se ter uma arquitetura de até três

camadas. Apesar de ter sido adotado a opção de uma só camada, *ACID*, *Snort* e servidor *Web* em uma única máquina, a figura 4.3 ilustra o modelo em três camadas, pois facilita a visualização desses níveis.

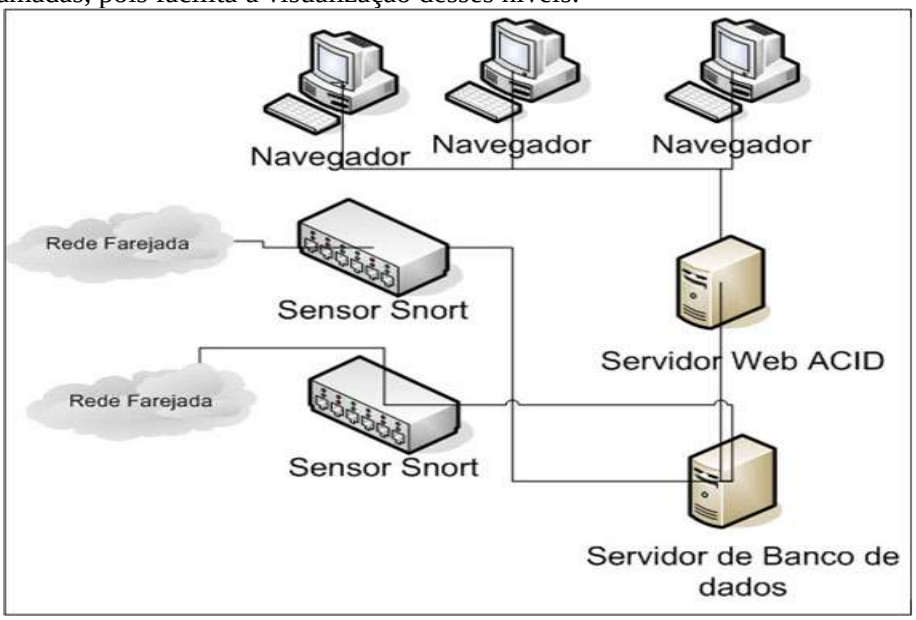


**Figura 4.3** – Arquitetura de várias camadas de um SDI

Como pode ser observado, o *ACID* trabalha com alertas armazenados em um banco de dados pelos sensores *Snort*. Um conjunto de *scripts PHP* é usado para criar consultas e navegar pelos resultados. Pode ser usado qualquer servidor w*eb*, desde que haja suporte a *PHP* 4.

Optou-se pelo servidor *web Apache* na elaboração deste ambiente. Para realização do *download* deste aplicativo, bem como consulta à sua documentação acesse http://www.apache.org. A instalação pode ser realizada via compilação dos fontes ou através de pacotes pré-compilados.

Como observado anteriormente os *scripts ACID* são escritos em *PHP*, portanto, torna-se necessário adicionar suporte a *PHP* 4 ao servidor de *web*, bem como o suporte a *MySQL* para *PHP*. Existem diversas maneiras de realizar esta configuração, uma boa documentação sobre configuração do *Apache* pode ser

encontrada em [25]. Os recursos importantes para este ambiente são:

- **Suporte para banco de dados** – neste ambiente proposto o *MySQL*;
- **Suporte a *PHP* 4**;
- **Suporte para *GD*** – biblioteca gráfica utilizada na produção de gráficos;

Para certificar-se que o *Apache* e o *PHP* estão funcionando a contento, cria-se um arquivo chamado *teste.php* em */var/www/html* e adiciona-se o seguinte conteúdo:

```
<?php
phpinfo();
?>
```

Este arquivo deve ser acessado de um navegador qualquer digitando a *url http://IP_address/teste.php*. Deverá ser retornado informações do sistema *Apache* e *PHP*.

Para adicionar maiores funcionalidades ao *ACID* algumas bibliotecas devem ser instaladas, sendo que a *ADODB* é obrigatória e as outras podem ser omitidas, mas ocasionará deficiência nos recursos de produção de gráficos do *ACID*.

Já foi mencionada a biblioteca *GD*, que é utilizada para manipulação de imagem, suportando os formatos *GIF*, *JPEG* e *PNG*. Ela pode ser encontrada em http://www.boutell.com/gd/. A biblioteca *GD* apresenta as seguintes dependências:

- *libpng*, disponível em http://www.libpng.org/;
- *libjpeg-6d*, disponível em http://www.ijg.org/;
- *zlib*, disponível em http://www.gzip.org/zlib/.

O *ACID* utiliza a biblioteca *JPGraph*, que é um conjunto de *scripts PHP*, para produção de gráficos. Ela está disponível em http://www.aditus.nu/jpgraph/. Ela precisa ser simplesmente descompactada em um diretório onde o *PHP* possa acessar os *scripts*, normalmente em /

*var/www/html/jpgraph*.

É necessário também a instalação da biblioteca *ADODB*, que constitui uma camada de abstração para interação do *PHP* com o banco de dados. Ela pode ser encontrada em http://phplens.com/lens/dl/adodb453.tgz.

O *ACID* necessita de um usuário no banco de dados a fim de permitir a manipulação de dados. Os seguintes passos são necessários para criação deste usuário:

```
#mysql -u root -p
mysql> connect snort_db
mysql> grant CREATE, DELETE, UPDATE, INSERT, SELECT on snort_db.* to acid@localhost;
mysql> grant CREATE, DELETE, UPDATE, INSERT, SELECT on snort_db.* to acid@”%”;
mysql> connect  mysql
mysql> set password for 'acid'@'localhost' = password('senha');
mysql> set password for 'acid'@'%' = password ('senha');
mysql>flush privileges;
mysql> exit;
```

A instalação do *ACID* também é bem simples. É necessário apenas descompactar o conjunto de *scripts* em um local sob o diretório raiz da *web*. O *ACID* pode ser obtido em http://acidlab.sourceforge.net/.

Antes da utilização do *ACID* é necessário configurar alguns parâmetros no seu arquivo de configuração principal (*acid_conf.php*), localizado no diretório *ACID* do servidor *web*. A tabela 4.1 ilustra os parâmetros mais importantes.

**Tabela 4.1 – Parâmetros de Configuração do *ACID*.**

| *Parâmetros* | *Descrição* |
|---|---|
| *$Dblib_path* | Caminho para instalação do *ADODB*. (Obs.: Não incluir / ao final de qualquer variável de *path*) |
| *$ChartLib_path* | Caminho para instalação do *JPGraph* |
| *$chart_file_format* | As opções de formato de arquivo são *png*, *jpeg* e *gif*. |
| *$Dbtype* | Tipo de banco de dados |
| *$alert_dbname* | Nome do banco de dados de alertas |
| *$alert_host* | Servidor de banco de dados de alertas |
| *$alert-port* | Porta na qual o servidor de banco de dados esta escutando, se a porta é a padrão não é necessário configurar este parâmetro |
| *$alert_user* | Nome do usuário do banco de dados de alertas |
| *$alert_password* | Senha do nome de usuário |

Nesta proposta de ambiente esses parâmetros terão os seguintes valores:

$Dblib_path = “/var/www/html/adodb”;

$Dbtype = “mysql”;

$alert_dbname = “snort_db”;

$alert_host = “localhost”;

$alert_port = “”;

$alert_user = “acid”;

$alert_password = “senha”;

$ChartLib_path = “/var/www/html/jpgraph”;

$chart_file_format = “png”;

É recomendável proteger o acesso às páginas do *ACID* através de uma senha. Os passos a seguir configura um usuário e senha para acesso ao *ACID*.

```
#mkdir /etc/httpd/passwords
#htpasswd -c /etc/httpd/passwords/.htpasswd ids
```

Depois de criar as senhas para o usuário ids, é necessário adicionar as seguintes linha s no arquivo *httpd.conf*:

```
<Directory "/var/www/html/acid">
  Authtype Basic
  AuthName "ACID Console"
  AuthUserFile /var/www/passwords/.htpasswd
  Require user ids
  AllowOverride None
</Directory>
```

Após a realização das configurações anteriores torna-se necessário reinicializar o serviço *httpd*. Quando o *ACID* for acessado pela primeira vez, será pedido usuário e senha.

No primeiro acesso ao *ACID*, a tela ilustrada pela figura 4.4 será mostrada. Esta tela apresenta um erro, que significa que algumas tabelas estão faltando. O *ACID* adiciona essas tabelas extras no banco de dados, para isso basta clicar no *link* ***Setup Page***. A seguir o *ACID* apresentará a tela ilustrada pela figura 4.5, tornando necessário clicar no botão ***Create ACID AG*** para que seja efetuada a criação das tabelas.

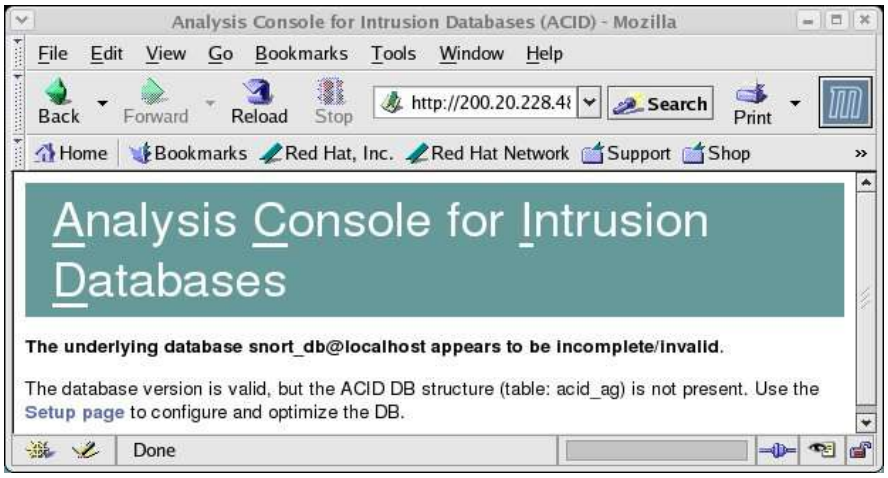


**Figura 4.4** – Criação das tabelas extras para o *ACID*

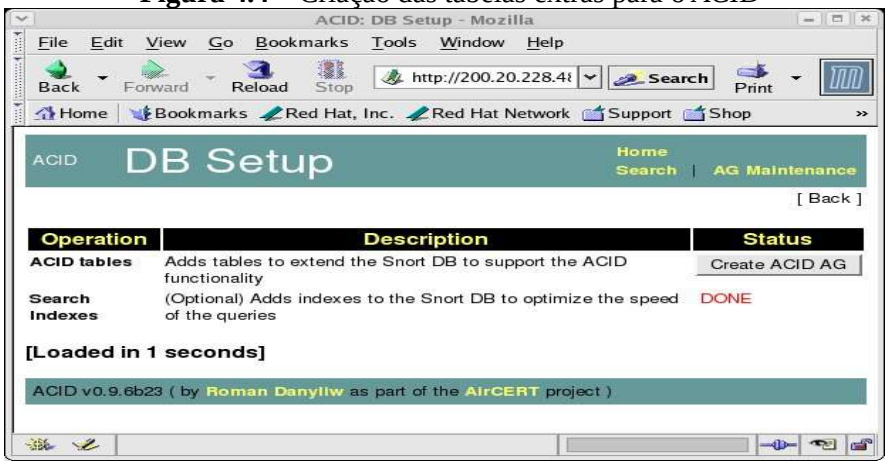


**Figura 4.5** – Configuração das tabelas

A partir da criação das tabelas extras pode-se utilizar o programa. Usar o *ACID* é muito simples. Suas telas são auto-explicativas e bem intuitivas. A figura 4.6 ilustra a tela principal do *ACID*.

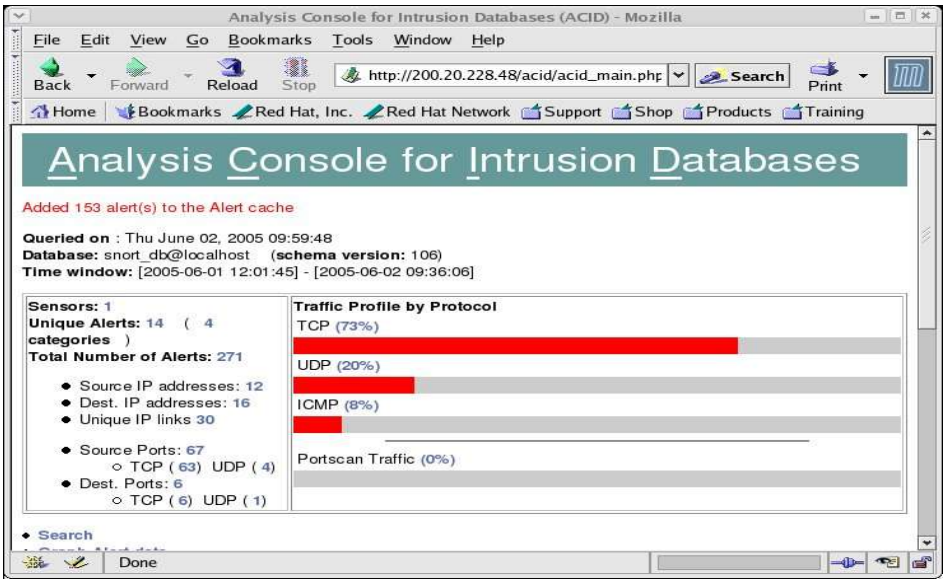


**Figura 4.6** – Tela principal do *ACID*

Esta ferramenta apresenta estatísticas de alertas, permitindo descobrir o número de acessos divididos por protocolo, por portas de origem e destino, entre outras informações. Cada pacote individual registrado pode ser exibido em um formato decodificado, mostrando várias *flags*, opções e conteúdos dos pacotes.

Um importante recurso do *ACID*, é sua ferramenta de pesquisa. É permitido a realização de consultas por diversos parâmetros, entre eles, assinatura de ataque, protocolo, endereço *IP*, data etc. Os alertas exibidos podem ser adicionados em um grupo de alertas, excluídos do banco de dados, enviados por *e-mail*, ou arquivado em outro banco de dados.

Para maiores informações sobre instalação de um servidor SDI com *Snort* e *Acid* consulte [12].

### 4.2.3 Ferramenta de bloqueio automático de ataques

O *Snort* é uma ferramenta que gera alertas das tentativas de invasão. O *Snort* sozinho não faz o bloqueio automático de ataques. Para esta tarefa ou

compila-se o *Snort* com suporte ao módulo *inline*[8] ou utiliza-se uma ferramenta extra para realização desde bloqueio. Neste documento opta-se pela utilização da ferramenta extra chamada *Guardian*.

O *Guardian* tem a capacidade de ler os *logs* do *Snort* em tempo real, agindo contra um possível invasor no momento da tentativa. O *download* desta ferramenta pode ser realizado em http://www.snort.org/dl/contrib/other_tools/guardian/.

Após a realização do *download* e de sua descompactação, torna-se necessário copiar o arquivo *guardian.pl* para algum diretório do *path*, por exemplo, /*usr/bin*. Deve-se também copiar o arquivo *guardian.conf* para o diretório /*etc*. É necessário a criação do arquivo de *log* do *Guardian* no diretório */var/log.*

Também torna-se necessário a realização de algumas alterações no arquivo *guardian.conf*, para que o *Guardian* funcione corretamente. Entre os ítens que devem ser alterados destaca-se os seguintes:

- ***HostIpAddr*** – Endereço *IP* visível para Internet. Só deve ser informado se o *host* tem *IP* estático.
- ***Interface*** – interface que terá os *host* hostis barrados (*eth0* ou *eth1* ou *ethN*);
- ***AlertFile*** – arquivo onde o *Guardian* lerá os alertas do *Snort*, normalmente, */var/log/snort/alert;*
- ***TimeLimit*** – tempo que o *host* ficará barrado no *firewall*.

O parâmetro *TimeLimit* deve ser bem analisado,pois dependendo do valor informado pode permitir um ataque *Denial of Services*, pois se o invasor estiver *spoofando* o *IP*, o *Guardian* pode negar o acesso a diversas máquinas.

8 A partir da versão 2.3.0 RC1 foi integrado ao *Snort* a capacidade de *Intrusion Prevention System* (*IPS*). *Snort inline* obtém pacotes do *Iptables* ao invéz da L*ibpcap* e então usa novo tipos de regras para ajudar o *Iptables* passar ou negar pacotes baseados nas regras do *Snort*. Maiores informações consulte http://www.snort.org/docs/snort_htmanuals/htmanual_233/node7.html.

Deve-se analisar o intervalo de bloqueio, um bom valor pode ser 7200 (duas horas).

O *Guardian* apresenta *scripts* para realizar o bloqueio compatíveis com os *firewalls Iptables*, *Ipchain* e *Ipfw*. Deve-se copiar o *script* apropriado para o *firewall* instalado para um diretório do *path*, por exemplo, */usr/bin*, renomeando o *script* de bloqueio para *guardian_block* e o de desbloqueio para *guardian_unblock.sh*.

Pode-se criar um arquivo chamado */etc/guardian.ignore*, com os *IPs* que devem ser ignorados pelo G*uardian* na hora da realização do bloqueio. Certifique-se de adicionar todos os *IPs* que julgue necessário. Uma boa opção é o endereço de *loopback* e dos servidores da rede.

Para inicializar o *Guardian*, basta digitar na linha de comando *guardian.pl -c /etc/guardian.conf*. É interessante inicializar o *Guardian* na inicialização do sistema. Uma boa opção é adicionar o referido comando em */etc/rc.local*. Maiores informações sobre o *Guardian* pode ser obtidas em http://www.opencode.com.br/modules.php?name=Sections&op=viewarticle&artid=3.

### 4.2.4 Atualização automática de regras

Como já relatado neste documento, o *Snort* realiza a detecção de invasão através da comparação dos pacotes capturados com sua base de regras (assinaturas). Torna-se muito importante mantê-las atualizadas.

A atualização pode se dá pela adição de regras novas ou por modificações das regras antigas. Manter a base de regras atualizadas minimiza os falsos positivos e negativos.

Existem várias fontes de regras para o *Snort*, sendo a principal delas localizada no endereço www.snort.org. A partir de março de 2005 o *SourceFire*[9] alterou a licença das regras do *Snort.* Para consultar esta licença acesse

9 http://www.sourcefire.com/

http://www.snort.org/about_snort/licenses/vrt_license.html. A partir desta data existem duas diferentes formas gratuitas de realização do *download* das regras do *Snort* em seu *site* oficial.

A primeira opção é baixar as regras da *SourceFire VRT Certified Rules* cinco dias após seu lançamento, visto que, no ato do lançamento essas regras são liberadas somente para os parceiros. Para este tipo de acesso é necessário a realização de registro no *site*, para obtenção do código de *donwload*. A segunda forma é baixar as regras disponibilizadas pela comunidade via *GPL*[10].

Existe uma ferramenta chamada *Oinkmaster* que é um *script Perl* desenvolvido para automatizar o processo de *donwload* e mesclagem de regras do *Snort*. Esta ferramenta possibilita a atualização das duas forma a cima citadas.

O download desta ferramenta pode ser realizado em http://oinkmaster.sourceforge.net/download.shtml. Seu processo de instalação e configuração é bastante simples, basta descompactá-la e copiar o arquivo *oinkmaster.pl* para um diretório do *path*, por exemplo, */usr/bin*, e copiar o arquivo *oinkmaster.conf* para o diretório */etc*.

A configuração padrão do *oinkmaster.conf* deve funcionar para a maioria dos usuários, bastando apenas informar o parâmetro *url*. Este parâmetro especifica a fonte das regras do *Snort*. Este arquivo de configuração é bem intuitivo e comentado tornando fácil seu entendimento e configuração.

Após as devidas configurações basta inicializar o *Oinkmaster* informando o diretório das regras do *Snort*, supondo que o diretório seja */etc/snort/rules* o *Oinkmaster* deve ser inicializado da seguinte forma:

```
oinkmaster.pl -c /etc/snort/rules
```

É recomendável a realização do *backup* das regras atuais antes de sua atualização. Como as regras do *Snort* devem estar permanentemente atualizadas,

10 *General Public License*

uma boa sugestão é programar a tarefa de atualização através do programa *Cron*. Para maiores informações sobre o *Oinkmaster* consulte http://oinkmaster.sourceforge.net/docs.shtml.

# 5 Validações e Resultados

## 5.1 Introdução

Este capítulo apresenta o ambiente utilizado para validação do servidor SDI proposto implementado sobre o sistema operacional *Linux*. Apresenta-se também as ferramentas utilizadas nos testes e os resultados obtidos.

## 5.2 Ambiente implementado

Como laboratório para a solução proposta utiliza-se a rede da UENF[11]. O ambiente de teste e validação é composto por um sensor *Snort*, implementado sobre a distribuição *Fedora,* contendo os seguintes *sofware* instalados:

- *Snort* 2.3.3;
- *Iptables* 1.2.9;
- *ACID* 0.9.6b23;
- *Guardian* 1.6;
- *Apache* 2.0.51-2.9;
- *Mysql* 3.23.58;
- *PHP* 4.3.10-2.4.

Este sensor além da funcionalidade de segurança proposta pelo ambiente fornece os serviços *NAT*[12] e *DNAT*[13], criando uma *DMZ* [14]. Tais serviços permitem que as máquinas protegidas pelo *firewall* acessem a Internet, bem como permite acesso externo aos serviços providos pelos servidores da rede privada.

11 Universidade Estadual do Norte Fluminense
*12 Network Address Translation*
*13 Destination Network Address Translation*
14 Zona Desmilitarizada – Segmento de rede que prove serviços de internet, acessados externamente, controlados por um *firewall* e pelas regras de segurança, podendo ser constantemente auditado.

Na rede privada foram instalados dois servidores. Um provendo serviço de *web* e *DNS* e outro provendo serviço de *e-mail*. A figura 5.1 ilustra este ambiente.

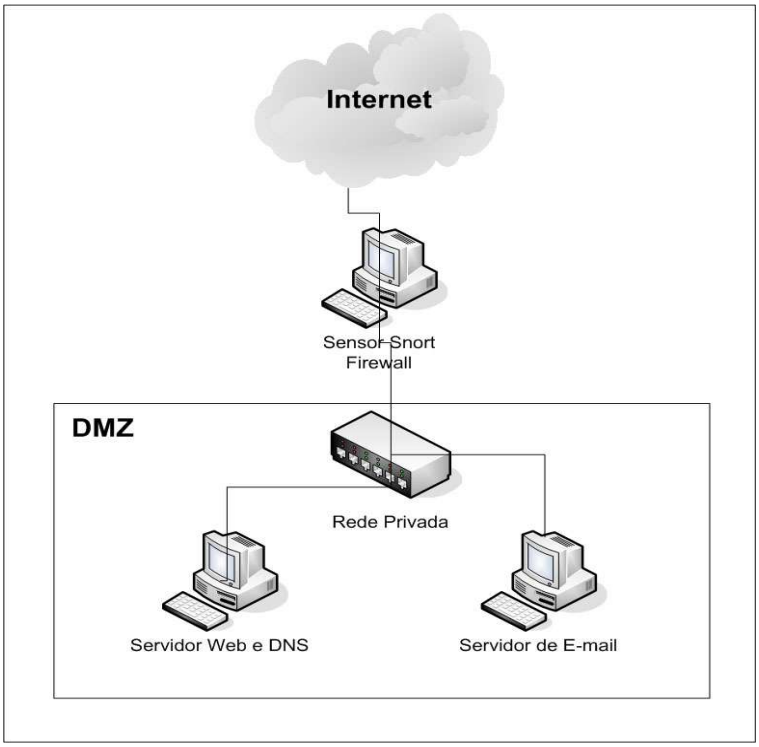


**Figura 5.1** - Ambiente Implantado

Para validação do sensor *Snort* implementado utiliza-se duas ferramentas de segurança, o *Nmap e* o *Nessus e* um *exploit* para explorar um vulnerabilidade conhecida da dupla *Apache ModSSL*.

O *Nmap* é uma ferramenta de segurança que permite ao administrador ou possíveis invasores vasculhar uma rede verificando quais servidores estão ativos e quais serviços eles estão oferecendo. O *Nmap* fornece diversas tecnologias de *scanner*, como por exemplo, *UDP*, *TCP connect*, *TCP SYN*, *ICMP* entre outras. Maiores informações sobre *Nmap* pode ser obtidas em [23].

O *Nessus* é uma ferramenta que realiza varreduras de segurança,

também conhecido como *scanner* de segurança. Um *scanner* de segurança é um programa que verifica uma máquina ou uma rede em busca de brechas de segurança que possam vir a ser usadas por um *cracker*. Com esta ferramenta é possível verificar várias vulnerabilidades na rede, realizando auditoria de segurança tanto nas estações como nos servidores.

O *Nessus* pode ser utilizados para descobrir *Worms* e *Backdoor* instalados em estações de trabalho ou servidores. Ele disponibiliza vários *scripts*/*plugins* que permitem a verificação de uma grande gama de serviços de redes. Maiores informações sobre o *Nessus* podem ser obtidas em [21].

Ferramentas deste tipo nada mais são que um sistema que analisa se um computador tem portas *UDP* ou *TCP* abertas. Elas fazem uma varredura pelas portas existentes nesses protocolos.

A forma como essa detecção é realizada varia de programa para programa e pela forma como cada um é utilizado. De uma forma geral, essas ferramentas enviam pacotes para um servidor direcionando para cada uma das portas existentes. Quando a ferramenta recebe uma confirmação do pacote, ela detecta que o serviço que opera naquela porta esta ativo.

A outra técnica de teste realizada foi através da utilização de um *exploit*. Informações sobre *exploit* são encontradas na seção 2.2.5. O *exploit* utilizado para testar o servidor SDI explora a vulnerabilidade conhecida como "*OpenSSL SSLv2 Malformed Client Key Remote Buffer Overflow*". Para maiores informações sobre esta vulnerabilidade consulte http://securityresponse.symantec.com/avcenter/security/Content/2002.09.13.html.

## 5.3 Resultados dos testes com *Nmap*

Nos teste realizados como *Nmap* utiliza-se o *Nmapfe*, que nada mais é

que um *front end* para o *Nmap*, que facilita sua utilização, visto que o *Nmap* apresenta várias opções de parâmetros. Na primeira varredura ocorre com o *Guardian* desabilitado, tendo somente o *Snort* registrando as tentativas de *scanner*. Na figura 5.2 pode ser observado o resultado da execução do *Nmapfe*. A seguir na figura 5.3 observa-se os *logs* do *Snort* visualizado via *ACID*.

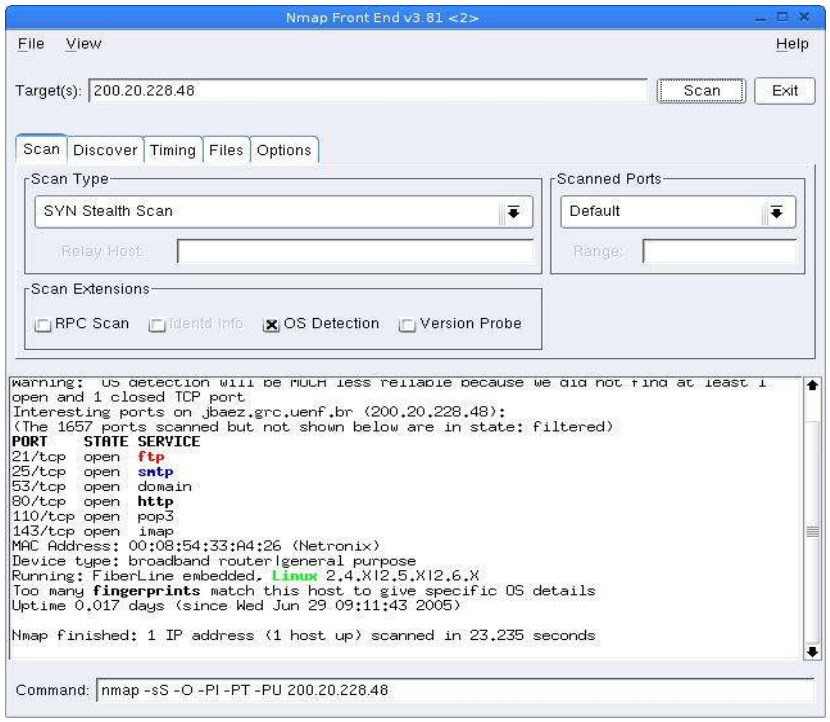


**Figura 5.2** – Resultado do *Nmap* sem *Guardian*

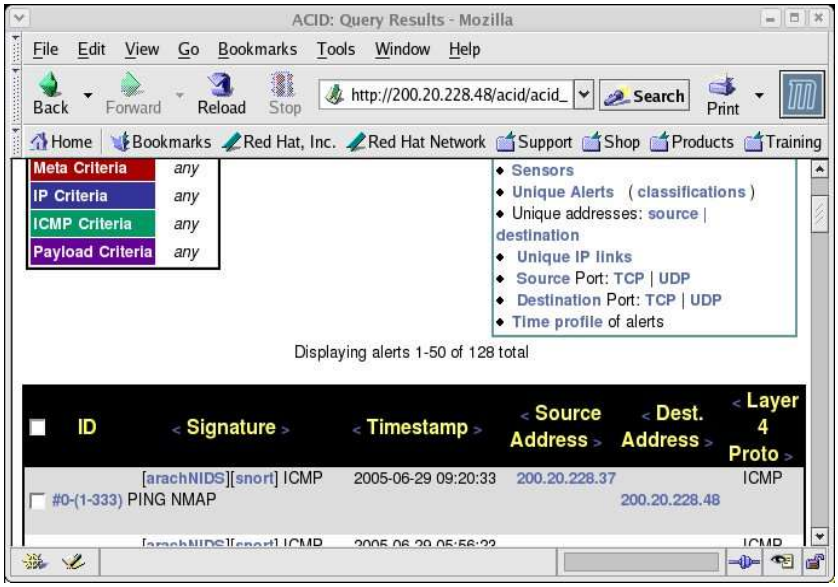


***Figura 5.3*** *– Log* do *Snort* da tentativa de varredura visualizado via *ACID*

Após o teste anterior, habilita-se o *Guardian* para que proceda o bloqueio das tentativas de varredura. O *Guardian* trabalha lendo os *logs* do *Snort*, e como o *Snort* registra as tentativas, só há registros após o início de uma varredura. É possível ainda detectar algumas portas, como pode ser observado na figura 5.4. Isso ocorre porque existe um *delay* entre o registro do *Snort* e a leitura do *Guardian*. A figura 5.5 ilustra os *logs* do *Guardian*, registrando a tentativa de varredura.

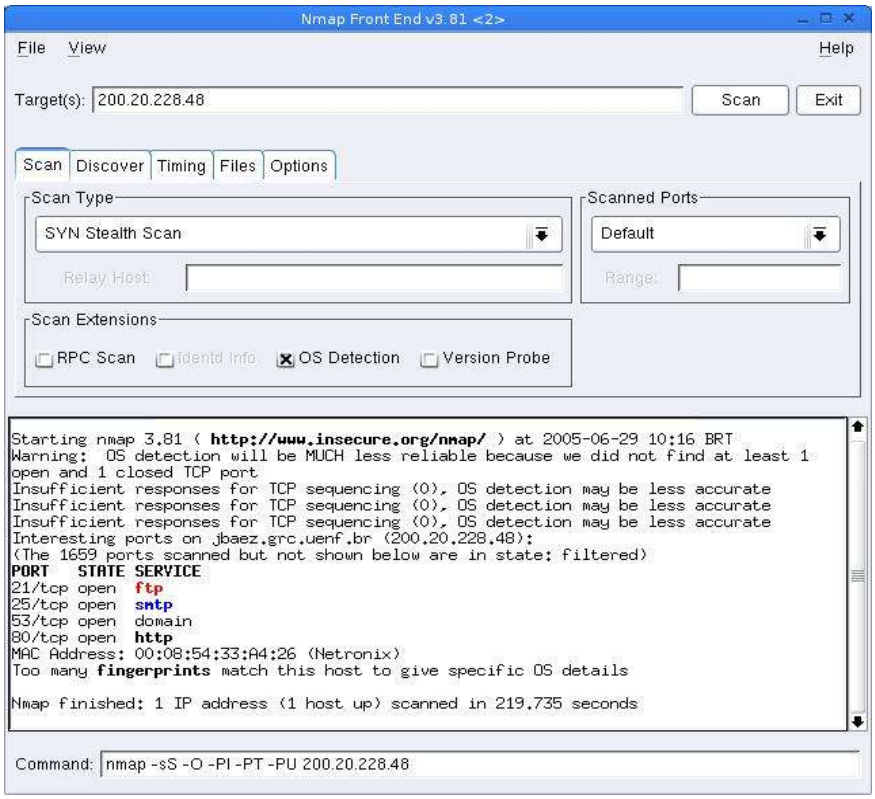


**Figura 5.4** – Resultado do *Nmap* com *Guardian* habilitado

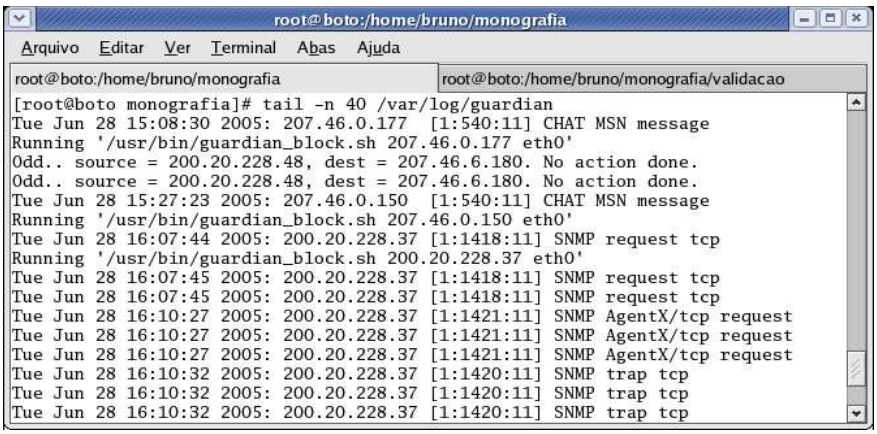


**Figura 5.5** – *Log* do *Guardian* registrando varredura do *Nmap*

Após o realização do bloqueio pelo *Guardian*, o *Nmap* não consegue

realizar a varredura, como pode ser observado pela figura 5.6. A figura 5.7 ilustra as regras de bloqueio adicionadas ao *Iptables*. Como pode ser observado foram adicionadas duas regras, uma na *chain input* e outra na *forward*, isso é necessário, visto que, o serviço *DNAT* esta habilitado no servidor *Snort*. Para isso foi necessário adicionar as seguintes linha nos arquivos */usr/sbin/guardian_block.sh* e no arquivo */usr/sbin/guardian_unblock,.sh* respectivamente.

/sbin/iptables -I FORWARD -s $source -i $interface -j DROP

/sbin/iptables -D FORWARD -s $source -i $interface -j DROP

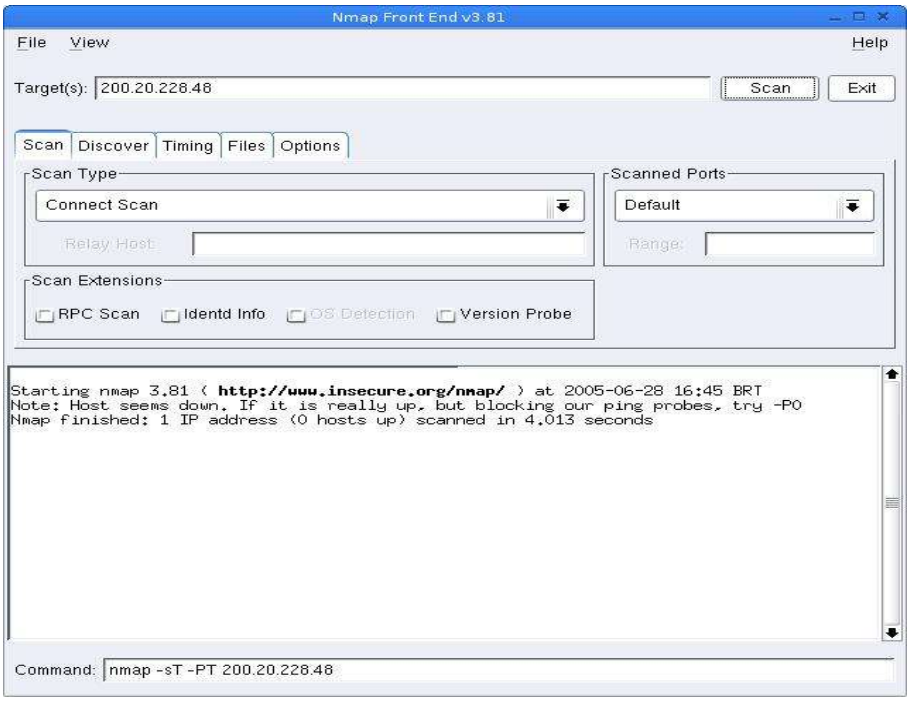


**Figura 5.6** – Varredura do *Nmap* após bloqueio do *Guardian*

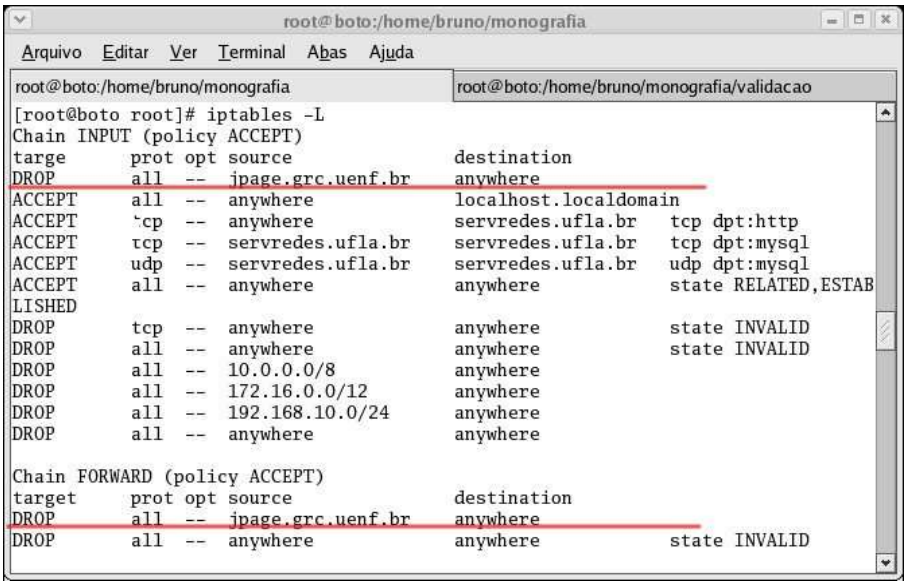


**Figura 5.7** – Regra adicionada ao *Iptables*

## 5.4 – Resultados obtidos com o *Nessus*

Os testes realizados com o *Nessus* foram executados sobre um ambiente gráfico. O *Nessus* apresenta interface amigável e de fácil utilização. Ele é dividido em dois módulos, o servidor (*nessusd*) e o cliente (*nessus*). A utilização do cliente sobre um ambiente gráfico facilita bastante o manuseio da ferramenta.

Realiza-se os mesmos testes realizados com a ferramenta *Nmap*, inclusive seguindo a mesma ordem de passos. Os resultados podem ser observados pelas figuras 5.8, 5.9, 5.10, 5.11 e 5.12.

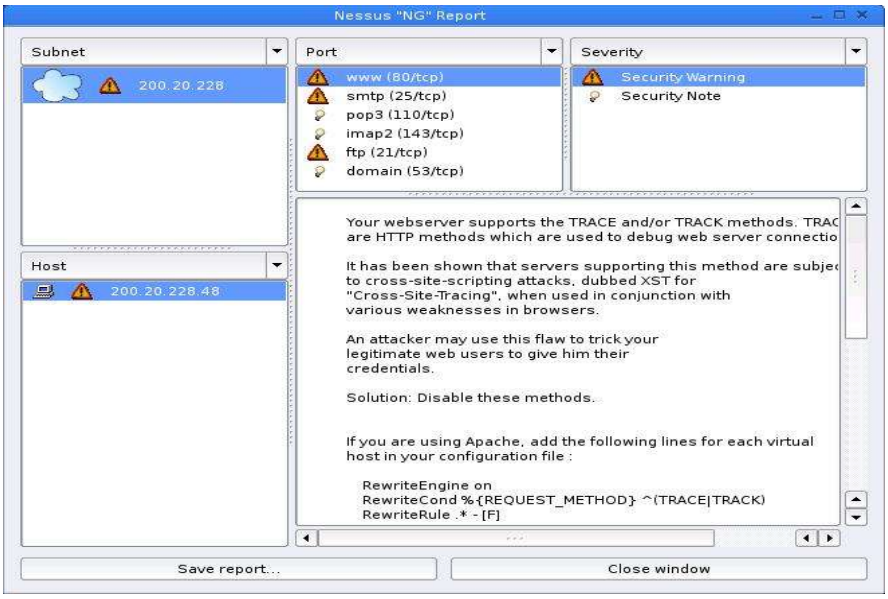


**Figura 5.8** – Resultado do *Nessus* com *Guardian* desabilitado

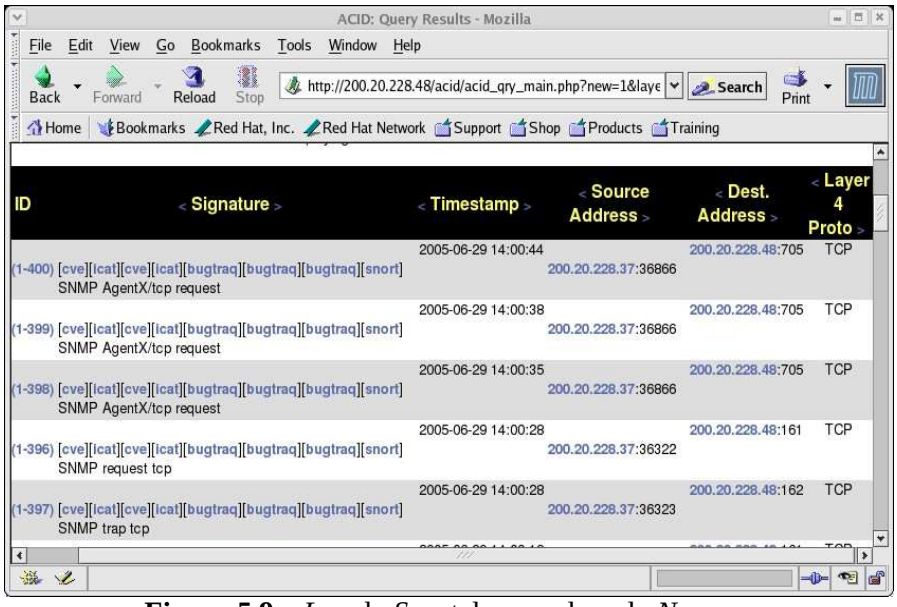


**Figura 5.9** – *Log* do *Snort* da varredura do *Nessus*

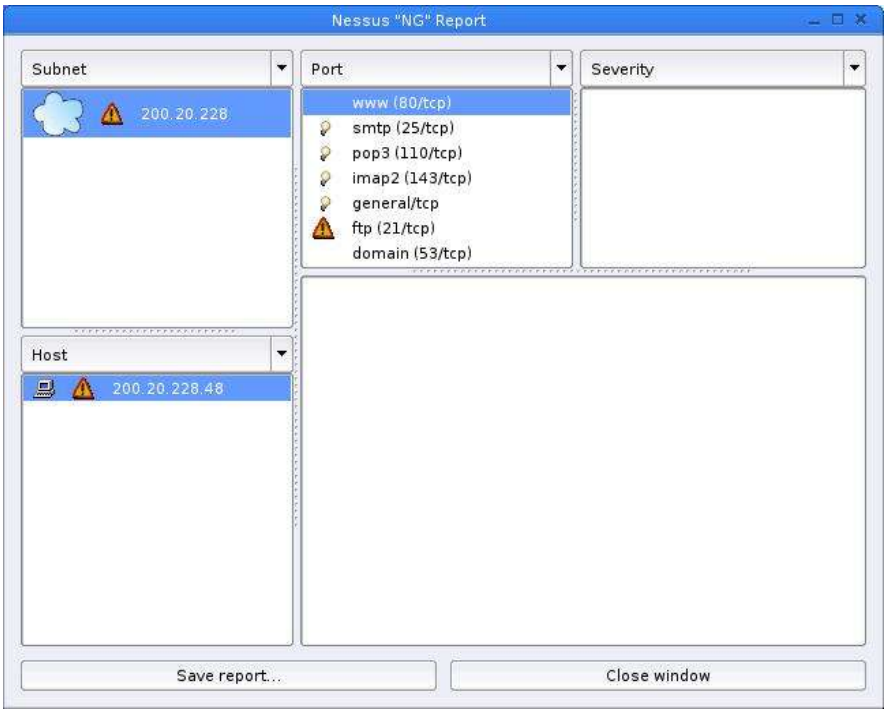


**Figura 5.10** – Resultado do *Nessus* com *Guardian* habilitado

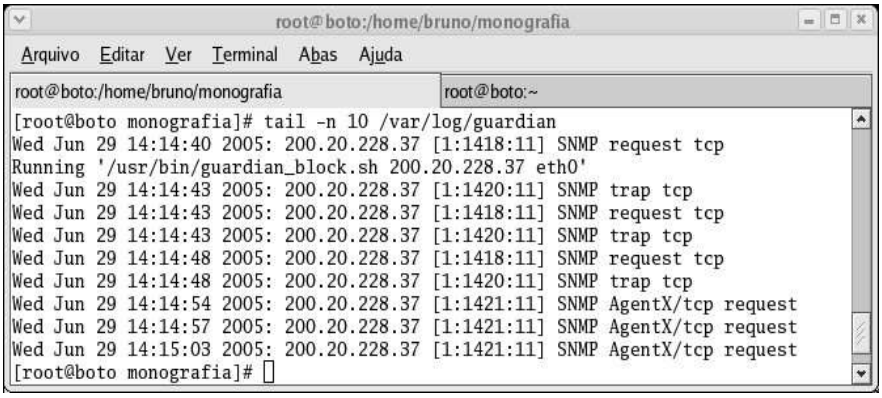


**Figura 5.11** – *Log* do *Guardian* registrando varredura do *Nessus*


**Figura 5.12** – Resultado da varredura do *Nessus* após bloqueio do *Guardian*

Como pode ser observado na figura 5.10, após o carregamento do *Guardian*, o *Nessus* conseguiu realizar o *portscan* em todas as portas. O que não foi possível foi a realização do *check* de vulnerabilidades, visto que não houve nenhum *Security Warning* nas portas 80 e 25, como nos teste com o *Guardian* desabilitado, isso porque neste momento o *Guardian* já tinha adicionado a regra de bloqueio.

## 5.5 Resultados Obtidos com a execução do *exploit*

Com a execução do *exploit* tenta-se explorar uma vulnerabilidade do *Apache* com *ModSSL*, verificando se o servidor SDI instalado identifica e bloqueia esta tentativa de ataque. Diferente dos testes anteriores, este teste tenta atacar um serviço ativo específico.

O *exploit* utilizado é um arquivo binário de nome *apachexpl* que é executado em linha de comando como descrito abaixo:

```
./apachexpl auto IP_DESTINO
```

O teste com o *exploit* foi realizado com o *Guardian* ativo. Como pode ser observado pelas figuras 5.13 e 5.14, a tentativa de exploração foi identificada e bloqueada pelo servidor SDI. Na figura 5.14 observa-se que o *Guardian* bloqueia a origem do ataque no *firewall* através da execução do *script guardian_block.sh.*

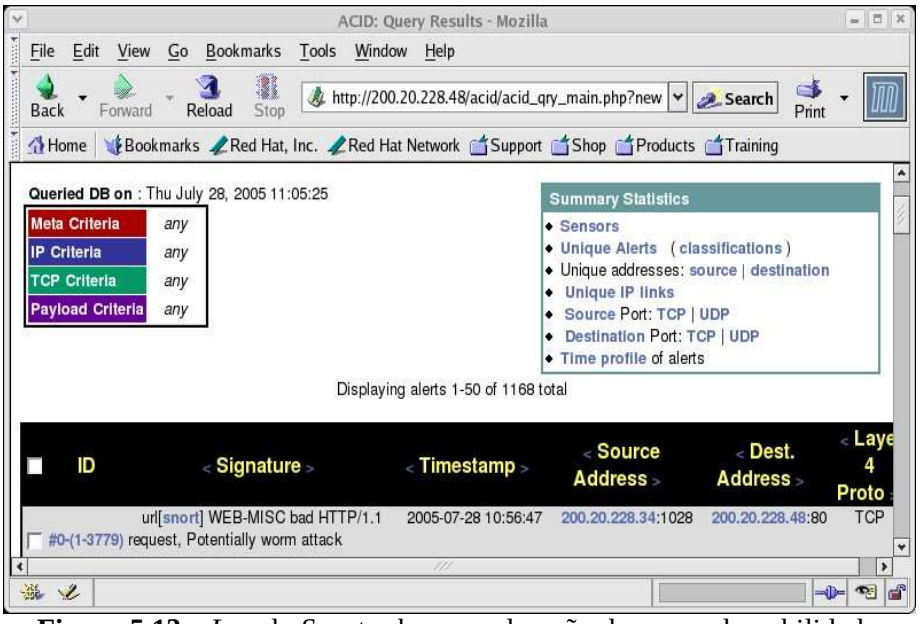


**Figura 5.13 –** *Log* do *Snort* sobre a exploração de uma vulnerabilidade

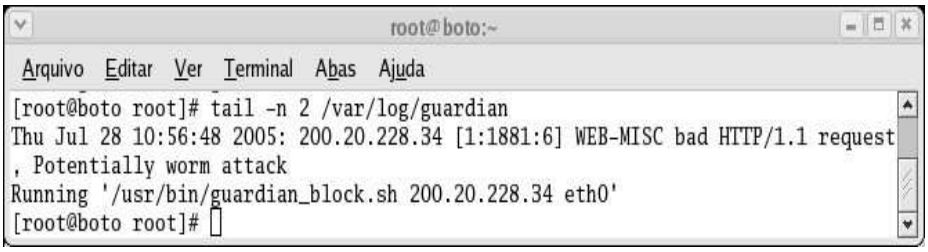


**Figura 5.14** – *Log* do *Guardian* sobre a exploração de uma vulnerabilidade

Com os teste realizados, conclui-se que a dupla *Snort* e *Guardian* acrescenta um bom nível de segurança para um sistema computacional. Em todos os testes foi possível detectar e bloquear as tentativas de varredura e exploração de vulnerabilidades.

# 6 Conclusão

O presente documento apresentou uma proposta de segurança de rede, através da implementação de um servidor SDI. Este servidor é baseado nos aplicativos *Snort, Guardian e ACID*, garantindo a detecção das invasões, o bloqueio de ataques durante as tentativas e uma leitura facilitada dos *logs*. Com esta implementação, consegue-se acrescentar um nível a mais de segurança à rede, uma vez que há a detecção e o bloqueio de uma grande gama de ataques existentes.

Neste documento, foram apresentados conceitos e definidos procedimentos para a instalação e configuração do SDI *Snort* e dos aplicativos auxiliares que compõem a implantação de um servidor SDI.

Ressalta-se que a proteção de uma rede não é uma tarefa singular e muito menos simples. Não é a implantação de um servidor SDI a solução de todos os problemas de segurança. Problemas complexos requerem soluções complexas. Não se pode imaginar segurança de rede, sem um trabalho contínuo de atualizações dos aplicativos e das ferramentas utilizadas.

É um erro enorme pensar que com a implementação de uma solução de segurança a rede estará protegida, não existindo mais problemas e preocupações com assuntos ligados à segurança da informação. O administrador deve estar constantemente preocupado com as atualizações e vulnerabilidades descobertas sobre os serviços implantados.

Como abordado anteriormente, opta-se pela implantação de um sensor SDI através da utilização do *Snort*. Vale ressaltar que apesar do *Snort* ser uma ferramenta destinada à segurança de rede, ele é um *software* e como todo *software* está sujeito a falhas. Deve-se ficar constantemente atento aos seus boletins de segurança, atualizações e vulnerabilidades descobertas.

Nos testes realizados o servidor SDI mostrou-se seguro, detectando e

bloqueando as tentativas de invasão a ele submetida. Uma ressalva que se faz é que o *Guardian* não opera no modo *inline*, ou seja, ele bloqueia depois que pelo menos alguns pacotes já passaram. Esse *delay* não é nada preocupante, pois com poucos pacotes não se concretiza um ataque.

O *Snort* é um SDI muito utilizado na atualidade, existindo muitas ferramentas que podem trabalhar junto a ele. Em virtude disso, sugere-se os seguintes trabalhos futuros:

- Utilização do *BASE* (*Basic Analysis and Security Engine*), para analise de *logs*. O *BASE* é baseado no código do *ACID*. Ele pode ser considerado a evolução do *ACID*. Maiores informações sobre esta ferramenta podem ser obtidas em http://secureideas.sourceforge.net;
- Implementação do *Snort* no modo *inline*. O modo *inline* permite ao *Snort* previnir as invasões, em vez de somente detectá-las. No modo *inline* o *Snort* obtém os pacotes do *Iptables* ao invés da biblioteca *Libpcap*. Desde modo o *Snort* trabalha em conjunto com o *firewall*, bloqueando e permitindo pacotes baseados nas regras do *Snort*. Maiores informações sobre o modo *inline* podem ser obtidas em http://www.snort.org/docs/snort_htmanuals/htmanual_233/node7.html;
- Utilização do *SnortSam*, que é um *plugin* para o *Snort* de código aberto. Este *plugin* permite o bloqueio automático de ataques em diversos *firewalls*;
- Acoplar o *Snort* a um *honeyPots*. não bloqueando mais um ataque mas sim o direcionando para um *honeypot*, afim de obter mais informações forenses.

O *Snort* mostrou-se ser uma ferramenta bastante poderosa para prover segurança em uma rede, constituindo uma ótima alternativa para implementação de um sensor SDI. Como toda ferramenta ela não é infalível, da mesma forma que não existe rede 100% segura. Deve-se manter atento com os assuntos relacionados à segurança da informação.

# 7 Referências Bibliográficas

**[1]** Almeida, A. R., **Como Funcionam os Exploits**. Firewall Security Corporation, 2003. Disponível na Internet via url: http://www.firewalls.com.br/files/alexisExploit.pdf. Consultado em Fevereiro 2005.

**[2]** Cholewa, R., **Segurança em Redes – Conceitos Básicos**. Setembro, 2001. Disponível na Internet via url: http://www.rmc.eti.br/documentos/tutorias/tutorial_segurança.pdf. Acessado em Dezembro 2004.

**[3]** Caswell, B., **Snort 2 – Sistema de Detecção de Intruso.** AltaBooks, 2003.

**[4]** CERT, **Advisory CA-1995-01 IP Spoofing Attacks and Hijacked Terminal Connections.** Disponível em http://www.cert.org/advisories/CA-1995-01.html Acesso em Abril 2005.

**[5]** CERT, **Advisory CA-1996-26 Denial-of-Service Attack via ping** Disponível na Internet via url http://www.cert.org/advisories/CA-1996-26.html Acesso em Abril 2005.

**[6]** CERT, **Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks.** Disponível na Internet via url http://www.cert.org/advisories/CA-1996-21.html Acesso em Abril 2005.

**[7]** CERT, **Advisory CA-1996-01 UDP Port Denial-of-Service Attack.** Disponível em http://www.cert.org/advisories/CA-1996-01.html Acesso em Abril 2005.

**[8]** CERT, **Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks.** Disponível na Internet via url http://www.cert.org/advisories/CA-1998-01.html Acesso em Abril 2005.

**[9]** Freiss, M., **Proteção de Rede com SATAN**, Ciência Moderna, 1999.

**[10]** Fx, Project by ThinkSec., NAI, Labs., **Pam_passwdqc(8) - Linux man page.** Disponível na Internet via url http://www.die.net/doc/linux/man/man8/pam_passwdqc.8.html Acessado em Maio 2005.

**[11]** Firewall Security Corporation, **Explorando Buffer Overflows – Técnicas de Invasão.** Disponível na Internet via url: http://www.firewalls.com.br/files/buffer.pdf. Acessado em Fevereiro 2005.

**[12]**Harper, P., **Snort, Apache, SSL, PHP, MySQL, Acid, Install on Fedora Core 2.** Internet Security Guru, Outubro 2004. Disponível na Internet via url http://www.internetsecurityguru.com/documents/Snort_SSL_FC2.pdf. Acessado em Janeiro 2005.

**[13]** International Network Services**, Microsoft IIS Unicode Exploit.** INS, Agosto de 2001. Disponível na Internet via url http://www.ins.com/downloads/whitepapers/ins_white_paper_ms_iis_unicode_exploit_0801.pdf. Acessado em Maio de 2005.

**[14]** Knobbe, F., **Documentação do SnortSam**. SnortSam, Março 2005. Disponível na Internet via url http://www.snortsam.net/documentation.html. Acessado em Abril 2005.

**[15]** Lari, P. A., Amaral, D. M., **Guia Rápido do Administrador de Redes: Snort, MySQL, Apache e ACID**. Brasport, 2004.

**[16]** Morgan, A. G., **The Linux PAM System Administrator's Guide**. Open-PAM working group, Setembro 2003. Disponível na Internet via url http://www.kernel.org/pub/linux/libs/pam/. Acessado em Março 2005

**[17]** Montoro, R., R., **Funcionamento do Snort**. Spooker Labs, Abril 2004. Disponível na Internet via url www.spooker.com.br/papers/como-funciona-snort.pdf. Acessado em Janeiro 2005.

**[18]** Ortale, P. N., **Snort IDS.** Disponível na Internet via url: http://www.linuxsecurity.com.br/sections.php?op=viewarticle&artid=10. Consultado em 2004.

**[19]** Pinheiro, A. C., **Uso e Análise do Tripwire**. LinuxIT, 2003. Disponível na Internet via url: http://www.linuxit.com.br/modules.php?name=Sections&op=viewarticle&artid=450. Acessado em Dezembro 2004.

**[20]** Rech F. S., **Verificando a Integridade do sistema com AIDE.** Linuxbsd.com.br, 2003**.** Disponível na Internet via url:http://www.linuxbsd.com.br/phpLinuxBSD/modules/artigos_tecnicos/aide.htm. Acessado em Fevereiro 2005.

**[21]** Reis A., **Cuidando da Casa – Nessus**. Linuxtemple.com, Outubro 2004. Disponível na Internet via url: http://www.linuxtemple.com/index.php?option=content&task=view&id=530. Acessado em Novembro 2004.

**[22]** Russel P., **Linux IPCHAINS-HOWTO.** Linuxit.com, Março de 1999. Disponível na Internet via url:http://www.linuxit.com.br/downloads/ipchains.txt. Acessado em Fevereiro de 2005.

**[23]** Sartini, H. S., **Usando Nmap**. Disponível na Internet via url:http://web.onda.com.br/humberto/arquivo/nmap.pdf. Acessado em Setembro 2005.

**[24]** Samar, V., Schemers, R., **Unified Login with Pluggable Authentication Modules (PAM)**. Open Software Foundation, Outubro 1995. Disponível na Internet via url: http://www.opengroup.org/tech/rfc/mirror-rfc/rfc86.0.txt. Acessado em Março 2005.

**[25]** Sica, F. C., Simeone, L. E., Uchôa, J. Q., **Administração de Redes Linux.** Lavras, UFLA/FAEPE, 2003. (Curso de pós Graduação “Lato Sensu” em Administração de rede linux).

**[26]** Sica, F. C., Uchôa, J. Q., **Administração de Sistemas Linux.** Lavras, UFLA/FAEPE, 2003. (Curso de pós Graduação “Lato Sensu” em Administração de rede linux).

**[27]** Uchôa, J. Q., **Segurança em Redes e Criptografia.** Lavras, UFLA/FAEPE, 2003. (Curso de pós Graduação “Lato Sensu” em Administração de rede linux).

**[28]** Vianna, W. A., **Proposta de Implementação de Segurança para Redes Locais com Acesso a Internet.** Lavras, UFLA, 2004. (Monografia Apresentada ao Departamento de Computação da UFLA para obtenção de Título de especialista em Administração de Redes Linux).