UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ALEXANDRE ANDRÉ CORSO

Instalação e Utilização de um Sistema de Detecção de Intrusão

Trabalho de Graduação.

Prof. Dr. Raul Fernando Weber
Orientador

Porto Alegre, dezembro de 2009.

AGRADECIMENTOS

Agradeço especialmente aos meus filhos por compreenderem que as horas de ausência tinham um motivo nobre e a minha esposa por me convencer a não desistir, quando eu mesmo já estava convencido. Agradeço ao professor Lisandro pela paciência e incentivo no TC1 e ao professor Weber, por não deixar que eu fugisse novamente do TC2.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

IDS	*Intrusion detection system*
DoS	*Denial of Service*
ER	Entidade Relacionamento
IP	*Internet Protocol*
TCP	*Transmission Control Protocol*
UDP	*Universal Datagram Protocol*
HTTP	*Hypertext Transfer Protocol*
FTP	*File Transfer Protocol*
ICMP	*Internet Control Message Protocol*
SMTP	*Simple Mail Transfer Protocol*
IIS	*Internet Information Services*
URL	*Uniform Resource Locator*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O aumento das transações eletrônicas através da Internet mesma e os valores agregados a informação, fazem com que a busca por melhores estratégias de segurança tem aumentem consideravelmente.

Em função disso, a comunicação segura através da rede se tornou muito valiosa. Ataques podem causar prejuízos e até mesmo manchar a imagem de uma empresa.

Existem diversas técnicas e métodos de proteção à informação, dentre os quais podemos 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 a implementação um Sistema de Detecção de Intrusão, através da solução de código aberto Snort.

Installation and use of an intrusion detection system

ABSTRACT

The growing in electronic transaction through the Internet and the values aggregated to information, the search for better security strategies has grown considerably.

Because of that, safe communication throughout the net became very valuable. Attacks may cause harm and even tarnish the image of a company.

There are many techniques and information protection methods, among which we can mention firewalls, backups, anti-viruses, invasion detection systems and others.

In this information war, be it for protection or to gain access to it, the administrator must be prepared to win it over. To know the invasion techniques and the protection tools is vital for a good administrator.

In this work it is made an implementation of an Intrusion Detection System, through use of the Snort open source code solution.

1 INTRODUÇÃO

O crescente uso da internet agregado ao grande número de redes corporativas existentes agravou o problema da insegurança. Intrusos utilizam-se de falhas de segurança nos sistemas e do descuido de usuários e administradores de rede a fim de obter informações pessoais, fazer propaganda ou até apagar informações necessárias.

Isso nos leva a desejar uma enorme necessidade de poder rastrear e identificar estes ataques. O sistema que possui a capacidade de fazer isso é o *IDS – Intrusion Detection System*, ou Sistema de Detecção de Intrusão. Ataque é uma ação inteligente que ameaça a segurança de um sistema. Um ataque pode ter sucesso ou não e estará explorando uma vulnerabilidade no sistema alvo ou inerente aos protocolos (RFC 2828). Um ataque bem sucedido pode caracterizar uma invasão ou até mesmo a negação de serviços no sistema alvo (DoS - *Denial of Service*).

Nesse sentido, um IDS tem uma função primordial no estudo das técnicas utilizadas para invasão e conseqüente auxilio ao administrador da rede no sentido de prevenir tais ataques. Existem hoje em dia várias soluções comerciais e em *software* livre que implementam a função de IDS. Em ambas, a instalação, configuração e utilização não são de solução trivial. Em função disto, esse estudo de caso tem por objetivo definir um roteiro para a execução desses passos por um usuário com conhecimentos razoáveis de rede e de sistemas operacionais.

No capítulo 2, apresentamos o *IDS Snort*, um ferramenta *open source* capaz de analisar o tráfego da rede em tempo real e, baseado em um vasto número de assinaturas de ataques conhecidos, identificar e registrar as tentativas de invasão. No capítulo 3 apresentamos as ferramentas que facilitam o registro das tentativas de invasão e são capazes de gerar estatísticas e apoiar o administrador da rede na tomada de ações de proteção. No capítulo 4, é definido um passo a passo para a instalação, configuração e utilização das ferramentas mencionadas nos capítulos 2 e 3.

2 SNORT

O Snort é uma ferramenta de detecção de intrusão de rede baseado em assinaturas, que funciona em várias plataformas e pode ser usado para monitorar redes TCP/IP de pequeno e médio portes e detectar uma grande variedade de tráfego de rede suspeito e ataques explícitos (Beale, 2004).

Ele é *sniffer* que tem como característica principal a capacidade de inspecionar o cabeçalho e área de dados de um pacote IP e comparar as informações com um banco de dados de regras que são criadas e mantidas por uma comunidade de usuários do Snort. Ao detectar um padrão de uma assinatura de ataque em um pacote, o Snort gera um alerta em um arquivo de *log*, que posteriormente analisado pelo administrador da rede, lhe dá condições de decidir sobre ações de proteção que poderão ser tomadas.

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.

Ao citar nomes de diretórios e de arquivos de configuração, deve ser considerada a instalação padrão do Snort no GNU/Linux Debian, utilizado como base para esse trabalho.

2.1 Arquitetura do Snort

A implementação do Snort, segundo uma arquitetura modular, focaliza a otimização do desempenho na coleta e análise de pacotes, que com o conjunto de regras correto para um dado *exploit* (ataque que explora as vulnerabilidades) aproxima-se do tempo-real (Cox 2004) . Oferece o conceito de plug-ins, que são seus subsistemas de:

Pré-processamento: Decodifica o pacote;

Detecção: Detecta se o pacote se encaixa nas assinaturas de regras;

Saída: Registra o alerta.

Primeiro, o decodificar de pacotes interpreta qual é conjunto de tráfego. Esse tráfego é decodificado e analisado por todo plugin pré-processador definido nas regras. Em seguida, o engine de deteção recebe o tráfego, aplica uma estrutura de regras e pode também aplicar plug-ins de detecção no tráfego excessivo para procurar outras assinaturas. Por fim, o fluxo de dados é enviado para o estágio de saída onde existem diferentes opções de registro e alerta, e ainda, pode ser enviados para plug-ins de saída.

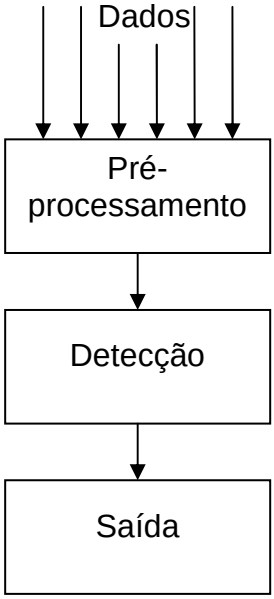

Figura 2.1: Arquitetura do Snort

2.1.1 Pré-processamento

Assim que o pacote IP é capturado na rede, ele é enviado para os pré-processadores. Os pré-processadores são responsáveis por remontar os pacotes – em caso de fragmentação, ver possíveis codificações como Unicode, entre outras tarefas. O Snort possui pré-processadores para diversos tipos de ataque, como Telnet, IIS, fragmentação de pacotes, http-inspect, smtp, etc.

Os pré-processadores podem ser ativados e configurados individualmente. Para aumentar o desempenho do Snort, recomenda-se que se ativem apenas os pré-processadores correspondentes aos serviços disponibilizados na rede.

2.1.2 Detecção

Após o encaminhamento do pacote pelos pré-processadores para o sistema de detecção, o Snort compara o pacote (já remontado e “legível”) com a base de assinaturas de ataques e define qual ação será tomada.

Caso seja detectada uma assinatura existente em alguma regra, o pacote será enviado para o sistema de saída.

2.1.3 Saída

Os plugins de saída são ferramentas que podem ser utilizadas para gerar alertas, logs ou para tomar algumas medidas de imediato. Os plugins podem enviar alertas em email, popups, gravar em arquivos de log (ação padrão), gravar em um banco de dados, entre outros.

2.2 Definição de regras

A Criação de Regras no Snort obedece a um formato ou modelo pré-estabelecido que será enviado para uma espécie de analisador léxico e gerará instruções ao programa em execução (Roesch 2009).

Na instalação padrão do Snort, os programas vêm acompanhado de vários arquivos de regras que se encontram no diretório “rules”. Estes arquivos possuem regras para diversos tipos de ataques, que podem ser adaptadas para a rede a ser protegida.

2.2.1 Variáveis

O primeiro passo para a criação de regras é a definição das variáveis de rede. Geralmente são duas variáveis que são mais usadas por analistas que manuseiam o Snort:

EXTERNAL_NET -> Refere-se a rede externa, diferente da rede a ser protegida. Isso inclui a Internet ou outras redes quaisquer que não a rede a ser protegida.

HOME_NET -> É a Rede a ser protegida.

Essas variáveis são setadas durante a instalação do Snort, ou diretamente em seu arquivo de configuração (snort.conf). Podemos definir essas variáveis como visto na figura 2.2.

```
var HOME_NET 192.168.0.1/0
var EXTERNAL_NET !$HOME_NET
```

Figura 2.2: Variáveis de rede do Snort

Acima, nós definimos o valor em uma classe de IP para HOME_NET e em seguida dizemos que tudo que for diferente dessa classe será considerado como EXTERNAL_NET. Podemos definir múltiplas redes, como apresentado na figura 2.3.

```
var HOME_NET [192.168.40.0/24,192.168.41.0/24,10.14.0.0/16]
var EXTERNAL_NET !$HOME_NET
```

Figura 2.3: Definição de múltiplas redes no Snort.

Também podem ser definidas variáveis de usuário:

var NOME_DA_VARIAVEL VALOR

Onde NOME_DA_VARIAVEL e VALOR são definidos pelo usuário. Um exemplo é apresentado na figura 2.4. No exemplo, atribuímos o valor 80 a uma variável de nome SHELLCODE_PORTS. Assim, onde aparecer a string $SHELLCODE_PORTS, o Snort entenderá como sendo “80”.

```
var SHELLCODE_PORTS 80
```

Figura 2.4: Variável de usuário

Caso a rede onde o Snort esteja configurado possua poucos serviços ou ainda poucos recursos computacionais, ao invés de setar uma classe de IP na variável HOME_NET e definir EXTERNAL_NET como sendo IPs diferentes dos pertencentes a classe, é possível utilizar para ambos o valor “any”, assim o Snort irá detectar ataques oriundos de qualquer máquina, inclusive de máquinas consideradas “confiáveis”.

2.2.2 Sintaxe das regras

As regras no Snort também obedecem a seguinte sintaxe descrita na figura 2.5.

```
<tipo_de_alerta> <protocolo> <rede_origem> <porta_origem>
     -> <rede_destino> <porta_destino>
     (Cabeçalho da Regra; <opções>; sid:X;...);
```

Figura 2.5: sintaxe básica para definição de regras do Snort.

Um exemplo bem simples pode ser visto na figura 2.6. Nesse exemplo, o tipo de alerta é “alert”, o protocolo é o TCP, a porta de origem é qualquer, a rede de destino é 192.168.1.0/24 e a porta de destino é 111.

```
alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86
a5|"; msg:"mountd access";sid:11)
```

Figura 2.6: Exemplo de definição de regras do Snort

A partir deste modelo, é possível construir regras poderosas para manusear o Snort. As possibilidades de manuseio dos campos de regras são:

Para <tipo_de_alerta>, nós temos:

alert – Gera um alerta usando um método selecionado e então loga o pacote;

log – loga o pacote;

pass – ignora o pacote;

activate – alerta e então ativa outra regra dinâmica;

dynamic – permanece inativa até ser ativado por uma regra *activate*, então atua como uma regra de log.

Em cima das opções acima, é possível interagir até mesmo na criação de regras dinâmicas. Isso é útil para detectar ataques que envolvem *worms* ou outras ferramentas de automação de ataques.

Para <protocolo>, nós temos: *tcp*, *udp* e *icmp*.

Para <opções>, temos quatro categorias principais:

meta-data – Essas opções provêem informação sobre a regra, mas não tem qualquer efeito durante a detecção;

payload – Todas essas opções procuram por dados dentro do payload do pacote e podem ser inter-relacionadas;

non-payload – Essas opções procuram por dados fora do payload;

post-detection – Essas opções são regras específicas que acontecem após uma regra ter sido detectada.

A partir de cada categoria, são definidas as opções propriamente ditas que são manuseadas pelo Snort. Para o escopo deste trabalho, veremos apenas algumas opções, uma lista completa pode ser obtida a partir do manual do Snort (Roesch 2009).

2.2.2.1 Opções de Regra de Meta-Data

msg – a opção *msg* informa a ferramenta de log e alerta qual a mensagem que deve ser imprimida quando o alerta for dado.

Formato:

msg ”<mensagem texto>”;

reference – a opção *reference* permite as regras incluírem referências externas ao ataque. Essas referências possuem maiores informações sobre o ataque. Atualmente o Snort suporta apenas alguns sistemas específicos com URLs únicas. Abaixo uma lista deles:

Sistema	**PrefixoURL**
bugtraq	http://www.securityfocus.com/bid/
cve	http://cve.mitre.org/cgi-bin/cvename.cgi?name=
nessus	http://cgi.nessus.org/plugins/dump.php3?id=
arachnids	(inativo)http://www.whitehats.com/info/IDS
Mcafee	http://vil.nai.com/vil/dispVirus.asp?virus_k=

Tabela 2-1

Formato:

reference: <id system>,<id>; [reference: <id system>,<id>;]

Um exemplo segue na figura 2.7

```
alert tcp any any -> any 7070 (msg:"IDS411/dos-realaudio"; \
flags:AP; content:"|fff4 fffd 06|"; reference:arachnids, \
IDS411;) alert tcp any any -> any 21 \
(msg:"IDS287/ftp-wuftp260-venglin-linux"; \
flags:AP; content:"|31c031db 31c9b046 cd80 31c031db|"; \
reference:arachnids,IDS287; reference:bugtraq,1387; \
reference:cve,CAN-2000-1574;)
```

Figura 2.7: Exemplo de definição de *reference*

sid – a opção sid é usada unicamente para identificar regras do Snort. Esta informação permite aos plugins que interagem com o Snort identificar regras facilmente. Deve ser usada com a opção “rev”, que veremos mais abaixo.

Formato:

sid <snort_rule_id>

Deve-se atentar a numeração de id que precisa ser obedecida para se formar um padrão. Ela é a seguinte:

Menor que 100 – Reservada para uso Futuro;

Entre 100 e 1000000 – Regras incluídas com a distribuição do Snort;

Maior que 1000000 – Regras usadas localmente;

rev – esta opção é usada unicamente para identificar revisões em regras do Snort. Revisões permitem demonstrar uma melhora na escrita das assinaturas de ataques. Esta opção deve ser usada em conjunto com a opção *sid*.

Formato:

rev <inteiro da revisão>

classtype – permite orientar a categoria de ataque que se está sendo detectado. Trabalha em cima de uma tabela com nome, descrição e prioridade. O usuário pode então especificar qual a prioridade que determinado tipo de ataque tem ao ser detectado.

Formato:

classtype: <nome da classe>;

As classificações de regras são definidas no arquivo classification.config. Este arquivo utiliza a seguinte sintaxe:

config classification: <nome da classe>,<descrição>,<prioridade>

Um exemplo de regra contendo esta opção segue abaixo na figura 2.8, Podemos então notar que a classtype descrita se chama “attempted-recon”, que possui entrada em *classification.config*:

```
alert tcp any any -> any 25 (msg:"SMTP expn \
root"; flags:A+; content:"expn root";nocase; \
classtype:attempted-recon;) config classification: \
attempted-recon,Attempted Information Leak,2
```

Figura 2.8: Exemplo de regra com a opção classtype.

2.3 Módulos de saída

Módulos de saída permitem ao Snort flexibilidade na formatação e apresentação de saída para o usuário. Os módulos são executados quando um registro de detecção de intrusão ocorrer.

Plugins de saída podem ser especificados no arquivo de configuração do Snort. Quando vários plugins do mesmo tipo (log, alerta) são especificados eles são disparados em seqüência, quando ocorre um evento. Assim como outros registros de log, o Snort armazena por padrão, o diretório /var/log/snort para armazenar os registros de eventos.

Os módulos de saída são especificados no arquivo snort.conf e possuem a sintaxe apresentada na figura 2.9.

```
output <name>: <opções>
output alert_syslog: LOG_AUTH LOG_ALERT
```

Figura 2.9: Formato do carregamento do módulos de saída.

Os módulos podem ser os seguintes: *alert_syslog*, *alert_fast*, *alert_full*, *alert_smb*, *alert_unixsock*, *log_tcpdump*, *database*, *csv*, *unified*, *log_null*.

Para o escopo deste estudo, podemos destacar os módulos de saída alert_syslog que é o padrão do Snort e o database que permite especificar um servidor de banco de dados onde o é possível registrar os alertas.

2.3.1 Módulo de saída alert_syslog

Como já mencionado, este módulo de saída é o padrão do IDS, usa o *syslog* do sistema operacional e permite ao usuário especificar a prioridade do alerta e o tipo de log. A Saída do syslog de um alerta pode ser verificado na figura 2.10.

```
[**] [119:2:1] (http_inspect) DOUBLE DECODING ATTACK [**]
[Priority: 3]
11/30-09:06:53.989012 143.54.13.194:12315 -> 143.54.13.147:80
TCP TTL:128 TOS:0x0 ID:30563 IpLen:20 DgmLen:425 DF
***AP*** Seq: 0xAEC0FFE7  Ack: 0xB1A2EC94  Win: 0xFFFF  TcpLen: 20
```

Figura 2.10: Um alerta gerado pelo módulo de saída alert_syslog.

Numa análise superficial, é possível perceber o IP de origem e destino do pacote, o tipo de alerta. Essas informações são importantes para o administrador saber que a tentativa ocorreu, mas uma análise estatística sobre os tipos de alertas ou ainda sobre períodos em que ocorreram, poderia demandar um tempo diretamente proporcional ao tamanho do arquivo de log (Beale 2004).

Além disso, informações sobre o ataque específico tais como formas de se proteger não são citadas pelo Snort, que na verdade tem o objetivo apenas de informar ao administrador da rede que a tentativa de intrusão ocorreu.

Esse na realidade é o principal motivador do estudo desse trabalho, ou seja, um estudo de caso sobre uma ferramenta que pode gerar estatísticas e informações sobre os ataques, bem como servir de orientação sobre as ações que o administrador de rede irá tomar para proteger a sua rede.

2.3.2 Módulo de saída database

Esse módulo permite ao Snort trabalhar com uma variedade de servidores de bancos de dados. É preciso definir detalhes da conexão como o endereço do host, nome da base de dados do snort, usuário e senha com autorização de manipular os registros.

3 FERRAMENTAS DE APOIO

Nesse capítulo iremos apresentar as ferramentas de apoio utilizadas para a instalação e configuração do IDS: O host, o sistema operacional, o servidor de banco de dados, e a ferramenta de geração de estatísticas.

3.1 Host

O Snort é um sniffer de rede que coloca a interface de rede em modo promíscuo, sendo possível dessa forma, utilizá-lo para capturar os pacotes não só destinados à própria máquina, como também destinados a toda a rede. Como base nessa informação podemos configurar um host como IDS da rede de duas formas: 1 – diretamente em um gateway da rede, configurado para capturar os pacotes que passam por ele, ou 2 – em uma máquina ligada a um HUB, capturando todo o tráfego da rede que passa pelo HUB (Cox, 2004). Nesse último cenário, o host que executa o Snort entra sem alterar nenhuma configuração da rede, e pode ser usado como uma máquina exclusiva para esse fim – Figura 3.1.

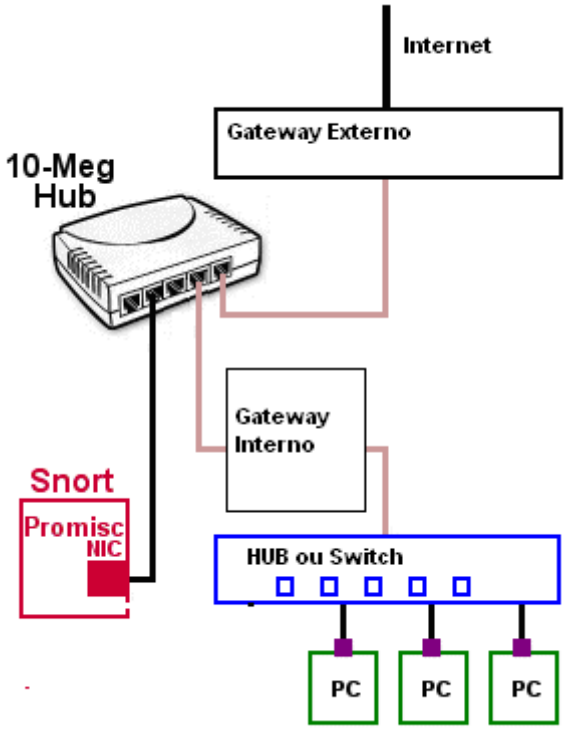

Figura 3.1: Ambiente para o IDS com Snort.

Como o objetivo do trabalho é definir um caso prático, para o IDS, utilizaremos apenas a sua instalação e configuração, dessa forma é irrelevante o hardware e o cenário da rede onde o host se encontra, tanto que para os teste, utilizamos uma máquina virtual com o *Microsoft Virtual PC* e um PC com *Windows XP SP3*. Essa máquina virtual estava configurada com 128 MB de memória, disco rígido de 2GB, e sua CPU com apenas um núcleo e sem virtualização, ou seja, poucos recursos. Esse cenário é apresentado na figura 3.2. Para um caso prático, vários trabalhos sugerem que se utilize uma máquina real com mais recursos de memória e CPU, principalmente se o tráfego da rede que se deseja captura for considerável (Bastos, 2004).

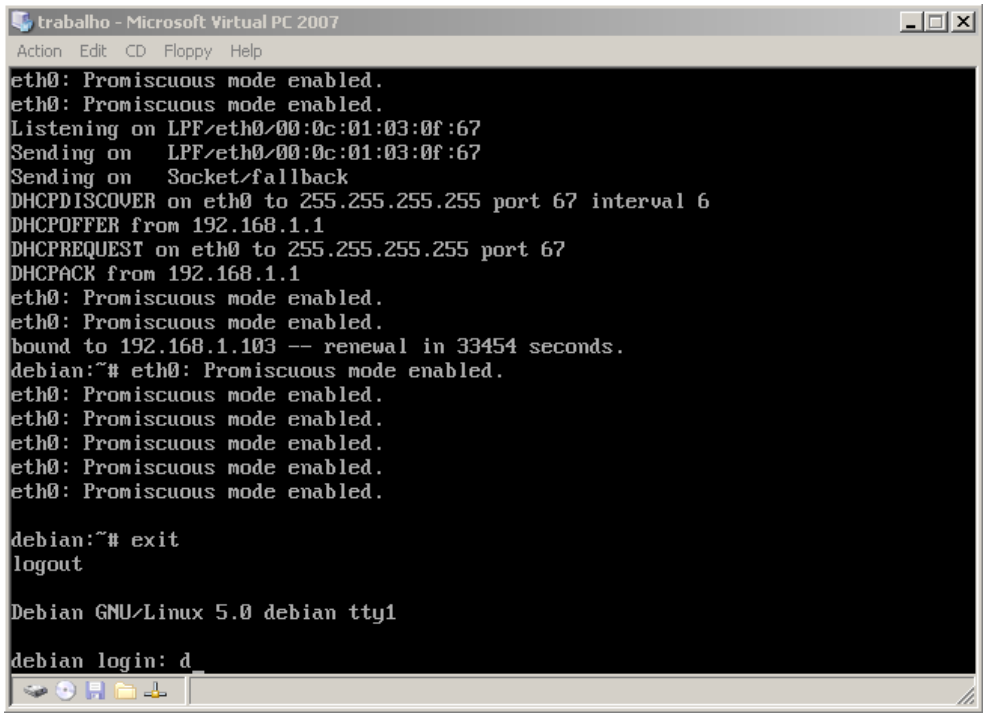

Figura 3.2: Máquina Virtual com o Host do IDS.

3.2 Sistema Operacional

Nosso estudo de caso será desenvolvido a partir de Sistema Operacional GNU/Linux Debian 5.0, mas também poderia ser utilizado um sistema baseado em Windows, visto que o Snort e o MySQL (as principais ferramentas necessárias) possuem versões para estas duas plataformas.

A escolha do Debian se deve em função de a obtenção e instalação de novos programas ser de relativa facilidade, visto que o está disponível um gerenciador de pacotes de instalação de programas, o *apt*.

3.2.1 Debian dpkg e apt

No Debian os softwares são instalados através de pacotes *.deb*. Um pacote de software é um arquivo contendo todos os arquivos (binários, de configurações, documentação entre outros tipos) necessários para a instalação de um software

juntamente com todas as informações necessárias para a configuração e manutenção pelo gerenciador de pacotes (Morimoto 2007).

Um gerenciador de pacotes é um software cuja função é facilitar a instalação, remoção, atualização, configuração, manutenção e até mesmo o empacotamento de outros softwares. O gerenciador de pacotes do Debian é o *dpkg*. O dpkg possui opções para instalação de pacotes mal configurados, mas não resolve a dependência de pacotes automaticamente.

O apt é um gerenciador de pacotes disponível na distribuição que permite ao administrador do sistema obter binários e fontes de aplicativos e utilitários a partir de várias meios: cd's, dvd's, HTTP, FTP, NFS, ou ainda um diretório local. Diferentemente do dpkg, utilizando o apt, é possível resolver dependências de outros pacotes automaticamente.

O apt baseia-se no conceito de repositório, onde os pacotes de software são armazenado e podem ser adquiridos. Na internet existem inúmeros repositórios públicos, e sua configuração pode ser feita no arquivo */etc/apt/sources.list*.

Existem várias ferramentas que podem manipular a fonte dos pacotes ou a obtenção, entra elas podemos destacar o *apt-get* e o *apt-cache*.

3.2.1.1 Configuração do apt

Como já mencionado, a configuração do apt é feita no arquivo /etc/apt/sources.list. Um exemplo desse arquivo é apresentado na figura 3.3.

```
deb uri distribution [component1] [component2] [...]
```

Figura 3.3: Configuração do repositório apt

Nessa linha, podemos destacar os seguintes itens:

deb: é o tipo de pacote, que pode ser o binário do aplicativo (deb) ou ainda o seu código fonte (deb-src).

uri: especifica a base da distribuição do Debian, da qual o APT irá encontrar as informações que necessita. Essa fonte pode ser do tipo *file*, *cdrom*, *http*, *ftp*, *copy*, *rsh* ou *ssh*.

component: podemos especificar a versão do debian (*stable*, *testing* ou *unstable*), stable é a versão atual estável do sistema; testing é a próxima versão que ao final de testes será lançada como stable; unstable é uma versão que recebe pacotes de softwares com versões novas que ainda não foram testadas pelos desenvolvedores do Debian.

Um exemplo prático da especificação da configuração de um repositório no arquivo /etc/apt/sources.list está apresentado na figura 3.4.

```
deb http://debian.pop-sc.rnp.br/debian stable main contrib non-free
```

Figura 3.4: Exemplo de linha do arquivo sources.list.

3.2.1.2 apt-get

O *apt-get* é o utilitário usado para gerenciar a instalação de novos pacotes, remoção de pacotes antigos e atualização das bases a partir dos repositórios configurados.

Para o escopo desse trabalho, vale apresentar dois modos de utilização do apt-get:

apt-get update: atualiza as informações locais sobre os repositórios configurados no arquivo sources.list.

apt-get install pacote: baixa um pacote da fonte disponível, descompacta e instala do pacote.

3.2.1.3 apt-cache

O apt-cache é o utilitário que efetua uma busca nos índices atualizados pelo apg-get update. Dessa forma é possível procurar por um determinado pacote, mesmo sem conhecer o seu nome.

3.3 Snort-MySQL

Os registros de alertas gerados pelo Snort permitem ao administrador da rede verificar as tentativas de intrusão e no máximo, tomar conhecimento que elas estão acontecendo. Entretanto, analisar um arquivo de texto puro, sem padrão de formatação e com vários códigos (dos alertas e pré-processadores), não é uma tarefa fácil para um humano.

O Snort na sua instalação padrão não possui nenhuma ferramenta que gere estatísticas e resumos dos alertas, o que facilitaria o trabalho do administrador da rede, podendo assim concentrar seus esforços nas vulnerabilidades do sistema e não no “garimpo” de informações de log.

Pensando nisso, várias ferramentas foram desenvolvidas para gerar estatísticas a partir do registro de alertas do IDS. Entre elas podemos destacar o *Snort-MySQL*, e o *BASE*.

O Snort-MySQL é um plugin do IDS que permite gerar os alertas do Snort em um banco de dados do servidor MySQL. Para isto o Snort precisa ser configurado com o módulo de saída *database* permite que se faça o registro dos alertas em um servidor de banco de dados conforme a seção 2.3.2.

3.3.1 Estrutura do banco de dados de alertas

O banco de dados utilizado pelo Snort-MySQL possui a estrutura apresentada na figura 3.5.

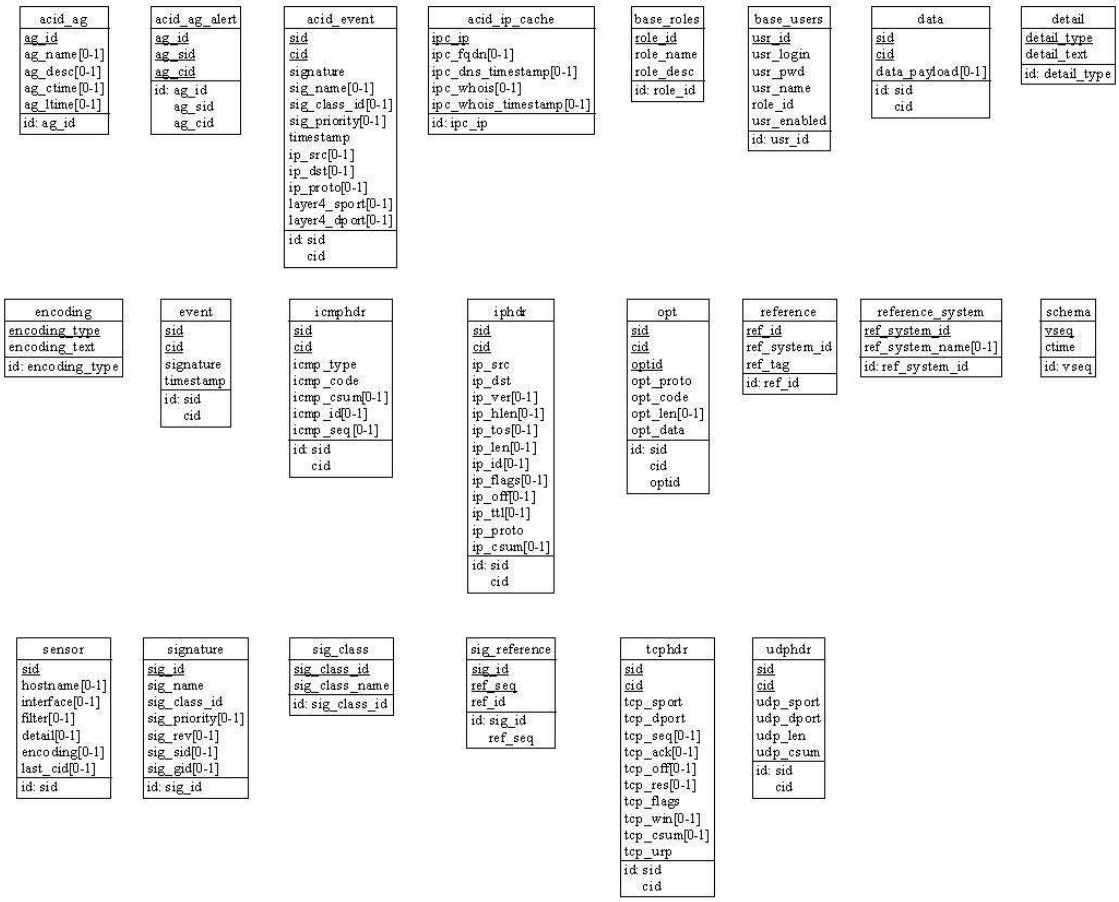

Figura 3.5: Diagrama ER da base de dados do Snort-MySQL.

O script em SQL que gera as tabelas está apresentado no Anexo B.

3.4 BASE

O BASE - *Basic Analysis and Security Engine* é uma ferramenta escrita em PHP que gera estatísticas a partir da base de dados do snort armazenada no servidor de banco de dados do MySQL. Como usa scripts PHP, é necessário ter o PHP e um servidor web instalado. Para o servidor Web, utilizaremos o Apache 2.2. Na figura 3.6 é apresentada uma tela do BASE.

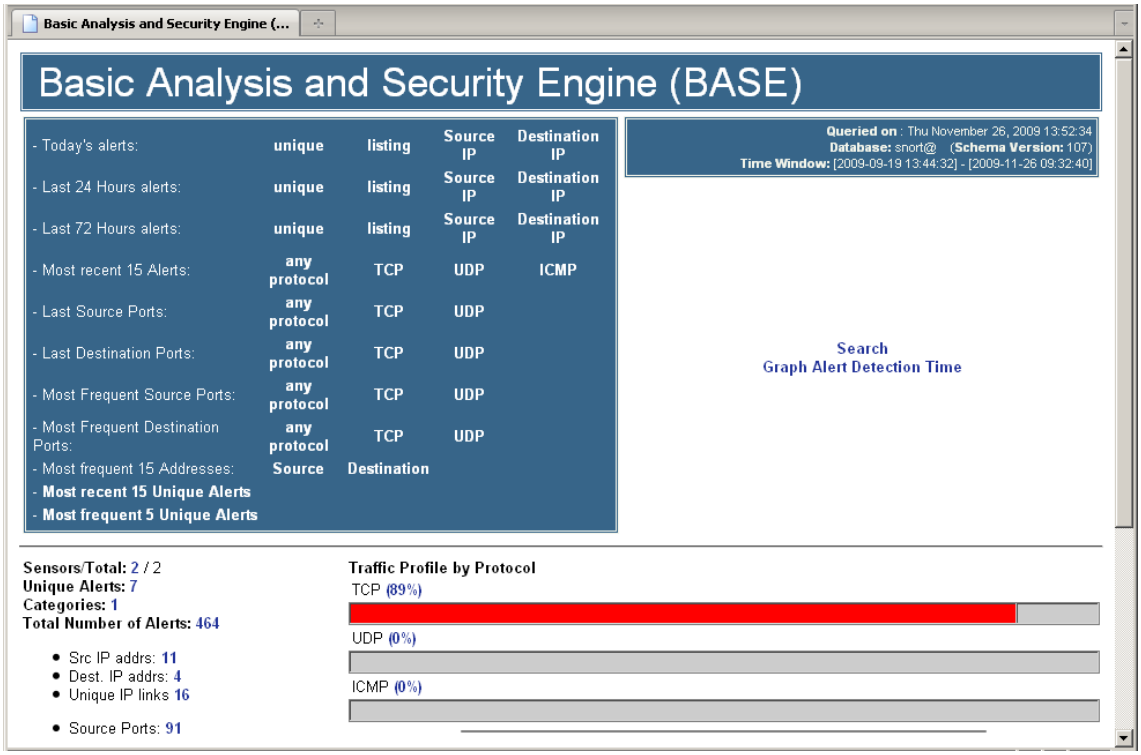

Figura 3.6: BASE

O BASE, além de gerar estatísticas da base de dados do Snort armazenada no servidor MySQL, também mostra o pacote específico que foi enviado ao host, apresentando assim as técnicas utilizadas pelo atacante. Um exemplo dessa informação é apresentado na figura 3.7.

Além disso, é possível obter maiores informações a respeito do tipo de ataque, utilizando as informações de referência do Snort, geradas pelo sistema de alerta, conforme a tabela 1.1. Um exemplo da obtenção dessas informações é apresentado na figura 3.8.

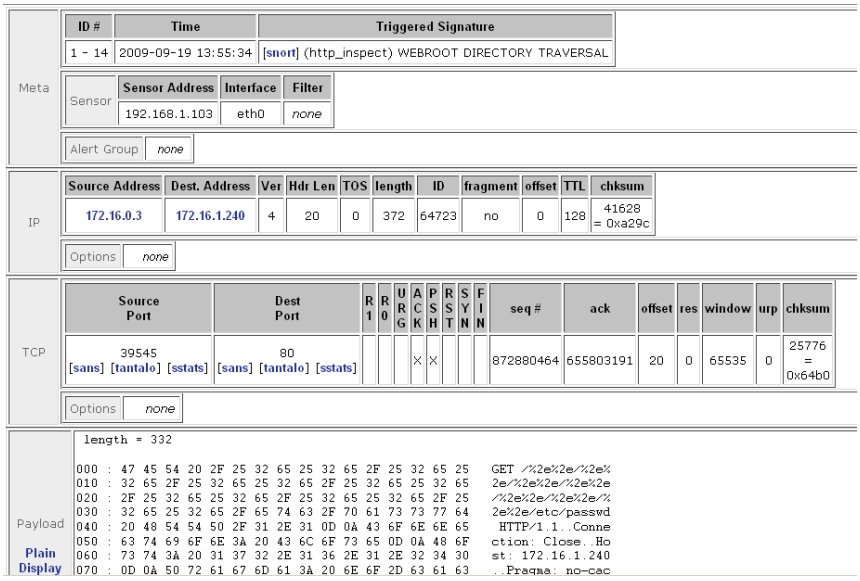

Figura 3.7: Pacote capturado apresentado no base.

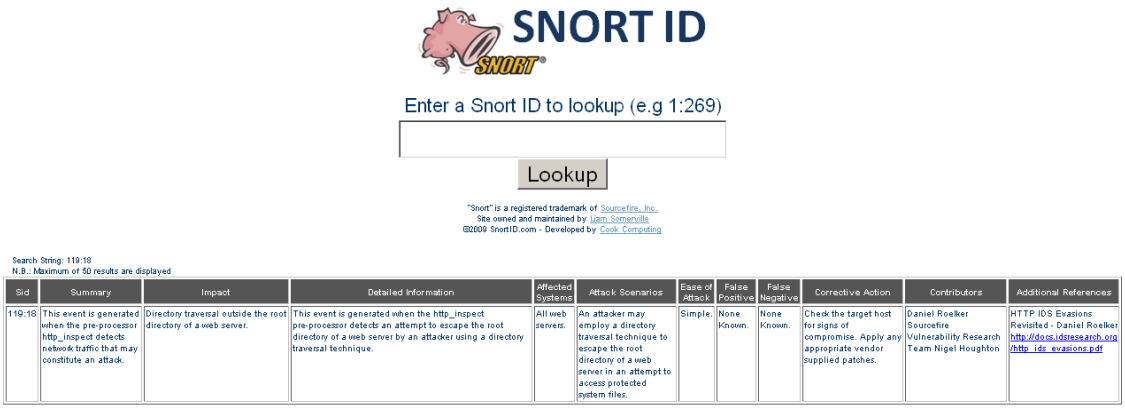

Figura 3.8: Informações do SNORT ID sobre o tipo de ataque.

3.5 nmap

O *nmap* é port scanner utilizado para fazer varredura de portas e detecção de finger prints em sistemas (Melo, 2004) . Basicamente, o nmap dispara pedidos de abertura de conexão (SYN) e aguarda a sua confirmação (SYN+ACK). Ao receber a confirmação, o nmap apresenta a porta descoberta ao usuário. No trabalho, esse software foi utilizado para gerar alertas, visto que o Snort possui a capacidade de detectar esse tipo de ataque. O *zenmap* – uma interface para o nmap é apresentado na figura 3.9, nessa figura estão listadas as portas descobertas em uma varredura. Na figura 3.10, é apresentado o nmap em ação, com o sniffer wireshark sendo executado no mesmo host do nmap, é possível observar os pacotes disparados pelo nmap com os pedidos de abertura de conexão, caracterizando assim um ataque de *port scanning*.

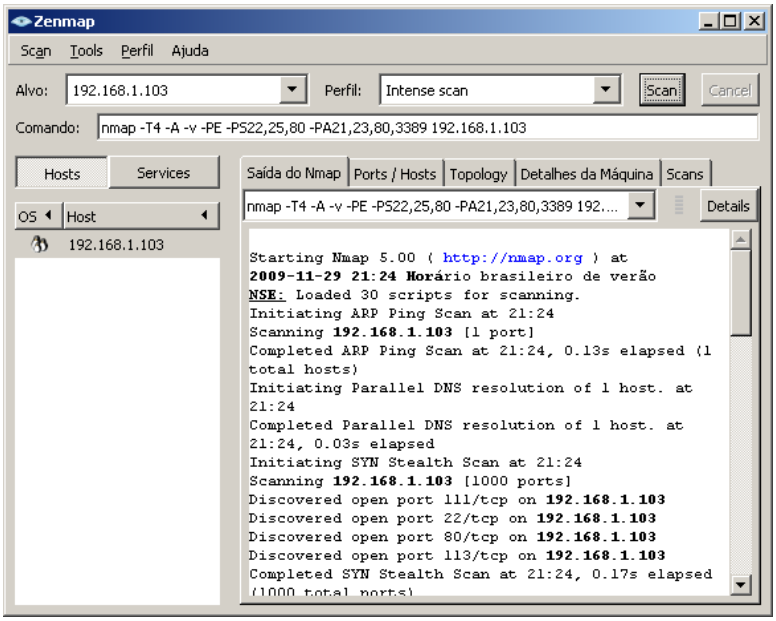

Figura 3.9: Zemap – uma interface para o nmap.

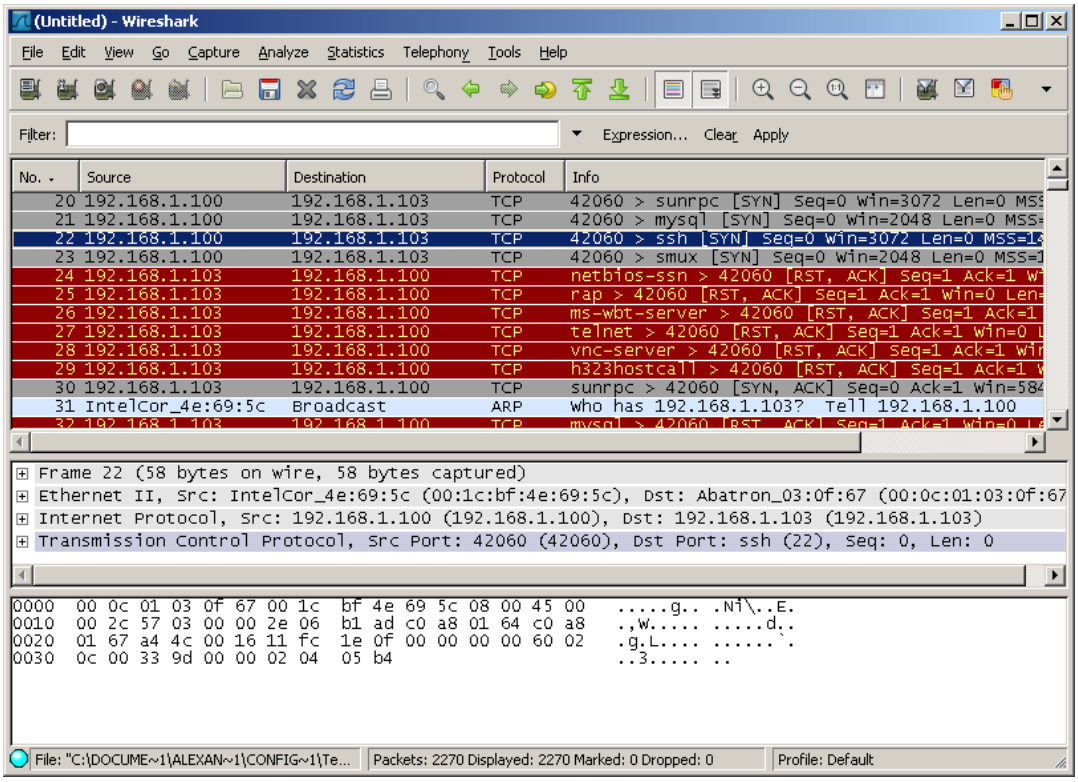

Figura 3.10: Wireshark capturando as mensagens enviadas pelo nmap ao IDS.

3.6 Nessus

O *Nessus*, assim como o nmap é um port-scanner, que também pode verificar algumas falhas ou vulnerabilidades de segurança (Melo, 2004). Ele é composto por um cliente e um servidor, sendo que o scan propriamente dito é feito pelo servidor.

3.6.1 Funcionamento

O nessusd (servidor Nessus) faz um port scanning no computador alvo, depois disso vários scripts (escritos em NASL, *Nessus Attack Scripting Language*) conectam-se a cada porta aberta para verificar problemas de segurança.

No cliente, é possível definir quais são as vulnerabilidades que serão testadas, estabelecendo uma política para o scanner. Esta política pode customizar a verredura para determinado host, por exemplo, se um servidor de banco de dados está instalado na máquina, é possível fornecer senhas de usuário para que o Nessus explore as vulnerabilidades específicas desse servidor.

3.6.2 Relatório

O Nessus exibe um relatório das vulnerabilidades do host , bem como sugestões de como corrigir as vulnerabilidades.

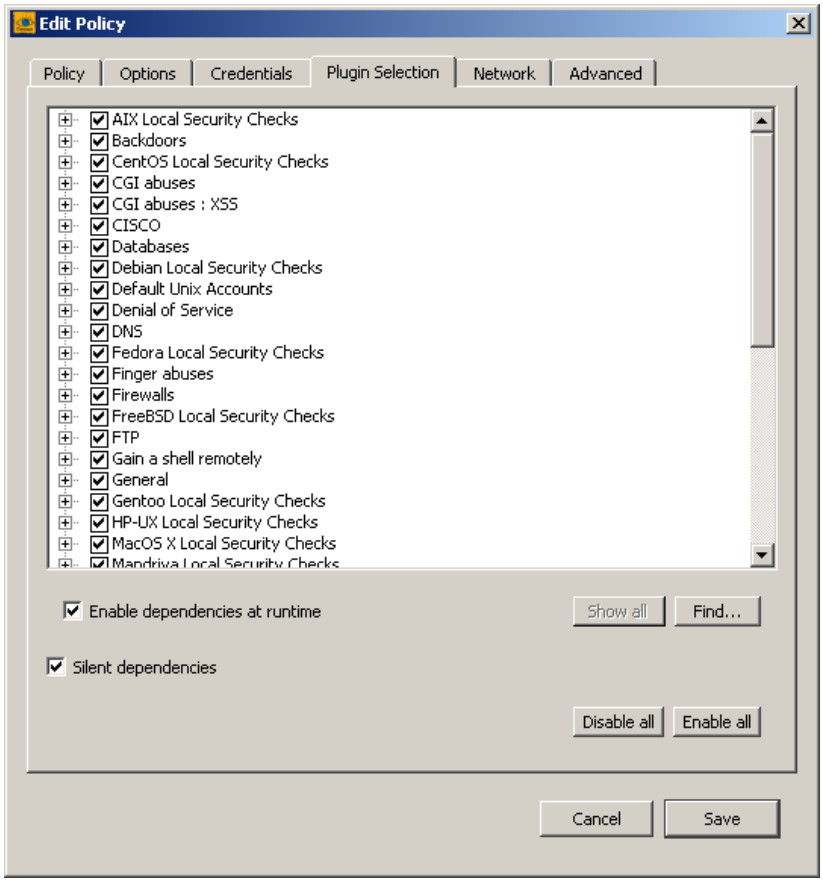

Figura 3.11: Editor de políticas do Nessus.

4 INSTALAÇÃO E CONFIGURAÇÃO DO IDS

Após a instalação do sistema operacional host que irá receber o IDS, é necessário instalar e configurar as ferramentas que farão parte do estudo. Para isso, foi criado um script que automatiza a instalação. Esse script é descrito neste capítulo.

4.1 Instalação e Configuração

A instalação do IDS será realizada então, para os nossos testes em uma máquina virtual do Virtual PC com o Debian 5 instalado. Supondo então que o sistema operacional já está ok, seguiremos os seguintes passos:

1 – Instalação do Servidor Web apache, do Servidor de Banco de Dados MySQL e do php. Esses podem ser instalados com o apt-get com os comandos representados na figura 4.1. Nesse passo, é requerida a entrada manual da senha do servidor MySQL por parte do usuário.

```
# apt-get install mysql-server
# apt-get install apache2
# apt-get install php
```

Figura 4.1: Instalação do apache, mysql e php.

2 – Instalação do Snort-MySQL. Nesse passo é instalado o IDS com o plugin de saída para os dados de alertas serem armazenados no Servidor de Banco de Dados. Assim como no passo 1, a instalação pode ser feita com o apt-get. A instalação irá solicitar a entrada manual do endereço CIDR da rede do usuário, que será armazenado na variável HOME_NET da configuração do Snort, conforme visto na seção 2.2.1. É solicitada entrada do usuário para a utilização do banco de dados. A variável $HOME_NET é associada a uma interface de rede, que pode não ser a mesma que está definida no sistema, em função disso, o snort irá falhar na instalação e sua configuração irá ficar pendente. Como somente poderemos corrigir a instalação após a criação da base de dados, isso será realizado no passo 5.

```
# apt-get install snort-mysql
```

Figura 4.2: Instalação do snort-mysql.

Ao final da execução desse passo, é gerado um erro, pois a instalação do Snort não está terminada. Trataremos desse erro nos passos subseqüentes.

3 – Instalação do BASE. Nesse passo são solicitadas entradas de qual servidor de banco de dados será utilizado, além das senhas para o servidor de banco de dados e da aplicação.

```
# apt-get install acidbase
```

Figura 4.3: Instalação do BASE.

Como os scripts do BASE são colocados em um diretório que não é acessível pelo Servidor Web, é necessário configurar o arquivo do site default do apache, a listagem de configuração do arquivo do site está no Anexo A.

4 – Durante a instalação do BASE, o usuário é perguntado sobre a criação do banco de dados para o Snort no Servidor de Banco de Dados, que não foi configurado ainda. O snort-mysql fornece um script que cria o banco de dados da figura 3.4 no servidor. Uma listagem desse script é fornecida no Anexo B.

O comando apresentado na figura 4.4 instala a base de dados do Snort no MySQL. Nesse comando, *password* é a senha que será utilizada para o usuário *snort* acessar o banco de dados *snort*.

```
# zcat /usr/share/doc/snort-mysql create_mysql.gz | mysql -u snort -D
snort -ppassword
```

Figura 4.4: Instalação da base de dados

5 – Feito isso, é necessário reconfigurar a instalação do Snort, que ficou pendente no passo 2. Os comandos são apresentados na figura 4.5.

```
# dpkg --configure --pending
# dpkg-reconfigure snort-mysql
```

Figura 4.5: Reconfigurando a instalação pendente.

Nesse ponto a instalação irá solicitar a forma de inicialização do Snort, a interface de rede que irá trabalhar em modo promíscuo para a captura dos pacotes da rede que serão enviados ao Snort, a variável HOME_NET, configurações adicionais, envio de alertas por email, usuário, senha e banco de dados da base do MySQL.

5 – Feito isso, basta terminar de configurar o BASE. Isto é feito diretamente no navegador através do endereço http://ip_ids/acidbase. Onde *ip_ids* é o endereço ip do host onde o ids foi instalado. A figura 4.6 mostra a tela de configuração do BASE. Nessa tela, basta selecionar o link *Setup* e em seguida pressionar o botão *Create BASE AG*, para terminar a configuração. Esse passo consiste basicamente da criação das tabelas do banco de dados que são usadas pelo BASE. O BASE está então configurado e instalado.

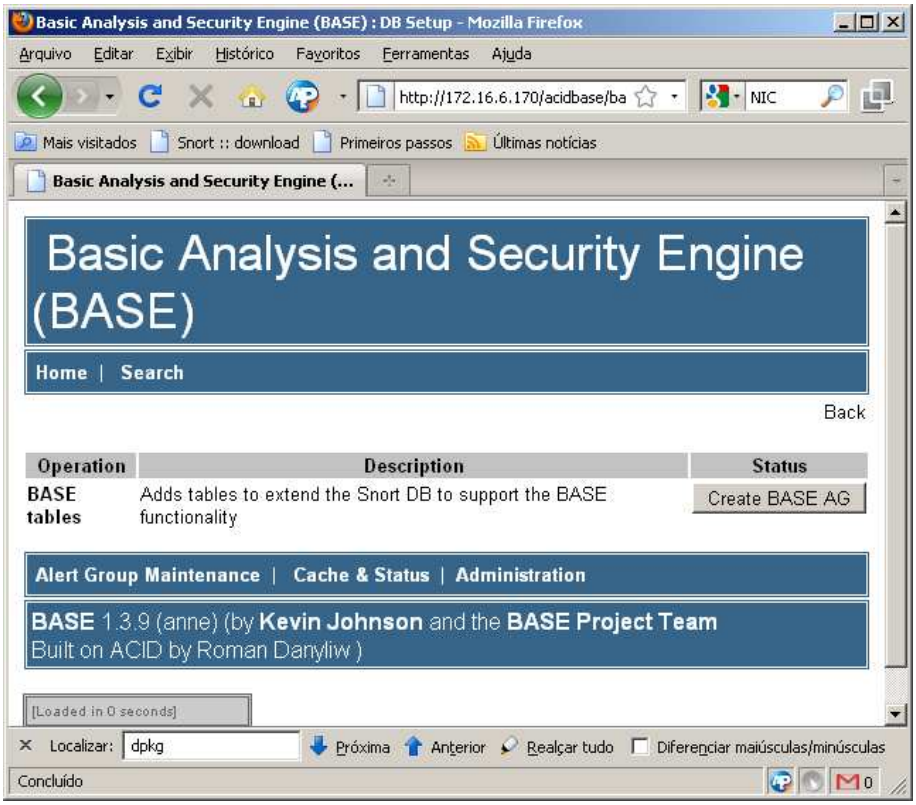

Figura 4.6: Tela de configuração do BASE.

4.2 Testando a instalação

Para testar a instalação, iremos utilizar o nmap para disparar ataques de *port scanning* no host onde o IDS está instalado. O que se espera é que o BASE apresente os resultados assim que o que o Snort detectar o ataque. A figura 4.7 mostra o nmap em ação, e a figura 4.8 apresenta o relatório do BASE com o registro dos ataques.

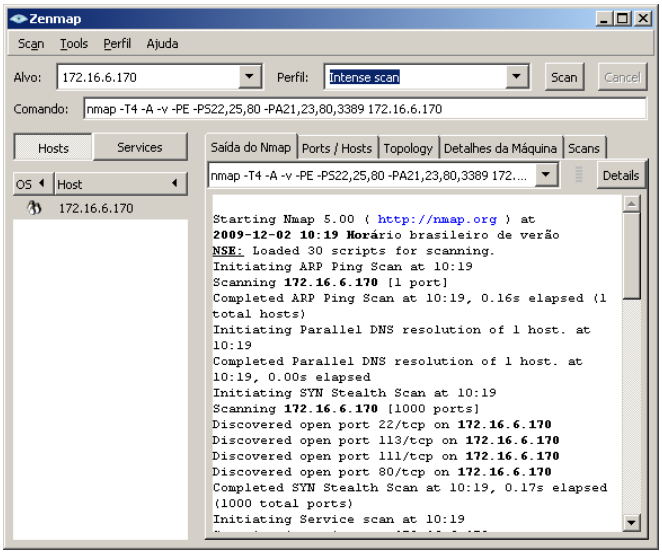

Figura 4.7: Port scanning no IDS.

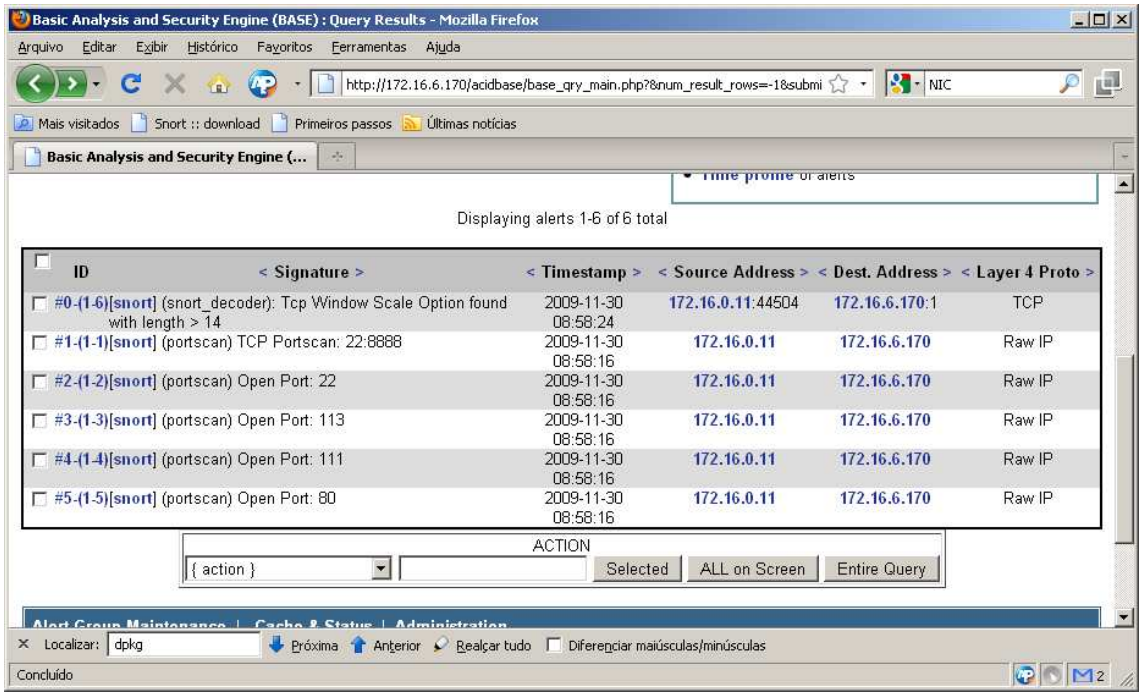

Figura 4.8: Registro do Port Scanning do nmap no BASE.

Os testes também foram realizados com o Nessus, e os resultados apresentados na figura 4.9.

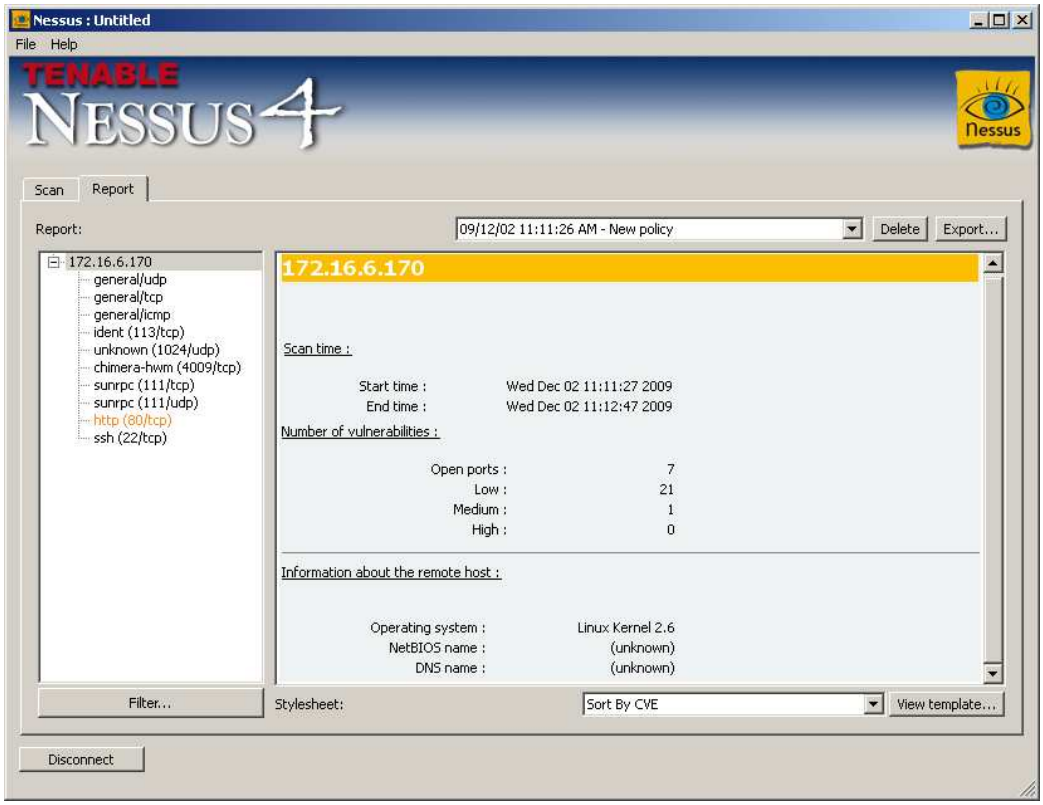

Figura 4.9: Tela de relatório do Nessus.

Da mesma forma que o port scanning do nmap foi detectado pelo Snort, os exploits do Nessus também o foram. A figura 4.10 apresenta a tela do BASE após a varredura do Nessus.

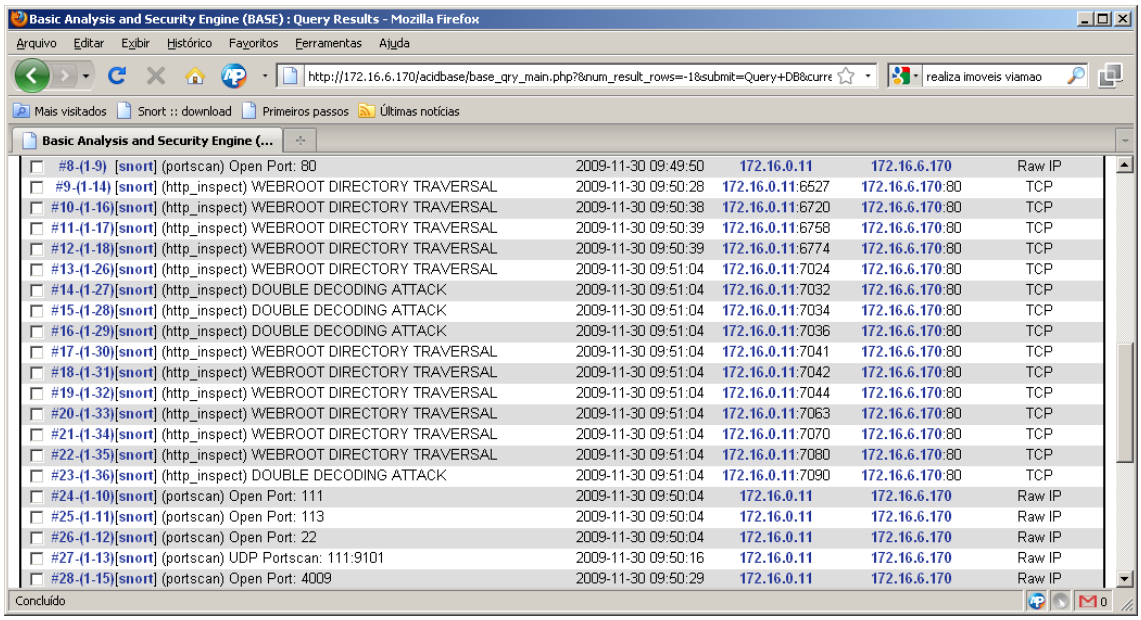

Figura 4.10: Registro da varredura do Nessus no BASE.

5 CONCLUSÃO

Este trabalho apresentou um estudo sobre a implementação prática de um IDS em um ambiente Linux utilizando o a integração de ferramentas populares como Snort, Apache, MySQL e php, bem como os scripts BASE que geram relatórios de fácil entendimento.

Esse conjunto de ferramentas fornece a um administrador de redes a possibilidade de tomar decisões em relação ao tráfego nocivo que transita na rede, de forma que possa utilizar ferramentas de defesa como firewalls, ou ainda procurar remover as vulnerabilidades por falta de atualização de softwares ou ainda por motivo de descuido.

Nesse sentido, o IDS não é uma ferramenta automática para defesa contra tentativas de invasão, ele simplesmente gera alertas para o administrador, o que foi o objetivo do estudo.

Ao longo desse estudo surgiram idéias de trabalhos futuros, como por exemplo ferramentas que geram bloqueios automáticos – por tratarem de uma solução completa para a defesa da rede. Entre essas ferramentas, podemos citar o *fail2ban* e o *guardian*, que manipulam diretamente as regras do firewall de acordo com as tentativas de invasão. O fail2ban (Jaquier, 2009) para ataques de força bruta como em um servidor *ssh* e o guardian (Stevens 2009) integrado aos alertas do snort.

REFERÊNCIAS

BEALE, Jay. **Snort 2.1 Intrusion Detection**. 2nd ed. Syngress Publishing Inc.: Rockland, 2004.

COX, Kerry. **Managing security with Snort and IDS tools**. Beijing: O'Reilly, 2004.

TANENBAUM, Andrew S.. **Redes de computadores**. Rio de Janeiro: Campus, 2003.

MELO, Sandro. **Estudo de técnicas para exploração de vulnerabilidades em redes TCP/IP**. Rio de Janeiro: Alta Books, 2004.

SHIREY, Robert W.. **Internet Security Glossary**: RFC 2828 [S.l.]: Internet Engineering Task Force, Network Working Group, 2000.

ROESCH, Martin. GREEN, Chris. **Snort users manual 2.8.5**. [S.l.]: Sourcefire Inc., October 22, 2009. Disponível em <http://www.snort.org/assets/125/snort_manual-2_8_5_1.pdf> Acesso em nov. 2009.

BASTOS, E. L.; Steffen, J.. **Análise das ferramentas de IDS SNORT e PRELUDE quanto à eficácia na detecção de ataque e na proteção quanto à evasões**. Revista Tecnologia e Tendências, v.3(1), Junho 2004, pp. 19-24.

MORIMOTO, Carlos E.. **Tutorial Completo do apt-get**. [S.l]: Guiadohardware.net, Abril de 2007. Disponível em <http://www.guiadohardware.net/tutoriais/tutorial-completo-apt-get/> Acesso em dez. 2009.

JAQUIER, Cyril. **Fail2Ban Manual 0.8**. [S.l.]: fail2ban.org, Aug, 2009. Disponível em <http://www.fail2ban.org/wiki/index.php/MANUAL_0_8> Acesso em dez. 2009.

STEVENS, Anthony. **Guardian Active Response for Snort**. [S.l]: Disponível em <http://www.chaotic.org/guardian/> Acesso em dez. 2009.

ANEXO A

Arquivo /etc/apache2/sites-available/default configurado para o BASE

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/
        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /var/www/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
        </Directory>
        ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
        <Directory "/usr/lib/cgi-bin">
                AllowOverride None
                Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
                Order allow,deny
                Allow from all
        </Directory>
        ErrorLog /var/log/apache2/error.log
        # Possible values include: debug, info, notice, warn, error, crit,
        # alert, emerg.
        LogLevel warn
        CustomLog /var/log/apache2/access.log combined
    Alias /doc/ "/usr/share/doc/"
    <Directory "/usr/share/doc/">
        Options Indexes MultiViews FollowSymLinks
        AllowOverride None
        Order deny,allow
        Deny from all
        Allow from 127.0.0.0/255.0.0.0 ::1/128
    </Directory>
<IfModule mod_alias.c>
  Alias /acidbase "/usr/share/acidbase"
</IfModule>
<DirectoryMatch /usr/share/acidbase/>
  Options +FollowSymLinks
  AllowOverride None
  order allow,deny
  allow from all
  deny from 127.0.0.0/255.0.0.0
  <IfModule mod_php4.c>
    php_flag magic_quotes_gpc Off
    php_flag track_vars On
    php_value include_path .:/usr/share/php
  </IfModule>
</DirectoryMatch>
</VirtualHost>
```

ANEXO B

Script para criação da base de dados

```
# Copyright (C) 2000-2002 Carnegie Mellon University
#
# Maintainer: Roman Danyliw <rdd@cert.org>, <roman@danyliw.com>
#
# Original Author(s): Jed Pickel <jed@pickel.net>    (2000-2001)
#                     Roman Danyliw <rdd@cert.org>
#                     Todd Schrubb <tls@cert.org>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License Version 2 as
# published by the Free Software Foundation.  You may not use, modify or
# distribute this program under any other version of the GNU General
# Public License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

CREATE TABLE `schema` ( vseq        INT      UNSIGNED NOT NULL,
                      ctime       DATETIME NOT NULL,
                      PRIMARY KEY (vseq));
INSERT INTO `schema`  (vseq, ctime) VALUES ('107', now());

CREATE TABLE event  ( sid         INT      UNSIGNED NOT NULL,
                      cid         INT      UNSIGNED NOT NULL,
                      signature   INT      UNSIGNED NOT NULL,
                      timestamp            DATETIME NOT NULL,
                      PRIMARY KEY (sid,cid),
                      INDEX       sig (signature),
                      INDEX       time (timestamp));

CREATE TABLE signature ( sig_id          INT              UNSIGNED NOT NULL
AUTO_INCREMENT,
                         sig_name     VARCHAR(255) NOT NULL,
                         sig_class_id INT          UNSIGNED NOT NULL,
                         sig_priority INT          UNSIGNED,
                         sig_rev      INT          UNSIGNED,
                         sig_sid      INT          UNSIGNED,
                         sig_gid      INT          UNSIGNED,
                         PRIMARY KEY (sig_id),
                         INDEX   sign_idx (sig_name(20)),
                         INDEX   sig_class_id_idx (sig_class_id));

CREATE TABLE sig_reference (sig_id  INT    UNSIGNED NOT NULL,
                            ref_seq INT    UNSIGNED NOT NULL,
```

```
                          ref_id  INT    UNSIGNED NOT NULL,
                          PRIMARY KEY(sig_id, ref_seq));

CREATE TABLE reference (   ref_id           INT            UNSIGNED NOT NULL
AUTO_INCREMENT,
                         ref_system_id INT         UNSIGNED NOT NULL,
                         ref_tag       TEXT NOT NULL,
                         PRIMARY KEY (ref_id));

CREATE TABLE reference_system ( ref_system_id   INT          UNSIGNED NOT NULL
AUTO_INCREMENT,
                               ref_system_name VARCHAR(20),
                               PRIMARY KEY (ref_system_id));

CREATE TABLE sig_class ( sig_class_id            INT      UNSIGNED NOT NULL
AUTO_INCREMENT,
                        sig_class_name      VARCHAR(60) NOT NULL,
                        PRIMARY KEY (sig_class_id),
                        INDEX       (sig_class_id),
                        INDEX       (sig_class_name));

# store info about the sensor supplying data
CREATE TABLE sensor ( sid         INT      UNSIGNED NOT NULL AUTO_INCREMENT,
                      hostname    TEXT,
                      interface   TEXT,
                      filter      TEXT,
                      detail      TINYINT,
                      encoding    TINYINT,
                      last_cid    INT      UNSIGNED NOT NULL,
                      PRIMARY KEY (sid));

# All of the fields of an ip header
CREATE TABLE iphdr  ( sid         INT      UNSIGNED NOT NULL,
                      cid         INT      UNSIGNED NOT NULL,
                      ip_src      INT      UNSIGNED NOT NULL,
                      ip_dst      INT      UNSIGNED NOT NULL,
                      ip_ver      TINYINT  UNSIGNED,
                      ip_hlen     TINYINT  UNSIGNED,
                      ip_tos      TINYINT  UNSIGNED,
                      ip_len      SMALLINT UNSIGNED,
                      ip_id       SMALLINT UNSIGNED,
                      ip_flags    TINYINT  UNSIGNED,
                      ip_off      SMALLINT UNSIGNED,
                      ip_ttl      TINYINT  UNSIGNED,
                      ip_proto    TINYINT  UNSIGNED NOT NULL,
                      ip_csum     SMALLINT UNSIGNED,
                      PRIMARY KEY (sid,cid),
                      INDEX ip_src (ip_src),
                      INDEX ip_dst (ip_dst));

# All of the fields of a tcp header
CREATE TABLE tcphdr(  sid         INT      UNSIGNED NOT NULL,
                      cid         INT      UNSIGNED NOT NULL,
                      tcp_sport   SMALLINT UNSIGNED NOT NULL,
                      tcp_dport   SMALLINT UNSIGNED NOT NULL,
                      tcp_seq     INT      UNSIGNED,
                      tcp_ack     INT      UNSIGNED,
                      tcp_off     TINYINT  UNSIGNED,
                      tcp_res     TINYINT  UNSIGNED,
                      tcp_flags   TINYINT  UNSIGNED NOT NULL,
                      tcp_win     SMALLINT UNSIGNED,
                      tcp_csum    SMALLINT UNSIGNED,
                      tcp_urp     SMALLINT UNSIGNED,
                      PRIMARY KEY (sid,cid),
                      INDEX       tcp_sport (tcp_sport),
                      INDEX       tcp_dport (tcp_dport),
                      INDEX       tcp_flags (tcp_flags));
```

```
# All of the fields of a udp header
CREATE TABLE udphdr(  sid         INT      UNSIGNED NOT NULL,
                      cid         INT      UNSIGNED NOT NULL,
                      udp_sport   SMALLINT UNSIGNED NOT NULL,
                      udp_dport   SMALLINT UNSIGNED NOT NULL,
                      udp_len     SMALLINT UNSIGNED,
                      udp_csum    SMALLINT UNSIGNED,
                      PRIMARY KEY (sid,cid),
                      INDEX       udp_sport (udp_sport),
                      INDEX       udp_dport (udp_dport));

# All of the fields of an icmp header
CREATE TABLE icmphdr( sid         INT      UNSIGNED NOT NULL,
                      cid         INT      UNSIGNED NOT NULL,
                      icmp_type   TINYINT  UNSIGNED NOT NULL,
                      icmp_code   TINYINT  UNSIGNED NOT NULL,
                      icmp_csum   SMALLINT UNSIGNED,
                      icmp_id     SMALLINT UNSIGNED,
                      icmp_seq    SMALLINT UNSIGNED,
                      PRIMARY KEY (sid,cid),
                      INDEX       icmp_type (icmp_type));

# Protocol options
CREATE TABLE opt    ( sid         INT      UNSIGNED NOT NULL,
                      cid         INT      UNSIGNED NOT NULL,
                      optid       INT      UNSIGNED NOT NULL,
                      opt_proto   TINYINT  UNSIGNED NOT NULL,
                      opt_code    TINYINT  UNSIGNED NOT NULL,
                      opt_len     SMALLINT,
                      opt_data    TEXT,
                      PRIMARY KEY (sid,cid,optid));

# Packet payload
CREATE TABLE data   ( sid           INT      UNSIGNED NOT NULL,
                      cid           INT      UNSIGNED NOT NULL,
                      data_payload  TEXT,
                      PRIMARY KEY (sid,cid));

# encoding is a lookup table for storing encoding types
CREATE TABLE encoding(encoding_type TINYINT UNSIGNED NOT NULL,
                      encoding_text TEXT NOT NULL,
                      PRIMARY KEY (encoding_type));
INSERT INTO encoding (encoding_type, encoding_text) VALUES (0, 'hex');
INSERT INTO encoding (encoding_type, encoding_text) VALUES (1, 'base64');
INSERT INTO encoding (encoding_type, encoding_text) VALUES (2, 'ascii');

# detail is a lookup table for storing different detail levels
CREATE TABLE detail  (detail_type TINYINT UNSIGNED NOT NULL,
                      detail_text TEXT NOT NULL,
                      PRIMARY KEY (detail_type));
INSERT INTO detail (detail_type, detail_text) VALUES (0, 'fast');
INSERT INTO detail (detail_type, detail_text) VALUES (1, 'full');

# be sure to also use the snortdb-extra tables if you want
# mappings for tcp flags, protocols, and ports
```