PROPOSTA E IMPLEMENTAÇÃO DE UM MODELO DE GERÊNCIA DE SEGURANÇA DE REDES EMPREGANDO CORBA

André Sarmento Barbosa

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Luis Felipe Magalhães de Moraes Ph.D.

Prof. Jorge Lopes de Souza Leão D.Sc.

Prof. Maria Cristina Silva Boeres Ph.D.

RIO DE JANEIRO, RJ - BRASIL
AGOSTO DE 2002

BARBOSA, ANDRÉ SARMENTO

Proposta e Implementação de um Modelo de Gerência de Segurança de Redes Empregando CORBA [Rio de Janeiro] 2002

XI, 184 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2002)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Segurança de Redes de Computadores
2. Sistemas Distribuídos
3. CORBA

I. COPPE/UFRJ II. Título (série)

DEDICATÓRIA

À minha família, pelo apoio e compreensão.

Aos professores do ensino público, que apesar de todas as dificuldades se dedicam a nos ensinar muito bem, não só a técnica, mas também a ética profissional e a batalhar pelos nossos objetivos.

Com muito amor, à minha noiva Paula, que esteve o tempo todo ao meu lado, me incentivando e ajudando.

AGRADECIMENTOS

Agradeço a Deus, a força maior que nos conduz e ilumina.

Agradeço muito à minha família por apoiar sempre minhas decisões, ter me colocado sempre no caminho certo, me dando uma educação com dignidade, respeito, humor e amor.

É absolutamente impossível fazer todas as coisas sozinho, a qualidade do nosso trabalho aumenta a medida que mais e mais pessoas se envolvem, criticam, e ajudam. Eu tive a sorte de estar rodeado de amigos excepcionais que assim me ajudaram. Meu orientador e meus colegas do Laboratório Ravel, meus colegas da UFF e várias outras pessoas. Agradeço de coração a todos vocês.

À CAPES, pelo apoio financeiro durante a realização deste trabalho.

Aos colegas e funcionários do PESC que acompanharam esta jornada.

Agradeço com muito amor à minha noiva Paula, quem eu tive a sorte de conhecer desde o início na COPPE, e que me acompanha e me dá a maior força, uma verdadeira amiga com que eu divido e compartilho meus desafios e felicidades.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

PROPOSTA E IMPLEMENTAÇÃO DE UM MODELO DE GERÊNCIA DE SEGURANÇA DE REDES EMPREGANDO CORBA

André Sarmento Barbosa

Agosto/2002

Orientador: Luis Felipe Magalhães de Moraes

Programa: Engenharia de Sistemas e Computação

Ferramentas de segurança de redes, tais como sistemas de detecção de intrusão e *firewalls* geralmente não possuem qualquer tipo de interoperabilidade, além disso, cada ferramenta possui sua própria interface de gerência e protocolo de comunicação, tornando o gerenciamento de segurança complexo e ineficiente. Existe portanto, um problema evidente, gerado pela falta de padronização.

O objetivo desta tese é propor um modelo de gerência de segurança de redes baseado no padrão CORBA, descrevendo um protocolo que procura padronizar a comunicação de sistemas de detecção de intrusão e *firewalls*, possibilitando a interoperabilidade destas ferramentas.

Uma das principais vantagens do modelo proposto é a facilidade no desenvolvimento de software em várias linguagens de programação, sistemas operacionais e plataformas, permitindo que um novo *firewall* ou sistema de detecção de intrusão possa facilmente utilizar o modelo.

Para demonstrar a funcionalidade do modelo foi implementado e documentado um sistema chamado SIGSEC, que é capaz de gerenciar, através da Web, um sistema de detecção de intrusão e um *firewall*. Este sistema está totalmente baseado no modelo proposto e sua implementação serve como guia para o desenvolvimento de outros sistemas.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

PROPOSAL AND IMPLEMENTATION OF A NETWORK SECURITY MANAGEMENT MODEL USING CORBA

André Sarmento Barbosa

August/2002

Advisor: Luis Felipe Magalhães de Moraes

Department: Computing Systems Engineering

Network security tools, such as intrusion detection systems and firewalls generally don't have any type of interoperability, besides, each tool has its own management interface and communication protocol, becoming the security management complex and inefficient. There is therefore, a evident problem, caused by the lack of standardization.

The objective of this thesis is to propose a network security management model based on CORBA standard, describing a protocol that aims to standardize the communication of intrusion detection systems and firewalls, making possible the interoperability of this tools.

One of the main advantages of the proposed model is easiness in the software development in several programming languages, operating systems and platforms, allowing that a new firewall or intrusion detection system can easily use the model.

To show the validity of the model it was implemented and documented a system called SIGSEC, that is capable to manage, through the Web, one intrusion detection system and one firewall. This system is totally based on the proposed model and its implementation serves as guide for the development of other systems.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Cuidar da segurança de uma rede de computadores, seja ela simples ou complexa, é hoje uma das principais preocupações dos administradores de rede. Não basta somente manter os softwares atualizados, impor políticas de segurança, senhas fortes, anti-vírus, backup, etc. Tudo isto ajuda muito, mas é extremamente necessária a monitoração do sistema e uma correta configuração dos seus equipamentos. Por que? Porquê os invasores estão cada vez mais especializados e munidos de ferramentas cada vez mais sofisticadas.

O objetivo deste capítulo é alertar aos problemas de gerenciamento de segurança, expondo posteriormente uma forma de minimizar o problema, oferencendo um modelo de gerência baseado numa tecnologia padronizada que pode trazer muitos benefícios, utilizando-o todo ou em parte.

1.1 O Gerenciamento da Segurança

Quando falamos de configuração de sistemas no início deste capítulo notamos um grave problema, visto que não é só dos softwares que utilizamos no dia a dia ou nos servidores de rede que estamos falando. Estamos falando também da configuração dos próprios mecanismos de segurança. Não adianta possuir os melhores e mais caros mecanismos de segurança se estes não estão configurados adequadamente, e portanto não operando de forma inteligente.

Uma das principais causas do problema é o fator humano, somos nós que configuramos e monitoramos nossos *firewalls* e nossos sistemas de detecção de intrusão. Soma-se a isso a escassez de tempo dos administradores que geralmente cuidam de uma infinidade de outras tarefas num ambiente de rede.

Gerenciar a segurança uma rede de computadores pode envolver diversas tec-

nologias, por exemplo:

- Anti-vírus;
- Filtros de conteúdo;
- *Backup*;
- *Firewalls*;
- IDSs (*Intrusion Detection Systems*);
- VPNs (*Virtual Private Networks*);
- *Scanners* (Ferramentas para procurar portas ou falhas num *host* ou rede).

A questão é que estas ferramentas devem ser propriamente configuradas e monitoradas a fim de desempenharem seu papel da melhor forma. Melhor seria ainda se elas pudessem colaborar, tornando o sistema mais inteligente.

Dentre as diversas tecnologias de segurança, podemos colocar os anti-vírus como obrigatórios, principalmente nos sistemas baseados na plataforma da Microsoft. No nível de proteção de servidores e da rede julgamos os *firewalls* e IDSs como sendo as tecnologias mais utilizadas, pesquisadas e desenvolvidas atualmente. Mesmo contando com isso, foram vários os motivos que levaram ao desenvolvimento da proposta apresentada neste trabalho estar direcionada inicialmente para *firewalls* e sistemas de detecção de intrusão baseados em rede. Estes foram os principais motivos:

- Tecnologias mais empregadas na segurança de servidores e redes;
- Possibilidade de interoperabilidade com diversas vantagens;
- Problema crítico de monitoração;
- Problema crítico de configuração errônea;
- Falta de padronização para configuração e monitoração.

Um primeiro passo seria uma forma de padronização para as interfaces de IDSs e *firewalls* que ajudaria em muito o desenvolvimento e customização de sistemas de gerenciamento de segurança, permitindo interoperabilidade e podendo diminuir os problemas de configuração e monitoração.

1.2 A Necessidade da Padronização

Invasões e outras formas de ataque se tornam cada vez mais públicas e sofisticadas, a capacidade de um sistema de detecção de intrusão responder a tais ataques torna-se cada vez mais limitada se este sistema não pode cooperar com outros sistemas, de acordo com STANIFORD-CHEN [2] os esforços para esta "cooperação" tem sido direcionados primariamente para componentes homogêneos com pouca, ou senão, nenhuma atencão à padronização.

O modelo de gerenciamento de segurança descrito neste trabalho se baseia inicialmente numa padronização das interfaces de comunicação de IDSs e *firewalls* no que diz respeito a sua configuração e monitoração.

Ao tentar padronizar estas interfaces de comunicação existia a preocupação de atender aos seguintes pré-requisitos:

- Permitir compatibilidade com diversos softwares existentes;
- Facilitar o desenvolvimento de aplicativos utilizando o modelo proposto;
- Tornar o sistema o mais independente possível de sistemas operacionais específicos;
- Tornar o sistema o mais independente possível de hardwares específicos;
- Alcançar flexibilidade e escalabilidade do sistema.

Todos estes requisitos foram atendidos quase que totalmente com o uso de CORBA. O modelo está fundamentado em uma API baseada em CORBA para comunicação entre IDSs e *firewalls*, podendo ser estendida a outras tecnologias.

O uso da liguagem de definição de interfaces (IDL) do padrão CORBA permite então padronizar os atributos e operações que podem ser realizadas nas ferramentas de segurança, como será apresentado no capítulo 4. O uso de um sistema distribuído permite ainda escalabilidade e uma vasta gama de possibilidades de métodos de implementação, um destes métodos foi desenvolvido e compõe a metodologia proposta nesta tese.

1.3 Organização do Texto

Para uma melhor compreensão vamos fazer uma breve análise dos capítulo deste trabalho. Cada capítulo está bem definido dentro do escopo do trabalho, e apesar do

capítulo 4 promover a junção e concretizar as idéias expostas nos capítulos anteriores não existe sobreposição nem repetição dos assuntos entre os capítulos, exceto a introdução.

Capítulo 1: Introdução, explicando o objetivo, motivação e organização deste trabalho.

Capítulo 2: Aborda as ferramentas de segurança, com ênfase nos *firewalls* e sistemas de detecção de intrusão, as duas ferramentas que mais motivaram o desenvolvimento do modelo proposto.

Capítulo 3: Como visto nas seções anteriores o padrão CORBA é o catalisador de todo o trabalho, permitindo a padronização e inúmeras vantagens devido ao seu uso. Este capítulo introduz o padrão CORBA, e depois se aprofunda em todas as funcionalidades utilizadas neste trabalho.

Capítulo 4: Neste capítulo é proposto o modelo de gerência de segurança, empregando fortemente os conceitos dos dois capítulos anteriores, detalhando-o através de diagramas e tabelas.

Capítulo 5: O desenvolvimento de um protótipo usando o modelo proposto é descrito neste capítulo. Este protótipo foi chamado de SIGSEC [15] (Sistema de Gerência de Segurança Empregando CORBA). Testes e avaliações do protótipo são apresentados no final.

Capítulo 6: Aqui temos a conclusão e sugestões para futuros trabalhos. Devido a possibilidade de extensão do protótipo e o interesse em seu crescimento, são feitos comentários sobre a contribuição da comunidade com o projeto.

Referências e Apêndices: Além das referências bibliográficas, constam apêndices com os principais protocolos da Internet detalhados, o código fonte do protótipo e ainda um glossário.

1.4 Objetivos e Contribuições do Trabalho

O objetivo principal é padronizar uma interface baseada no padrão CORBA para facilitar a comunicação e gerência de dispositivos de segurança de redes. Esta padronização é usada no SIGSEC [15] e pode ser usada em quaisquer outros sistemas que necessitem comunicações com *firewalls* ou IDSs. O SIGSEC [15] foi o

protótipo implementado neste trabalho, detalhado no capítulo 5. Em segundo lugar temos como objetivo o desenvolvimento de *drivers* para diversos *firewalls* e IDSs que serão desenvolvidos com o modelo e API propostas.

O SISGEC é baseado no conceito ISMS (*Integrated Security Management System*) o qual possui alguns representantes no mercado, tais como a plataforma OPSEC da *Check Point* e o *Active Security* da NAI.

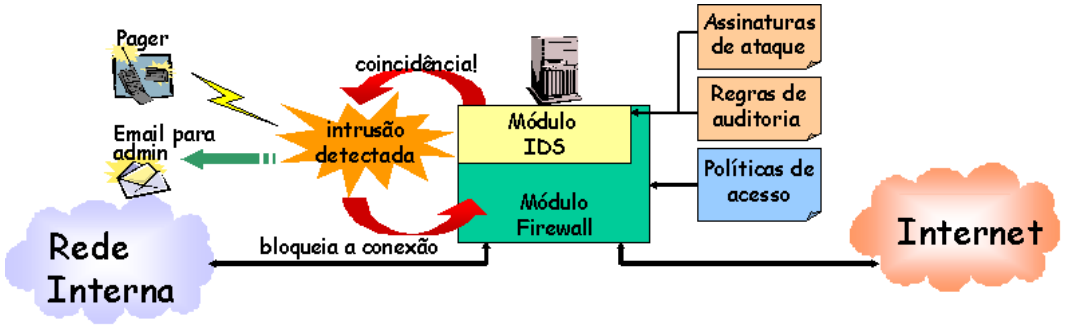


Figura 1.1: Modelo de Integração. Fonte: *Integrated Security Management, Real-Time Systems Laboratory, 2000*

Na Figura 1.1, CHUNG [3] propõe um modelo para sistemas integrados de gerenciamento de segurança que engloba funcionalidades de *firewalls* e IDSs que podem interoperar.

Este modelo de integração corresponde perfeitamente ao utilizado no projeto do SIGSEC [15].

O desenvolvimento do SIGSEC é suportado pelo site *http://www.sigsec.org* (Figura 1.2) que é o site original do projeto, fundado pelo autor, e que conta com a contribuição da comunidade acadêmica e de segurança.

O projeto SIGSEC [15] é aberto para contribuições tais como:

- Ajudando a desenvolver novos *drivers* para IDSs e *firewalls* disponíveis;
- Propondo alterações nas atuais padronizações de mecanismos de segurança;
- Propondo novas padronizações para outros mecanismos de segurança;
- Ajudando no suporte a outros ORBs;
- Descrevendo seus testes e experiências;
- Desenvolvendo interfaces/módulos de gerência e configuração;

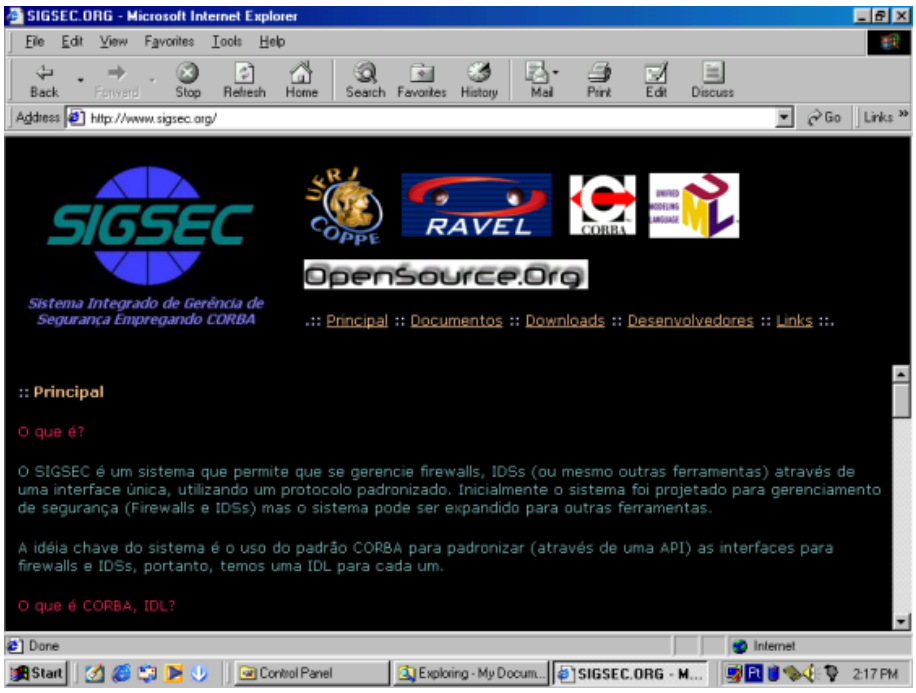


Figura 1.2: Site Oficial do Projeto SIGSEC

- Propondo novas idéias baseadas no modelo proposto de gerenciamento utilizando CORBA.

Além disso o projeto apóia o desenvolvimento de aplicações em CORBA não relacionadas diretamente com o propósito apresentado, e pode hospedar gratuitamente sites para o desenvolvimento de tais aplicações, a idéia é disseminar o uso de CORBA na comunidade e prover um ponto de encontro para troca de idéias entre desenvolvedores.

Capítulo 2

Ferramentas de Segurança

O primeiro ponto a se pesquisar na implementação de contramedidas de segurança de rede é considerar todas as opções possíveis. Por exemplo, no que diz respeito à conexão com a Internet: Você poderia simplesmente não usar a conexão? Você poderia optar por outro provedor de serviço (ISP ou *backbone*)? Uma conexão com a Internet representa muitos riscos, então, é necessário estudar bem as opções existentes em cada caso, não só no que se refere ao provedor, mas na forma como sua rede estará conectada.

Depois de todas opções estudadas e decisões tomadas, agora podemos pensar em contramedidas específicas para os diversos pontos críticos da rede. Estas contramedidas estão diretamente ligadas à política de segurança adotada e as ferramentas utilizadas. A escolha das ferramentas de segurança e a sua forma de utilização são fatores fundamentais. Tais considerações podem ser encontradas em [57] e [53].

Como existem sempre muitas opções, muitas tecnologias, diferentes tipo de hardware e diferentes níveis de experiência de administradores, a escolha das ferramentas de segurança é razoavelmente um processo difícil. Vamos examinar primeiramente o universo das ferramentas de segurança, dando destaque para as mais utilizadas, depois examinaremos os *firewalls* e os sistemas de detecção de intrusão (IDSs), que são as ferramentas incluídas inicialmente no modelo de gerência proposto. Finalmente, apresentaremos alguns sistemas que permitem a integração destas ferramentas de alguma forma.

2.1 Diversidade de Ferramentas de Segurança

Existem ferramentas de segurança para atender a um grande número de aplicações: clientes de email, web, autenticação de usuários, proteção a nível de rede, monito-

ração, gerenciamento, etc. Como apresentado na SECURITY FOCUS [12], podemos dividir as ferramentas de segurança nas seguintes categorias:

Controle de Acesso: Listas de controle de accesso de roteadores (ACLs), *firewalls*;

Auditoria: Verificadores de integridade de arquivos, analisadores de *logs*;

Autenticação: Certificados digitais, gerenciamento de senhas, autenticação de usuários;

Criptografia: Criptoanálise, VPNs, criptografia de dados e correio eletrônico;

Código Hostil: Anti-vírus, detecção/prevenção de ataque de negação de serviço;

Detecção de Intrusão: IDSs baseados em *host* e em rede;

Monitoração de rede: Analisadores de tráfego e *Sniffers*;

Senhas: Recuperadores, armazenadores e geradores;

Backup: Geradores e recuperadores de backup;

Scanners: Scanners de porta e de vulnerabilidades;

Utilitários: Removedores seguros de dados, criadores de túneis em rede;

Gerenciamento: Gerenciamento de usuários, *firewalls*, sistema operacional.

Algumas categorias foram adaptadas devido ao uso da língua portuguesa, podemos notar também que algumas ferramentas podem naturalmente se enquadrar em mais de uma categoria. O importante é notar que existe uma enorme quantidade de ferramentas e que, na prática, existem ferramentas muito mais utilizadas que outras.

Na mais recente pesquisa feita pela CSI/FBI [64] podemos verificar o grau de utilização das ferramentas de segurança (Figura 2.1), respondida por organizações de diversos setores da indústria (Figura 2.2). Vamos examinar o impacto dos resultados desta pesquisa neste trabalho. Notamos inicialmente que os anti-vírus, *firewalls*, mecanismos de controle de acesso e seguraça física são as ferramentas mais utilizadas.

O uso do anti-vírus é realmente obrigatório, pois os vírus de computador são hoje os principais causadores de prejuízos nas organizações. Apesar dos anti-vírus poderem trabalhar de alguma forma em rede (analisando emails num servidor de

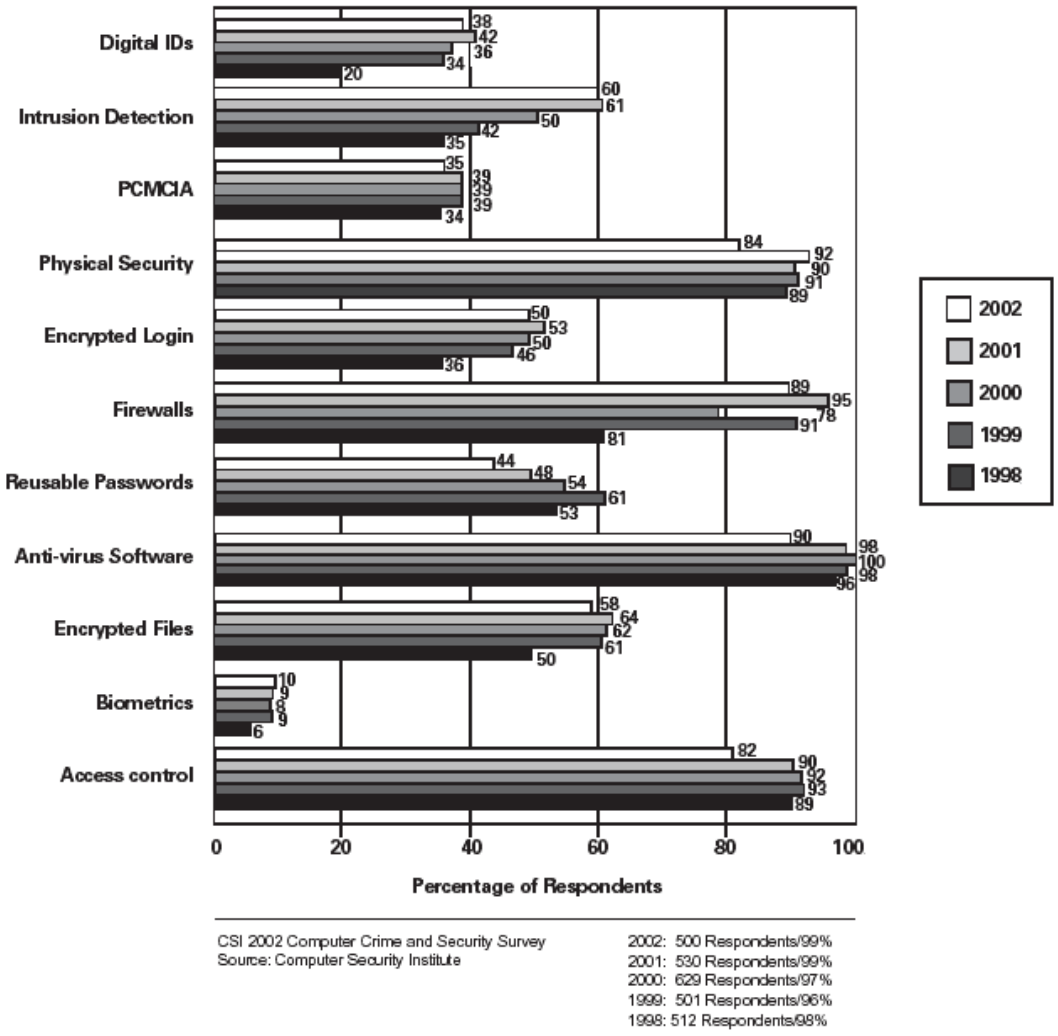


Figura 2.1: Tecnologias de Segurança mais Utilizadas. Fonte: *CSI/FBI - 2002, Computer Crime and Security Survey.*

correio por exemplo), eles não agem diretamente na rede IP. O anti-vírus é normalmente instalado localmente, em cada máquina na rede, e possui uma lista de assinaturas que é necessariamente incremental, atualizada automaticamente na maioria dos casos. Assim sendo, o anti-vírus é obrigatório, mas seu gerenciamento não é algo demasiado complexo. Estas características fazem dos anti-vírus sistemas geralmente independentes e pelos fatos descritos anteriormente não foram inicialmente considerados no modelo de gerência de segurança proposto.

Os mecanismos de segurança física estão numa situação similar aos anti-vírus. Eles são igualmente importantes, mas não agem diretamente na rede, possuindo características peculiares, e portanto, também não foram incluidos inicialmente no modelo de gerência de segurança proposto.

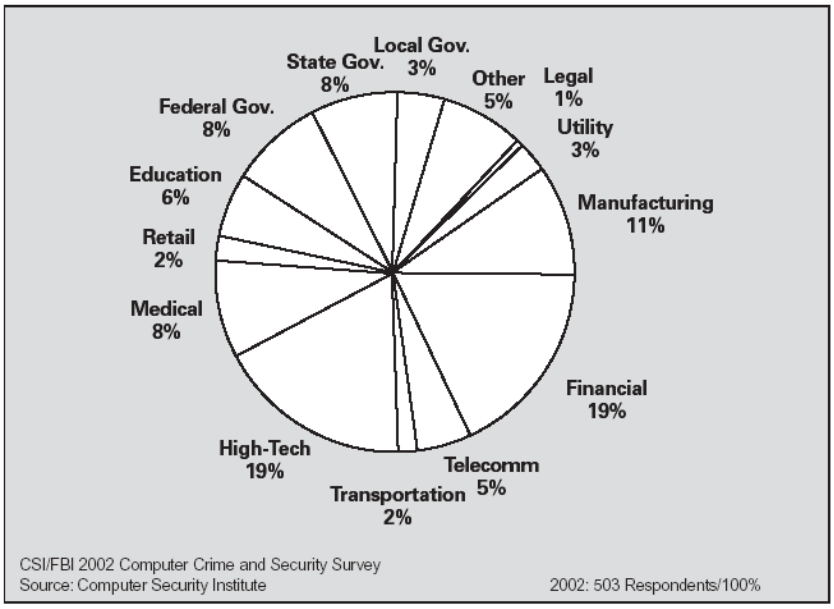


Figura 2.2: Setores que Responderam ao Levantamento de Segurança. Fonte: *CSI/FBI - 2002, Computer Crime and Security Survey.*

Os mecanismos de controles de acesso (geralmente filtros de pacotes e/ou filtros de conteúdo) e principalmente os *firewalls*, são hoje considerados obrigatórios. Os *firewalls*, primordialmente, possuem listas de controle de acesso (ou regras), e portanto, podem também ser considerados mecanismos de controle de acesso. O conceito de **mecanismo de controle de acesso** pode ter diferentes significados para diferentes organizações, por exemplo, um roteador com uma lista de controle de acesso poderia ser tratado como um mecanismo de controle de acesso. Os *firewalls* fazem parte do modelo de gerência proposto nesta tese, e por isso serão melhor analisados na seção seguinte.

Verificamos também o uso de criptografia, autenticação segura e sistemas de detecção de intrusão como as tecnologias que mais vem sendo empregadas. Os mecanismos de criptografia e autenticação podem variar muito, por exemplo: PGP, Certificados Digitais, SSH, Kerberos, etc. Os sistemas de detecção de intrusão, além de estarem em alto crescimento de uso (60% como mostra o gráfico), podem ser razoavelmente bem definidos dentro de duas categorias: baseados em rede e baseados em *host*. Os sistemas de detecção de intrusão baseados em rede serão analisados em uma seção particular, pelo fato de estarem também incluídos do modelo de gerêcia que será proposto.

2.2 *Firewalls*

Um *firewall* é basicamente um mecanismo de proteção. Quando se implementa um firewall a primeira coisa que deve ser analisada é o que está se querendo proteger. Quando a rede de uma empresa está conectada, ela está colocando em risco:

- Seus dados e informações;
- Seus recursos;
- Sua reputação.

Os dados e as informações possuem três características básicas:

Confidencialidade: Existem informações que outras pessoas não podem saber ou acessar.

Integridade: Outras pessoas não podem alterar estas informações.

Acessibilidade: Determinadas pessoas devem poder utilizar estas informações.

Estas características precisam ser bem administradas por ferramentas de segurança configuradas corretamente. Um *firewall* pode ajudar bastante neste aspecto.

2.2.1 A Correta Definição de *Firewall*

Atualmente o termo *firewall* tem sido utilizado com inúmeras representações diferentes, desde um PC rodando um software que filtra pacotes até um sistema completo com uma combinação de hardware dedicado e software. A questão é, o que é correto? Segundo RANUM [65], um dos maiores especialistas no assunto:

"Um *firewall* é um sistema ou grupo de sistemas que impõe uma política de controle de acesso entre duas redes, o significado atual que é adotado pode variar largamente, mas em princípio, um *firewall* pode ser imaginado como um par de mecanismos: um que existe para bloquear tráfego e outro que existe para permitir tráfego. Alguns *firewalls* podem enfatizar altamente um ou outro mecanismo. Provavelmente o fato mais importante a reconhecer sobre um *firewall* é que ele implementa uma política de controle de acesso."

Podemos acrescentar que muitos *firewalls* atualmente (comerciais e gratuitos) possuem outras características adicionais, tais como:

- NAT (*Network Address Translation*);
- Autenticação;
- Filtro de conteúdo.

2.2.2 Arquiteturas de Firewalls

As definições destas arquiteturas de *firewalls* foram baseadas nas descritas por STALLINGS [25].

Roteador com listas de acessos e *gateways* simples: Este é o caso mais simples onde temos geralmente um roteador implementando algumas ACLs (listas de controle de acesso) ou um *gateway* com um software de *firewall* fazendo apenas filtragem de pacotes (Figura 2.3).

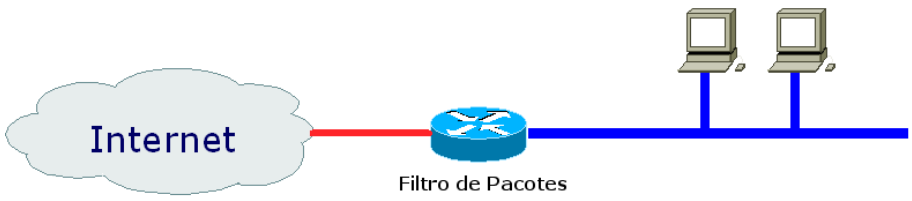


Figura 2.3: Roteador com Listas de Acesso ou *Gateway* Simples

***Screened host firewall (single-homed bastion)*:** A arquitetura denominada *single-homed* fornece serviços através de uma máquina chamada bastião que possui apenas uma interface com a rede. Nesta arquitetura, o primeiro nível de segurança é fornecido pela filtragem de pacotes.

A filtragem de pacotes no roteador é configurada de tal forma que a máquina bastião é a única que usuários da internet podem abrir conexão, e mesmo assim, somente alguns tipos de conexão são permitidos. Qualquer sistema externo que tente acessar serviços internos terá que passar pela máquina bastião (Figura 2.4).

***Screened Host Firewall (dual-homed bastion)*:** Esta arquitetura é montada utilizando-se uma máquina (bastião) que possui pelo menos duas interfaces de rede. A configuração é similar ao caso anterior. Para se implementar essa arquitetura de *firewall* deve se desabilitar a função de roteamento no bastião.

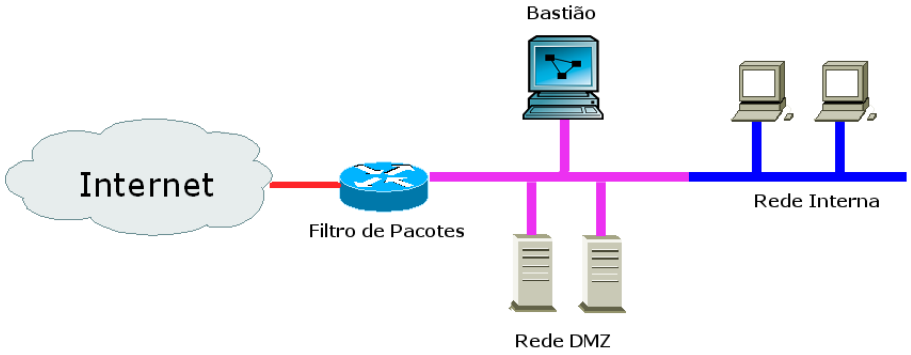


Figura 2.4: *Screened Host Firewall (single-homed bastion)*

Desta forma, pacotes IP de uma rede não são roteados diretamente para a outra rede. Máquinas na rede interna ou na Internet podem se comunicar com a máquina bastião, mas estas máquinas não podem comunicar entre si diretamente, o tráfego deve passar através de *proxies*, NAT ou outros controles definidos no bastião (Figura 2.5).

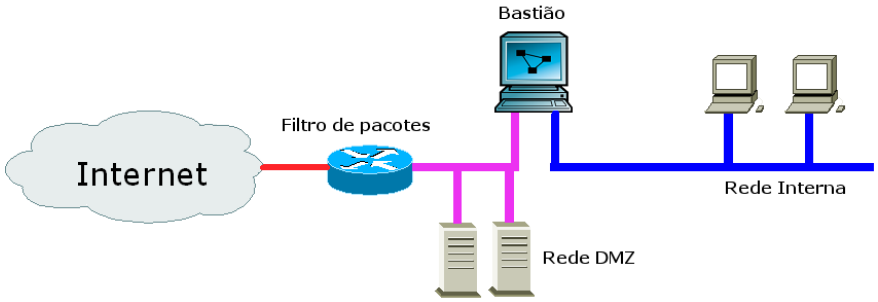


Figura 2.5: *Screened Host Firewall (dual-homed bastion)*

***Screened Subnet Firewall*:** Na arquitetura *Screened subnet firewall* coloca-se uma camada extra de segurança em relação à arquitetura *dual-homed*, adicionando uma sub-rede que isola a rede interna da internet.

A justificativa dessa arquitetura é que a máquina bastião é a máquina mais vulnerável e mais visada, podendo ser comprometida.

Isolando a maquina bastião em uma sub-rede pode-se reduzir os impactos de uma possível invasão ao sistema (Figura 2.6).

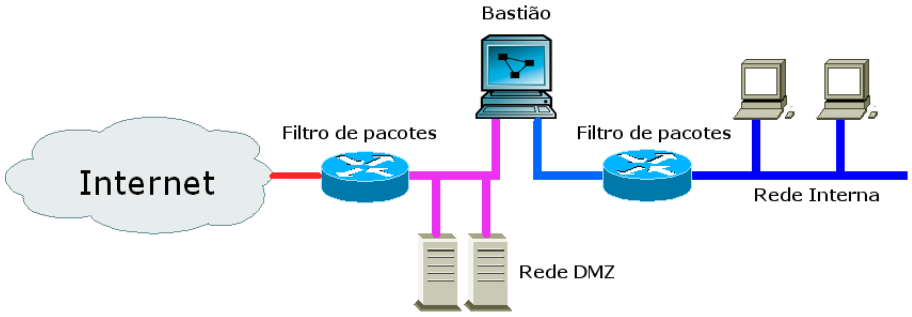


Figura 2.6: *Screened Subnet Firewall*

2.2.3 Tecnologias Mais Utilizadas em Firewalls

Filtro de pacotes: A filtragem de pacotes é um mecanismo de segurança de rede que funciona controlando quais pacotes podem fluir de uma sub-rede para outra.

Para transmitir informação através de uma rede, a informação tem que ser dividida em pequenos pedaços, dos quais cada um é enviado separadamente. Em uma rede IP esses pequenos pedaços são chamados de pacotes. Todas as transferências de dados em uma rede IP acontecem em formas de pacotes.

O roteador é o dispositivo básico que interconecta as redes IP. Os pacotes contêm informações de sua origem e seu destino e os roteadores se comunicam entre si utilizando protocolos de roteamento, como por exemplo o RIP (*Routing Information Protocol*).

Os roteadores ou *gateways*, quando agem como filtros de pacotes, além de se comunicarem para saber para onde eles devem enviar aqueles pacotes eles verificam se devem transmitir aqueles pacotes ou não.

Filtros de pacotes controlam (permitem ou bloqueiam) a transferência de dados baseando-se principalmente nos seguintes dados:

- O endereço de origem do pacote;
- O endereço de destino do pacote;
- A porta de origem do pacote;
- O porta de destino do pacote;
- Os protocolos de sessão e aplicação utilizados para a transferência dos dados.

Um pacote possui duas partes bem distintas: o cabeçalho e corpo. O cabeçalho contém informações relevantes para o correto processamento e encaminhamento do pacote, enquanto o corpo contém os dados que o pacote transporta (podendo incluir outros cabeçalhos).

A maioria dos filtros de pacotes não realizam nenhuma ação baseado no conteúdo dos dados do corpo do pacote , ou seja, eles não tomam decisões baseadas na camada de aplicação do protocolo TCP/IP.

Para a filtragem de pacotes o mais importante é analisar as informações contidas nos cabeçalhos.

***proxies*:** Sistemas de *proxy* fornecem acesso a internet para uma máquina, ou um número pequeno de máquinas, ao invés de fornecer acesso para todas as máquinas. As máquinas clientes que querem ter acesso a serviços na internet se comunicam com o servidor *proxy* ao invés de se comunicar diretamente com a internet. O servidor *proxy* avalia as requisições do cliente e decide quais delas passam e quais serão descartadas. Uma fez que a requisição é aprovada o servidor *proxy* age como intermediador da comunicação.

O servidor *proxy* permite a passagem unicamente daqueles serviços para qual existe um *proxy*. Se um servidor possui *proxies* para FTP e telnet, unicamente FTP e telnet são permitidos dentro da sub-rede protegida para a Internet. Todos outros serviços são completamente bloqueados. O *proxy* certifica-se que unicamente serviços confiáveis são permitidos através do *firewall* e previne que serviços não confiáveis sejam implementados no *firewall* sem o dado conhecimento.

Servidores *proxy* são aplicações especializadas que atuam como "procuradores" entre clientes da rede interna e servidores na rede externa. Pode-se dizer que *proxies* são compostos por dois elementos: um servidor *proxy* e um cliente *proxy*. Os clientes da rede interna fazem requisições por meio de clientes *proxy* previamente instalados. A conexão é feita através do servidor de *proxy* para aquele serviço, que verifica as permissões para que a conexão seja estabelecida com seu destino real.

Se a conexão é permitida, o servidor *proxy* contata o servidor real que foi especificado no cliente. Caso contrário, a conexão é rejeitada. A utilização de um *proxy* não requer hardware especial, mas requer que software seja instalado

ou esteja disponível no cliente. Como é um mecanismo que atua na camada de aplicação do protocolo IP é necessário que exista um *proxy* específico para cada serviço. Devido a estes motivos, os *proxies* não foram inicialmente inseridos no modelo de segurança proposto nesta tese.

NAT: Como mais e mais negócios e usuários progridem na Internet, endereços IP válidos estão se tornando mais difíceis de obter. A solução para muitos casos têm sido a tradução de endereço de rede (NAT). O NAT proporciona uma forma poderosa de se ter uma rede conectada à Internet sem precisar adquirir ou alugar endereços IP válidos para cada máquina. O NAT é também conhecido como *IP Masquerading* na cultura Linux.

O NAT permite que usuários de uma rede interna acessem a Internet através de diferentes endereços IPs inválidos na Internet. Cada máquina na rede interna está configurada para usar um IP inválido Ex.: (10.10.1.10).

Quando um cliente na rede interna deseja conectar para um servidor na Internet ele envia seus pacotes através na máquina que executa o NAT. No cabeçalho do pacote IP está o endereço do cliente (Ex.: 10.10.1.10) e o endereço IP da máquina que se deseja conectar (Ex.: 123.45.67.89).

A máquina que está executando o NAT intercepta este pacote e altera o endereço IP do cliente de 10.10.1.10 para seu endereço IP válido (Ex.: 123.5.5.5). Isso faz com que o servidor com quem está se conectando receba o pacote da máquina que está executando o NAT, e não do cliente que realmente enviou o pacote.

Seguindo o exemplo, o servidor (123.45.67.89) envia as respostas para a máquina NAT. Quando a máquina NAT recebe a resposta ela traduz o IP de destino para a máquina cliente que iniciou a conexão.

Para que a máquina que executa o NAT consiga tal proeza ela mantém o estado das conexões ativas e utiliza as portas de origem para multiplexar as conexões.

O processo é exemplificado abaixo:

$$[\text{Cliente}] \Longleftrightarrow [\text{NAT}] \Longleftrightarrow [\text{Servidor}]$$

Pacote 1: (Cliente $\Rightarrow$ NAT)

- Origem: 10.10.0.10
- Destino: 123.45.67.89

Pacote 1*: (NAT ⇒ Sevidor)

- Origem: 123.5.5.5
- Destino: 123.45.67.89
- **Ocorreu tradução*

Pacote 2: (Sevidor ⇒ NAT)

- Origem: 123.45.67.89
- Destino: 123.5.5.5

Pacote 2*: (NAT ⇒ Cliente)

- Origem: 123.45.67.89
- Destino: 10.10.0.10
- **Ocorreu tradução*

O manual [27] do IP Filter [28] provê vários exemplos de configurações de NAT e filtro de pacotes.

Existem outras características que vários *firewalls* incorporam atualmente, podemos citar:

- Filtros de conteúdo;
- Autenticação;
- Criptografia.

Nenhuma característica adicional foi inclúida no modelo proposto nesta tese, somente foram padronizadas as configurações de filtro de pacotes e NAT que são as tecnologias presentes em todos os *firewalls* existentes.

2.2.4 Firewalls Gratuitos e Comerciais

A fim de criar um modelo genérico para a configuração de *firewalls*, foram estudados alguns dos softwares mais utilizados no mercado. Nesta seção apresentamos os modelos estudados e posteriormente serão apresentados detalhes de configuração genéricos para filtro de pacotes e NAT, que serão a base para a construção da padronização das interfaces de gerência.

FireWall-1: O Firewall-1 da Check Point [29] é uma suíte de segurança para a Internet e Intranet/Extranet. Suas características habilitam o acesso de usuários com autenticação, criptografia, tradução de endereço de rede (NAT) e serviços de análise e bloqueio de conteúdo. O FireWall-1 é uma solução integrada de segurança que atende as demandas de organizações de grande e pequeno porte. O conjunto de produtos é unificado pela arquitetura OPSEC [30] (*Open Plataform for Secure Enterprise Connectivity*) oferecendo um ambiente de administração e configuração centralizado. O OPSEC será analisado em mais detalhes na Seção 2.4.

A Check Point é líder do mercado de *firewall* e VPN (*Virtual Private Network*) com mais de 125.000 produtos instalados, incluindo mais de 42.000 instalações de VPN *gateway* e 28 milhões de clientes que possuem acesso remoto VPN.

Aker: A empresa Aker [31] produz um *firewall* nacional com alta reputação no mercado, baseado no sistema operacional FreeBSD, possuindo uma inteface de gerenciamento pelo Windows, além de muitas caraterísticas encontradas em outros *firewalls* de custo superior.

IPF: O IP Filter (ou IPF) [28] é um dos *firewalls* gratuitos mais utilizados, podendo ser instalado em diversos sistemas operacionais, tais como: NetBSD, FreeBSD, Solaris, etc. Uma característica fundamental do IPF é que ele possui dois arquivos de configuração (filtro de pacotes e NAT), de forma que não é necessário executar vários comandos consecutivamente a fim de se obter a funcionalidade desejada do *firewall*. Além de filtro de pacotes e NAT existem também alguns *proxies* disponíveis.

Netfilter: Este é o *firewall* [32] disponível atualmente no Linux. Neste *firewall*, os comandos são executados através de um aplicativo (**iptables**) que altera de forma dinâmica as configurações de filtro de pacotes e NAT. Os comandos montados com o **iptables** podem ser incluidos em um script, obtendo um funcionamento similar ao uso do IPF. Alguns *proxies* também existem para o Netfilter.

OpenBSD PF: Este é o *firewall* disponível no OpenBSD [20]. O OpenBSD é considerado um dos sistemas operacionais mais seguros atualmente. Este sistema utilizava antigamente o IPF como seu *firewall* nativo, mas devido a questões

de licença o IPF foi descartado e a equipe do projeto desenvolveu o seu próprio *firewall*. Apesar de ser completamente desenvolvido de forma independente, este *firewall* possui configuração quase idêntica ao IPF.

2.2.5 Configuração de Filtro de Pacotes e NAT

Os filtros de pacotes podem ser configurados a partir de regras que bloqueiam ou permitem que os pacotes sejam transmitidos, baseando-se em diversos campos dos cabeçalhos TCP/IP. Vamos examinar inicialmente as regras mais comuns. Para isso utilizaremos uma linguagem simples:

`block in:` Bloqueia a entrada de pacotes no *firewall*;

`block out:` Bloqueia a saída de pacotes do *firewall*;

`pass in:` Permite a entrada de pacotes no *firewall*;

`pass out:` Permite a saída de pacotes do *firewall*;

`from:` Endereço IP de origem;

`to:` Endereço IP de destino;

`any:` Qualquer endereço IP.

Alguns exemplos para ilustrar:

```
block in from 192.168.1.1 to any
```

Esta regra irá bloquear os pacotes que chegam, vindos da máquina cujo IP é 192.168.1.1 e se destinam a qualquer IP. Resumindo: Esta regra impede que a máquina 192.168.1.1 passe através do *firewall*.

```
block in from any to 192.168.2.2
```

Esta regra irá bloquear os pacotes que chegam de qualquer máquina e se destinam a máquina cujo IP é 192.168.2.2. Resumindo: Esta regra impede que qualquer máquina tente acessar a máquina 192.168.2.2.

Estes exemplos utilizam a sintaxe do *firewall* IPF, mas podem ser considerados bastantes genéricos pois até mesmo as regras mais complexas podem ser bem compreendidas e implementadas em qualquer outro *firewall*, devido ao fato das regras estarem intimamente ligadas aos campos dos cabeçalhos TCP[38], IP[36], UDP[35] e ICMP[37].

Vamos considerar mais alguns campos dos protocolos e demonstrar mais algumas regras:

`proto:` Protocolo encapsulado na camada IP;

`port:` Porta de origem ou destino;

`icmp-type:` Tipo de pacote ICMP;

`flags:` Flags marcadas no protocolo TCP.

Exemplos:

```
block in proto tcp from any to 192.168.2.2 port = 80
```

Esta regra irá bloquear os pacotes TCP que chegam de qualquer máquina e se destinam a máquina cujo IP é 192.168.2.2 na porta 80. Resumindo: Esta regra impede que qualquer máquina tente acessar um servidor Web (porta 80 = HTTP) na máquina 192.168.2.2.

```
pass in proto icmp from any to 192.168.1.1 icmp-type = 11
pass in proto icmp from any to 192.168.1.1 icmp-type = 0
```

Estas regras permitem que pacotes ICMP vindos de qualquer máquina passem pelo *firewall* e atinjam a máquina 192.168.1.1, mas somente os pacotes do tipo 11 e do tipo 0. Resumindo: Esta regra permite que a máquina 192.168.1.1 possa receber um "ping", pois o "ping" usa exatamente estes tipos de pacotes ICMP (*echo request* e *echo reply*).

A configuração para o NAT é bastante similar, incluindo mais algumas palavras chave podemos exemplificar alguns cenários:

`map:` Palavra chave que significa mapear ou traduzir

<interface>: Indicativo da interface de rede na qual o *firewall* fará o mapeamento

Exemplo:

```
map eth0 10.10.0.10 -> 123.5.5.5
```

Esta regra de NAT espelha o comportamento apresentado na seção anterior, ou seja, quando um pacote sai pela interface de rede `eth0` com endereço de origem 10.10.0.10 (rede interna) este endereço é mapeado para 123.5.5.5 (endereço IP válido). O processo inverso é feito automaticamente para os pacotes que chegam e que foram mapeados por essa conexão, o que significa que o *firewall* tem que manter o estado de cada conexão que passa através do NAT.

Em geral, as interfaces de configuração dos *firewalls* são compostas das duas listas independentes de regras, filtro de pacotes e NAT. O que difere é a forma como estas regras são atualizadas e processadas. No Capítulo 4 analisaremos todas as palavras chave que podem aparecer nestas regras e que podem ser padronizadas para todos os *firewalls*. Esta análise é a base para montar a padronização em CORBA para *firewalls* descrita neste mesmo capítulo.

2.3 Sistemas de Detecção de Intrusão (IDSs)

Baseado nas informações de várias entidades de pesquisa em segurança tais como CERT[43] e ICSA[44] podemos afirmar que um sistema foi atacado ou invadido mais que uma vez por segundo nos últimos anos. Só nos Estados Unidos, a ICSA[44] identificou que, em média, um site ou computador foi invadido a cada 11 minutos.

Estas estatísticas nos levam a desejar uma enorme necessidade de poder rastrear e identificar estes ataques. O sistema que possui a capacidade de fazer isso é um sistema de detecção de intrusão, ou IDS como é chamado mais frequentemente (IDS, do inglês: *Intrusion Detection System*). Estamos definindo o termo ataque conforme a RFC 2828 [34], onde 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. Um ataque bem sucedido pode caracterizar uma invasão ou até mesmo a negação de serviços no sistema alvo (DoS, do inglês: *Denial of Service*).

Na seção seguinte classificamos os IDSs quanto a sua forma de funcionamento,

tecnologia empregada e sistema a ser monitorado. Dentre as tecnologias empregadas a mais utilizada é a análise de assinaturas.

Ao longo do texto o acrônimo IDS (*Intrusion Detection System*) está sendo usado com o significado de sistema de detecção de intrusão, o acrônimo em inglês foi escolhido por ser de uso corrente na literatura brasileira e estrangeira, além disso, o acrônimo IDS pode caracterizar sistemas baseados em rede ou em host, para os IDS baseados em rede utiliza-se também o acrônimo NIDS (do inglês: *Network Intrusion Detection System*).

2.3.1 Definição e Classificações de um IDS

Sistema de detecção de intrusão: Sistema composto de hardware e software que trabalham juntos para identificar eventos inesperados que podem indicar que um ataque está acontecendo ou aconteceu em um sistema operacional ou ambiente de rede.

Podemos classificar os IDS quanto a tecnologia empregada, um IDS pode trabalhar com:

- Análise de assinaturas;
- Análise estatística;
- Sistemas adaptativos.

O mecanismo de análise de assinaturas funciona de forma similar a um anti-vírus, e por ser o método mais utilizado iremos estudá-lo com mais detalhes na sub-seção seguinte.

Um sistema de análise estatística constrói modelos estatísticos do ambiente se baseando em fatores tais como a duração média de uma sessão de telnet, por exemplo. Qualquer desvio de um comportamento normal pode ser identificado como suspeito, tal como uma variação grande da média descrita anteriormente.

Um sistema adaptativo começa por generalizar regras de aprendizado para o ambiente que está inserido e então determinar o comportamento dos usuários em relação ao sistema. Depois do período de aprendizado, o sistema pode reconhecer determinados padrões como sendo acessos normais ou ataques. Uma rede neural pode ser a escolha natural para tais sistemas.

Naturalmente podemos ter produtos que empregam mais de uma tecnologia.

A divisão clássica dos IDSs é quanto ao ambiente no qual ele está inserido e monitorando, desta forma podemos ter:

- IDSs baseados em rede (NIDS);
- IDSs baseados em *host*;
- Verificadores de integridade de arquivos.

Esta classificação foi baseada na sugerida pela ICSA[44].

Vamos examinar cada tipo separadamente, lembrando que produtos atuais já estão suportando algumas funções em comum, ou seja, IDS baseados em rede podem fazer algumas verificações no *host* onde estão instalados. Os IDS baseados em *host* fazem algum tipo de monitoramento na rede, e tanto os baseados em rede como os baseados em *host* podem já ter embutidos verificadores de integridade de arquivos.

IDSs baseados em rede (NIDS) :

Os NIDS possuem geralmente dois componentes:

Sensor: Agente principal de um IDS cuja função é monitorar um *host* ou rede, a fim de identificar intrusões, gravar *logs* localmente e gerar mensagens alertando tais eventos. Estas mensagens podem ou não ser enviadas a uma estação de gerenciamento.

Estação de gerenciamento: É uma estação encarregada de administrar um ou mais sensores espalhados pela rede. O software utilizado deve ter uma interface gráfica que permita configuração e monitoração dos agentes (Sensores IDS).

Os sensores, apesar de estarem instalados numa máquina específica, monitoram todo o tráfego no segmento de rede, pois a interface com a rede é colocada no chamado "modo promíscuo". Neste modo, o software recebe todos os pacotes que a interface consegue captar no segmento de rede, não apenas os destinados ao IP da interface, por isso, um único sensor tem a capacidade de monitorar a atividade de rede em diversas máquinas.

Principais vantagens encontradas nos NIDS:

- Detectam acessos com excesso de autoridade ou sem autoridade;

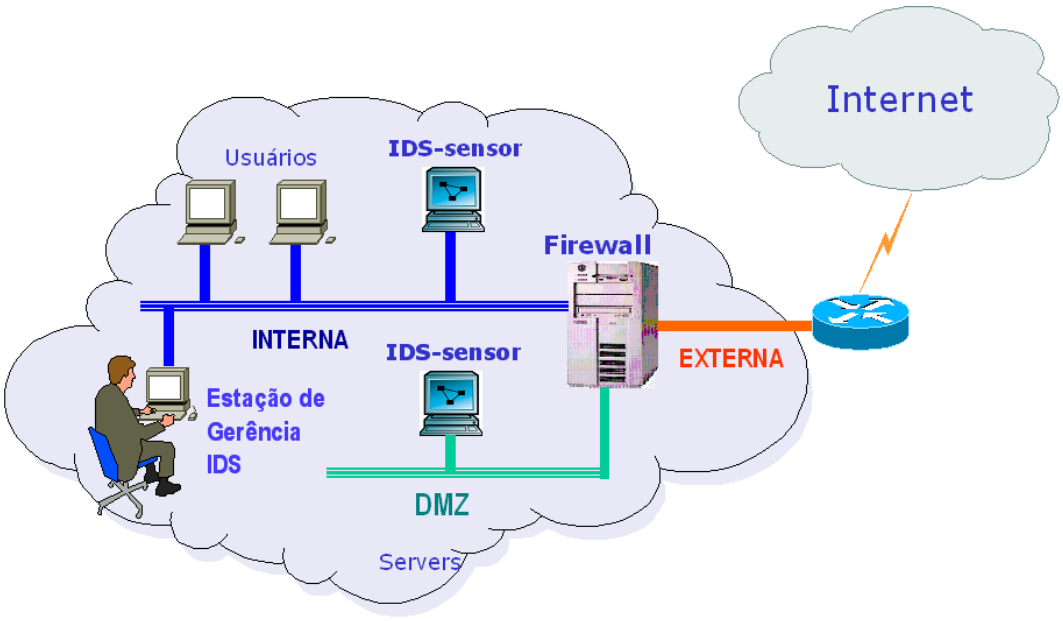


Figura 2.7: Sistema de Detecção de Intrusão Baseado em Rede

- Não necessitam de alterações em servidores ou quaisquer outras máquinas;
- Não afetam diretamente o sistema onde são instalados;
- Tendem a ser mais independentes, tipo "solução caixa-preta".

Principais desvantagens encontradas nos NIDS:

- Possuem visão localizada, monitoram um segmento de rede. Numa rede muito segmentada a quantidade de sensores será grande, isto se torna pior numa rede com *switches*;
- Inadequados para tratar ataques mais complexos;
- Grande quantidade de dados podem trafegar entre os agentes e estações de gerência;
- Não conseguem monitorar tráfego em sessões encriptadas.

Obs.: Muitos softwares comerciais já diminuiram estas desvantagens;

Os IDSs baseados em rede (NIDS), os quais geralmente utilizam o mecanismo de análise de assinaturas, constituem uma poderosa ferramenta, sendo portanto, o tipo de IDS mais utilizado. Na próxima sub-seção serão apresentados maiores detalhes sobre o NIDS.

O NIDS, juntamente com o *firewall*, são as duas ferramentas de segurança que compõem o modelo de gerência de segurança proposto nesta tese.

IDSs baseados em *host* :

Os IDSs baseados em *host* analisam sinais de intrusão na máquina nos quais estão instalados. Eles frequentemente usam os mecanismos de *log* do sistema operacional e estão muito ligados aos recursos do sistema. Eles agem procurando por atividades não usuais em: tentativas de *login*, acesso à arquivos, alterações em privilégios do sistema, etc.

Ex.: Se algum usuário usar o comando `su` do UNIX para tentar obter acesso como *root*, este evento será registrado, armazenando o nome do usuário que tentou, hora, etc.

Principais vantagens encontradas nos IDSs baseados em *host*:

- Detectam acessos com excesso de autoridade ou sem autoridade;
- Podem em várias circustâncias dizer exatamente o que o atacante fez;
- Geram menos falsos positivos(alarmes falsos) do que IDSs baseados em rede;
- Uso em ambientes onde largura de banda é crítica;
- Menor risco de fazer uma configuração errada;
- Mais difícil de serem enganados.

Principais desvantagens encontradas nos IDSs baseados em *host*:

- Dependem das capacidades do sistema no qual está instalado;
- Requerem instalação e possivelmente alteração no equipamento que se deseja monitorar;
- Necessitam de maior atenção por parte dos administradores;
- São relativamente mais caros;
- Possuem visão extremamente localizada, só monitoram uma máquina.

Verificadores de integridade de arquivos :

São programas que examinam os arquivos num computador e determinam se eles foram alterados desde a última vez que o verificador foi executado. Esses programas são baseados em funções de *hash*.

Função de *hash*: Processo matemático que reduz uma sequência de bytes para uma cadeia de *bits*, geralmente de comprimento fixo. Uma função de *hash* muito utilizada é a MD5.

O processo consiste em gerar um *hash* para cada aquivo e armazená-los numa tabela contendo o nome do arquivo e seu respectivo *hash*. Se após isso quaisquer dos arquivos que passaram por este processo forem alterados o seu *hash* será diferente e portanto, diferente daquele armazenado. Assim, o programa poderá alertar ao operador qual arquivo foi alterado. Existem os utilitários em UNIX chamados `md5` ou `md5sum` que aceitam como entrada um arquivo e imprimem o *hash* do mesmo.

Principais vantagens encontradas nos verificadores de integridade de arquivos:

- É computacionalmente impraticável "derrotar" a matemática por trás de um verificador de integridade, mas a força bruta pode ser tentada.
- São extremamente flexíveis, podem ser configurados para verificar quaisquer arquivos no sistema;
- Protegem quanto aos cavalos de tróia, *backdors* e apagamento de rastros.

Principais desvantagens encontradas nos verificadores de integridade de arquivos:

- Consomem muitos recursos do sistema: CPU, memória e disco;
- São vulneráveis à modificação pelo próprio atacante;
- Devem ser executados frequentemente, e com isso podendo reportar centenas de alterações se não forem configurados corretamente. Ex.: Se todos os arquivos do sistema forem verificados existirão centenas de alterações feitas pelo próprio sistema operacional ou usuários legítimos. Isso irá prejudicar muito a correta análise e auditoria do sistema.

2.3.2 NIDS e o Mecanismo de Análise de Assinaturas

O princípio de funcionamento do NIDS pode ser mostrado através do diagrama em blocos na Figura 2.8. Este diagrama foi criado no modelo CIDF (*Common Intrusion Detection Framework*) proposto por STANIFORD-CHEN [2].

Este modelo para projeto de um IDS é composto dos seguintes componentes:

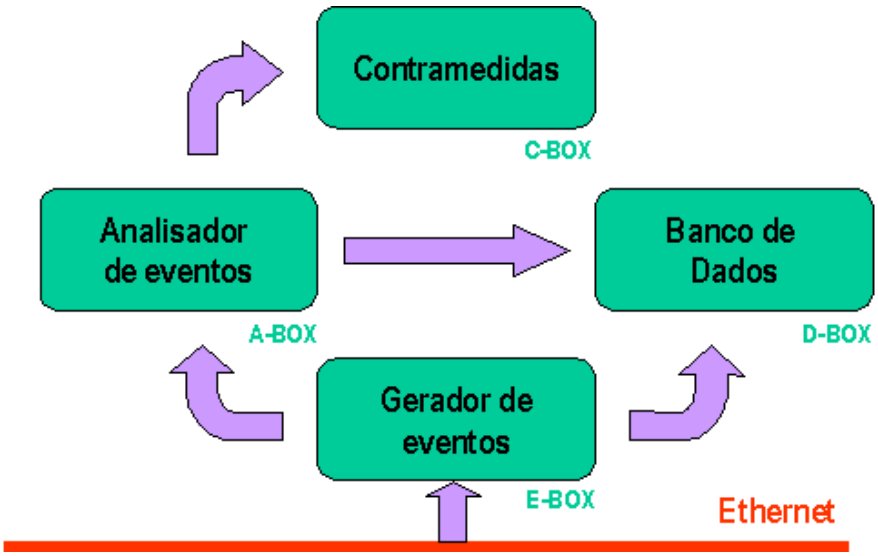


Figura 2.8: Modelo CIDF para o Projeto de um NIDS

E-Box : Gerador de eventos;

A-Box : Analisador de eventos;

D-Box : Banco de dados;

C-Box : Contramedidas.

O gerador de eventos captura todos os pacotes da rede (TCP, UDP, ICMP, etc.) e entrega ao analisador de eventos para que este aplique os critérios de identificação do ataque, estes dados são salvos num banco de dados. O banco de dados contém todos os pacotes detectados como ataques. Os dados também podem, se necessário, ser salvos automaticamente do gerador para o banco de dados. No momento da identificação, o analisador de eventos pode disparar uma unidade de contramedidas.

Gerador de eventos : Geralmente a interface é colocada em modo promíscuo. No ambiente UNIX pode-se ter acesso facilitado através da biblioteca `libpcap` [39] que pode decodificar *Ethernet*, *Token Ring*, FDDI, PPP, SLIP e RAW e posteriormente IP, TCP, UDP e ICMP.

Analisador de eventos : É o cérebro do IDS. É o componente responsável por identificar o que é e o que não é um ataque. A maioria dos fabricantes usam somente o mecanismo de análise de assinaturas, e alguns produtos estão apenas começando a utilizar análise estatística. Os sistemas adaptativos são assunto de pesquisas atuais.

Banco de dados : Pode receber dados tanto do gerador de eventos quanto do analisador de eventos e além disso deve ter desempenho compatível com a aplicação, sendo dimensionado de acordo com a capacidade da máquina e a capacidade do *link* a ser monitorado.

Contramedidas : É o bloco responsável por tomar ações baseadas nos eventos, deve ter a capacidade de se comunicar com estações de gerência, outros IDSs ou até um *firewall*.

O analisador de eventos é um dos blocos mais importantes, portanto, vamos examinar através do mecanismo de análise de assinaturas, como é feita a configuração deste componente.

O mecanismo de análise de assinaturas é nada mais do que uma lista contendo assinaturas de ataques e seus respectivos alertas a serem enviados. A assinatura do ataque é construída com base nas características do pacote que contém o ataque, estas características podem ser:

- Portas de origem e destino;
- Números de sequência;
- Flags dos protocolo TCP, ex.: SYN, FIN, etc.
- Outros campos dos protocolos IP[36], TCP[38], UDP[35], ICMP[37] e suas opções;
- Principalmente uma pequena parte do conteúdo da camada de aplicação do pacote.

O mecanismo de análise de assinaturas age como um anti-vírus, analisando cada pacote contra o conjunto de regras criado. Este conjunto de regras, formado pelas assinaturas e os seus respectivos alertas são geralmente armazenados em memória como uma lista encadeada, ou duplamente encadeada, a fim de agilizar ao máximo o processo de verificação dos pacotes.

O banco de dados das regras, que muita das vezes é um arquivo texto, deve ser atualizado frequentemente, assim como um anti-vírus.

Exemplos de regras para IDS podem ser encontradas no site do Snort[21] e ArachNIDS[22].

Dois exemplos de regras do Snort podem ser vistas abaixo:

```
alert tcp any any -> 10.1.1.0/24 80 (content: /cgi-bin/phf; msg:
‘‘Detectada tentativa de exploração de uma falha do PHF’’;)
```

```
alert tcp any any -> 10.1.1.13 143 (content: |E8 C0 FF FF
FF|/bin/sh msg: ‘‘Detectado buffer overflow IMAP’’;)
```

A primeira irá gerar um alerta de um pacote vindo de qualquer máquina com qualquer porta de origem para a sub-rede 10.10.1.0 (classe C, pois a máscara é 255.255.255.0) na porta 80 (HTTP) cujo pacote contenha a *string* “/cgi-bin/phf”, o alerta irá gerar uma mensagem do tipo: “Detectada tentativa de exploração de uma falha do PHF”, que é uma falha encontrada em algumas versões antigas do PHF quando instalado no diretório **cgi-bin** de um servidor web.

O segundo exemplo irá gerar um alerta de um pacote vindo de qualquer máquina com qualquer porta de origem para a máquina 10.1.1.13 na porta 143 (IMAP) cujo pacote contenha a *string* “|E8 C0 FF FF FF|/bin/sh”, o alerta irá gerar uma mensagem do tipo: “Detectado buffer overflow IMAP”, que é uma falha encontrada em algumas versões antigas do servidor de e-mail IMAP, que proporciona ao *cracker* se utilizar de um estouro de *buffer* no programa e executar comandos remotamente na máquina.

Neste segundo exemplo, os números hexadecimais na *string* do campo **content** são realmente os bytes que se desejam procurar, na ordem apresentada, para isso eles devem ser colocados entre os caracteres “|”.

2.3.3 IDSs gratuitos e comerciais

A fim de criar um modelo genérico para o gerenciamento de um IDS, foram estudados alguns dos softwares mais utilizados no mercado. Na Tabela 2.1 são apresentamos alguns modelos de IDSs dentre os quais estão alguns dos que foram estudados mais profundamente.

É importante lembrar que o modelo proposto nesta tese para o gerenciamento de IDSs foi baseado apenas nos tipos baseados em rede. O NFR, o ISS RealSecure e principalmente o Snort foram os sistemas mais estudados.

NFR : O NFR [40] Possui uma estação central que gerencia os sensores IDS, chama-

Produto	Fabricante	Site	Tipo
Intruder Alert	Axent	www.axent.com	*Host*
Net Prowler	Axent	www.axent.com	Rede
RealSecure	ISS	www.iss.net	*Host*/Rede
NetRanger	Cisco	www.cisco.com	Rede
NFR	NFR	www.nfr.net	Rede
SessionWall	Computer Asscociates	www.ca.com	Rede
Snort (free)	Marty Roesch	www.snort.org	Rede
Abacus (free)	Psionic	www.psionic.com	*Host*

Tabela 2.1: IDSs Gratuitos e Comerciais

dos de IDA (*Intrusion Detection Appliances*), através da estação pode se configurar os sensores e processar os alertas.

Os sensores são baseados no sistema operacional OpenBSD, mas com o *kernel* e o resto do sistema bastante modificado, não há como administrar ou entrar no sistema a não ser pela estação central. A comunicação entre sensores e estação central é criptografada.

Um fato interessante é que o software é executado do CD-ROM e o HD é praticamente todo utilizado para o banco de dados de *logs* do sistema.

A interface de gerência não é das melhores, mas também não é muito confusa, ela pode ser instalada em qualquer máquina Windows e não interfere de forma alguma no restante do sistema.

ISS RealSecure : O RealSecure [41] que é parte integrante da família de produtos SAFEsuite, possui de forma integrada um sistema de detecção baseado em rede e em host.

O sistema possui características bastante interessantes, tais como: alerta por email, alerta por pager, reconfiguração de *firewalls* e programação de ações definidas pelo usuário. Além disso, o sensor RealSecure pode enviar um alarme para a estação de gerência (*RealSecure Manager*) ou para um console de gerenciamento de outro fabricante.

Snort : A primeira característica notável do Snort [21] é sua qualidade, apesar de ser um software totalmente gratuito. O Snort possui uma vasta gama de usuários e várias características encontradas em produtos comerciais, a saber:

- Possui muitas opções para configuração de regras;
- Detecta *buffer overflows*, DoS, varredura de portas, ataques de CGI, etc;
- Pode ser utilizado como um analisador de protocolo simples;
- Possui vários mecanismos de *log* e alerta;
- É bastante extensível através de plugins.

As desvantagens do Snort estão ligadas a sua forma de gerenciamento, muitos destes problemas são resolvidos através de softwares gratuitos de terceiros, tal como o ACID [23], ou comerciais como o Demarc [24], que proporcionam uma central de gerência para os sensores Snort. Este gerenciamento envolve uma boa experiência do usuário com sistemas UNIX a fim de instalar e configurar o sistema completo. Depois de corretamente instalado e configurado o sistema deixa muito pouco a dever para produtos comerciais, em alguns casos até supera.

2.4 Tecnologias que Integram e Gerenciam Ferramentas de Segurança

Nesta seção apresentaremos algumas soluções comerciais e propostas de padronização para integração e padronização de ferramentas de segurança. Estes sistemas são conhecidas como ISMS (*Integrated Security Management Systems*).

A utilização de ISMS é importante devido a diversos problemas que se verificam:

- Aumento da complexidade dos produtos de segurança;
- Diversidade de políticas de segurança para sistemas heterogêneos em grandes redes;
- Aumento de risco resultantes de erros humanos;
- Necessidade de respostas automáticas e imediatas para vários tipos de ameaças;
- Necessidade de unificar interfaces para facilitar o gerenciamento;
- Integrar funcionalidades de IDSs e *firewalls*.

Na Figura 2.9 mostramos uma topologia física de rede onde existe a interoperabilidade entre IDSs e *firewalls*. Em seguida, analisaremos algumas propostas de ISMS.

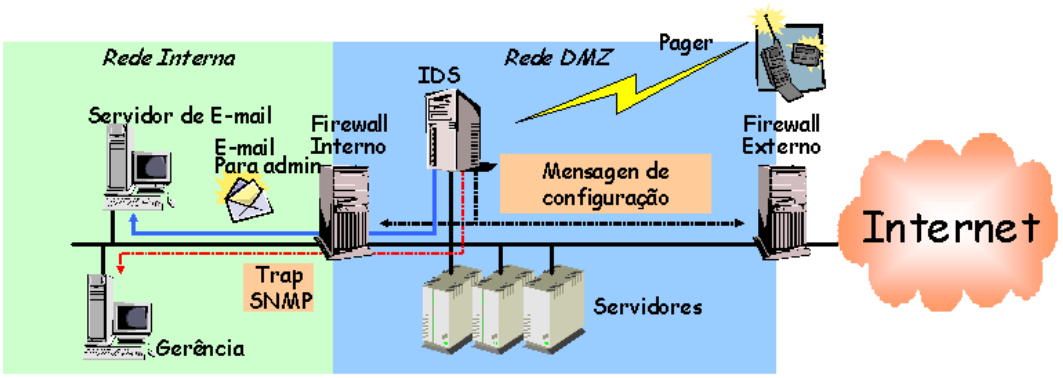


Figura 2.9: Diagrama de Rede com Interoperabilidade entre *Firewall* e IDS. Fonte: *Integrated Security Management, Real-Time Systems Laboratory, 2000*

OPSEC : A OPSEC [30] (*Open Platform for Secure and Enterprise Connection*) é o padrão da *Check Point* (fabricante do Firewall-1) para a integração de produtos de segurança. Ela provê interoperabilidade para diversos produtos, tais como: Firewall-1, VPN-1, HP Openview, Norton Anti-vírus, ISS RealSecure, Tivoli, etc. Apesar de parecer um padrão genérico, a OPSEC permite, na verdade, que outros fabricantes desenvolvam produtos que sejam compatíveis com os produtos da Check Point.

Active Security : O *Active Security* [42] é uma plataforma com o mesmo propósito da OPSEC, só que desenvolvida pela NAI (*Network Associations Inc.*). Através de uma estação central chamada *Event Orchestra* ela é capaz de gerenciar o *Gauntlet Firewall*, o *Cybercop Scanner* e outros produtos da NAI ou compatíveis com eles.

CDSA : O CDSA [4] (*Common Data Security Architecture*) é de fato um modelo realmente genérico, aberto e extensível. A especificação foi adotada pelo *Open Group* em 1997 e foi baseada em códigos da Intel, largamente revistos pela indústria.

Apesar das aparentes vantagens do CDSA, sendo um padrão proposto por uma organização neutra, ele possui uma complexidade e quantidade de detalhes

extremamente grande, o que fez com que poucos fabricantes se interesassem em seguí-lo.

SIGSEC : O SISGEC (Sistema Integrado de Gerenciamento de Segurança Empregando CORBA) é o ISMS proposto e documentado nesta tese.

Capítulo 3

O Padrão CORBA

Sistemas distribuídos tem estado em uso de uma forma ou de outra há muito tempo, apesar deles não terem sido chamados desta forma e certamente não possuirem a flexibilidade que possuem hoje.

Para definir melhor o que são sistemas distribuídos e como CORBA se encaixa neste contexto, vamos examinar um pouco da sua história e evolução, a partir de conceitos básicos como o *mainframe*.

Posteriormente examinaremos o padrão CORBA destacando as principais facilidades e serviços providos e também apresentaremos algumas implementações do padrão, com algumas comparações.

Finalmente é apresentado o uso de CORBA na proposta defendida nesta tese, destacando os motivos da escolha do padrão e das diversas opções dentro do mesmo.

3.1 O Uso de Sistemas Distribuídos

Começaremos com um pouco de história, mostrando a evolução do uso de sistemas monolíticos, até o paradigma cliente/servidor e finalmente nos sistemas distribuídos.

Sistemas Monolíticos e Mainframes: Consistiam de um sistema hierárquico de banco de dados e terminais sem processamento, também conhecidos como "terminais burros". Os *mainframes* davam muito trabalho de manutenção mas podiam atender um grande número de usuários e tinham a vantagem (ou desvantagem, dependendo do ponto de vista) de possuirem um gerenciamento centralizado.

Sistemas de software escritos para *mainframes* eram frequentemente monolíticos, isto é, a interface com usuário, lógica principal e o acesso aos

dados eram todos contidos num grande aplicativo, e os terminais acessavam este aplicativo que era executado totalmente no *mainframe*, vide Figura 3.1.

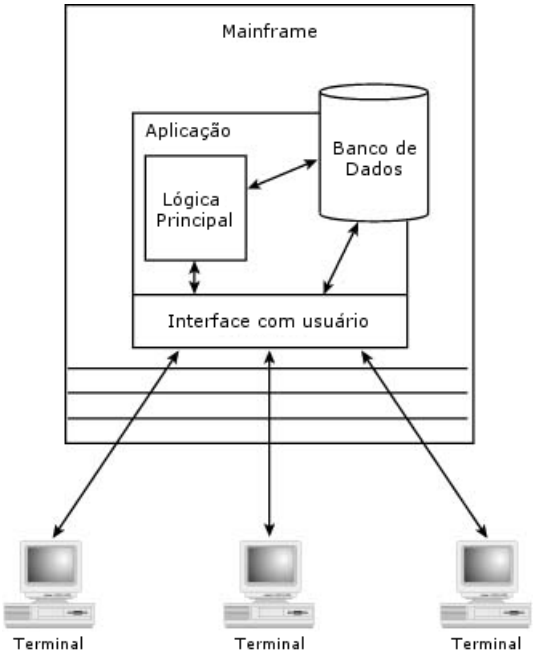


Figura 3.1: Sistemas Monolíticos e Mainframe

Arquitetura Cliente/Servidor: Com a disseminação dos PCs foi possível que algum processamento fosse feito nos computadores dos usuários, permitindo que estes pudessem acessar somente o que fosse necessário em um servidor, e ainda poderiam executar outras aplicações de forma independente. Nesta mesma época, a arquitetura UNIX estava se consolidando e foi a escolha óbvia em termos de custo e desempenho para utilização em servidores.

As aplicações cliente/servidor (Figura 3.2) tipicamente distribuiam os componentes do sistema tal que o banco de dados residia em um servidor (geralmente uma máquina UNIX), a interface com o usuário no cliente, e a lógica principal em um ou ambos cliente e servidor.

Arquitetura Cliente/Servidor Multicamadas: p A solução cliente/servidor (Figura 3.2) foi, em muitas maneiras, a revolução do modo antigo de se fazer as coisas. Entretando tal solução ainda possuia fragilidades. Alterações em componentes no sistema poderiam demandar a distribuição de um novo aplicativo cliente para todos os usuários do sistema.

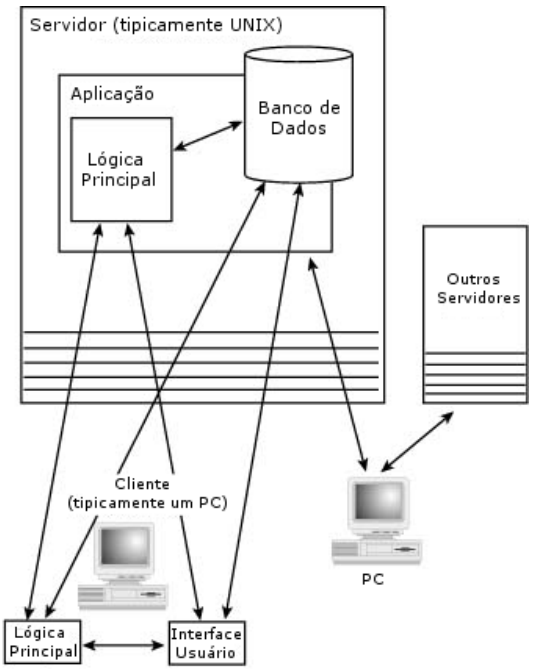


Figura 3.2: Arquitetura Cliente/Servidor

Os problemas com a tradicional arquitetura cliente/servidor (frequentemente chamada cliente/servidor de duas camadas) foram resolvidas em parte pela arquitetura multicamadas (Figura 3.3). Conceitualmente a aplicação poderia possuir qualquer número de camadas, mas a mais popular era aquela que particionava o sistema em três camadas lógicas: interface com usuário, lógica principal e o acesso ao banco de dados.

Sistema Distribuídos: O próximo passo lógico na evolução das arquiteturas dos softwares é o modelo de sistema distribuído. Esta arquitetura extrapola o conceito da arquitetura multicamadas. Ao invés de diferenciar entre acesso a dados e interface com usuário por exemplo, o modelo de sistema distribuído simplesmente expõe toda a funcionalidade da aplicação como objetos, cada um do qual pode usar o serviço provido por outro objeto (Figura 3.4).

A arquitetura de sistemas distribuídos pode quebrar a distinção entre cliente e servidor pois cada componente (objeto) no sistema pode se comportar como cliente e servidor. Esta arquitetura provê o máximo em flexibilidade.

A arquitetura de sistemas distribuídos encoraja, e de certa forma obriga a definição clara de interfaces entre os seus componentes. A interface de um componente especifica para os outros componentes quais serviços são oferecidos

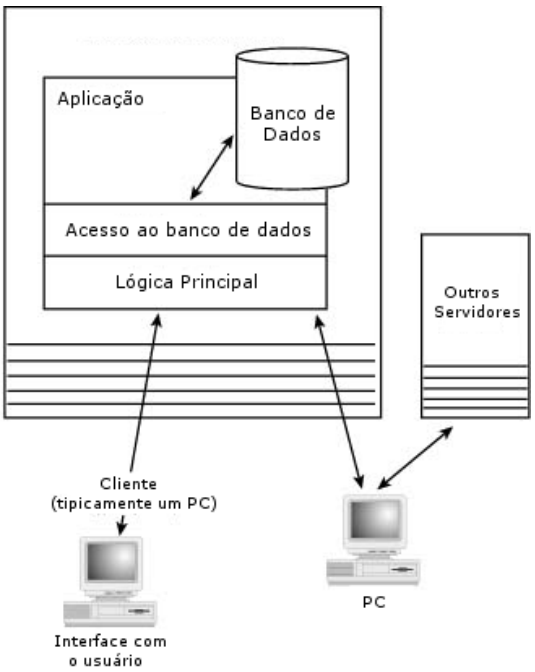


Figura 3.3: Arquitetura Cliente/Servidor Multicamadas

por ele e como são usados. Como a interface uma vez especificada não deve mudar, a alteração na implementação de um componente (desde que correta) não afeta os outros componentes do sistema.

Nós percebemos anteriormente, a evolução de uma arquitetura monolítica centralizada para uma arquitetura distribuída, altamente descentralizada, sem mencionar os avanços que tem sido feitos em CORBA [18].

Na prática, como então é feita a comunicação entre os diversos componentes em um sistema distribuído? E quais são as regras, para que aplicações desenvolvidadas neste modelo possam interagir? É aqui que entra o padrão CORBA [18], mas antes de analisá-lo (o que será feito na seção seguinte), vamos apresentar algumas outras tecnologias que permitem desenvolvermos sistemas distribuídos.

Programação com *sockets*: Em muitos sistemas modernos, a comunicação entre máquinas e entre processos na mesma máquina é feita através do uso de *sockets*. Um *socket* é um canal através do qual aplicativos podem se conectar com outros e se comunicar. O programador implementa funções que escrevem dados nos *sockets* ou lêem dados dos *sockets*.

A API (*Application Programming Interface*) para *sockets* é de muito baixo nível, portanto, a aplicação pode ser extremamente eficiente. Por outro lado,

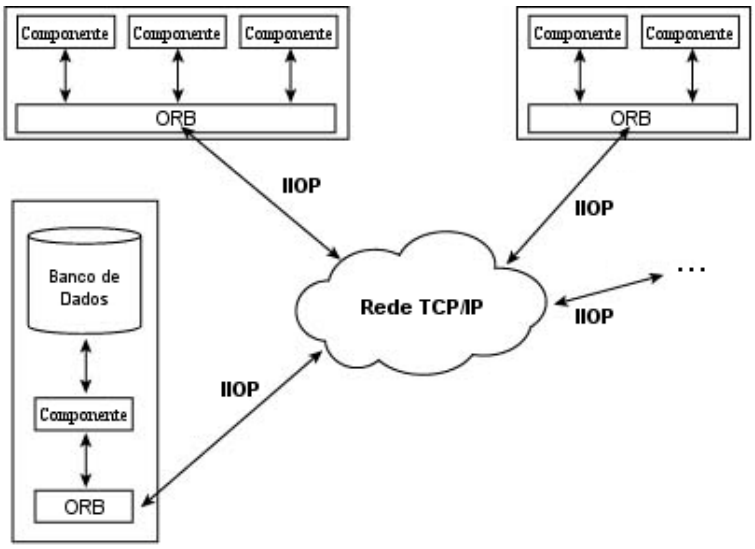


Figura 3.4: CORBA: Exemplo de Arquitetura de Sistemas Distribuídos

a programação pode se tornar complexa, devido a dificuldade de se trabalhar com diferentes tipos de dados em arquiteturas de hardware diferentes. É extremamente difícil, senão impossível, definir interfaces coerentes entre os diversos componentes do sistema em máquinas distintas em aplicações mais complexas.

RPC (*Remote Procedure Call*): Usando RPC, ao invés do programador se preocupar e manipular diretamente a conexão através de *sockets*, ele escreve funções, tal como aquelas na linguagem C e gera um código automaticamente que cuida de todo o processo de comunicação usando *sockets* para executar as funções desejadas em outras máquinas.

Isto provê uma interface para o programador orientada a função, muito mais fácil do que se programar em *sockets* diretamente. Apesar de existirem várias implemetações de RPC incompatíveis, existe um protocolo RPC padrão que é disponível para muitas plataformas.

Microsoft DCOM (*Distributed Component Object Model*): O DCOM da Microsoft oferece capacidades similares ao CORBA, ele é um modelo robusto que possui bom suporte nos sistemas operacionais da Microsoft pois já é integrado nestes sistemas. Entretanto, sendo uma tecnologia Microsoft sua disponibilidade é muito fraca fora dos sistemas operacionais Windows.

A Microsoft têm trabalhado para corrigir esta disparidade, mas deixou claro

em muitas ocasiões que o DCOM é melhor suportado nos ambientes Windows.

Um interessante desenvolvimento é a disponibilidade de pontes CORBA-DCOM que possibilitariam componentes dos dois padrões se comunicarem, mas isto não é uma solução perfeita, deve ser usado apenas em situações extremas onde é realmente necessário, e não em um projeto que está sendo criado, até porquê, existem muitas implementações de CORBA para o ambiente Windows.

Java RMI (*Remote Method Invocation*): Java RMI é uma alternativa muito próxima a CORBA. Uma desvantagem é que RMI é uma solução exclusiva para Java, o que significa que clientes e servidores RMI devem ser escritos em Java. Para aplicações completas em Java a RMI pode ser uma boa escolha, mas se há uma chance destas aplicações interagirem com outras escritas em outra linguagem, CORBA é uma escolha bem melhor. Felizmente, implementações completas de CORBA existem para Java.

Todas as tecnologias vistas possuem suas vantagens e desvantagens, veremos que o padrão CORBA, por possuir uma forte organização suportada por membros da indústria altamente qualificados, se tornou bastante robusto e flexível, possuindo ínumeras vantagens as quais serão observadas na seção seguinte.

3.2 Arquitetura, Facilidades e Serviços

Linguagens de programação modernas empregam o paradigma de orientação a objetos para desenvolver aplicações dentro de um único processo de um sistema operacional. Ex: Um programa simples, com uma ou mais classes em C++.

O próximo passo é distribuir o sistema em diversos processos em diferentes máquinas pela rede. Devido ao fato da orientação a objetos ter provado ser a forma mais adequada de desenvolver e manter aplicações em larga escala, parece razoável aplicar o paradigma de orientação a objetos para um sistema distribuído. Assim, objetos são distribuídos entre as máquinas em um ambiente de rede e se comunicam uns com os outros.

De fato, computadores dentro de um ambiente de rede diferem em arquitetura de hardware, sistema operacional, e linguagens de programação usadas para implementar os objetos, isto é o que chamamos de ambiente distribuído heterogêneo. Para

permitir a comunicação entre os objetos num ambiente deste tipo é necessário um componente de software chamado de plataforma de *middleware*. O termo *middleware* deriva do fato de que este componente de software reside entre a aplicação e o sistema operacional.

O padrão CORBA (*Common Object Request Broker Architeture*) é a especificação de tal plataforma de *middleware* pela OMG (*Object Management Group*). É importante deixar bem claro que CORBA não é uma implementação, e sim um padrão que deve ser (e é) seguido pelos desenvolvedores (ver Seção 3.3).

O padrão CORBA (atualmente na versão 2.6) possui as seguintes características:

Orientação a objeto: Objetos são os blocos básicos de construção de aplicações em CORBA.

Trasparência: Um objeto que deseja invocar uma determinada operação em outro objeto utiliza os mesmos mecanismos e sintaxe se este está no mesmo espaço de endereçamento, na mesma máquina ou em outra máquina.

Independência de hardware, sistema operacional e linguagem: Componentes CORBA podem ser implementados usando diferentes linguagens de programação, diferentes arquiteturas de hardware e em diferentes sistemas operacionais.

Independência de fabricante: Implementações de CORBA de diferentes fabricantes interoperam.

O CORBA é um padrão no sentido de que, qualquer um pode obter a especificação e implementá-lo. Apesar de todas as suas características técnicas, esta é considerada uma das principais vantagens de CORBA sobre as outras soluções proprietárias.

3.2.1 Conceitos e Terminologia

Vamos definir alguns dos principais componentes da arquitetura CORBA.

ORB (*Object Request Broker*): É o principal componente da arquitetura, o qual serve como um barramento, conectando os diversos objetos à rede. No lado do cliente, o ORB oferece uma interface para invocar uma operação, enquanto que no lado do servidor o ORB oferece uma API para entregar

esta invocação. É tarefa do ORB localizar o servidor apropriado e entregar a invocação da operação via o adaptador de objetos (OA).

OA (*Object Adapter*): O propósito do adaptador de objetos é despachar invocações das operações e suportar o ciclo de vida dos objetos no servidor, isto é, a criação e destruição dos objetos. Antigas versões do CORBA incluiam o BOA (*Basic Object Adapter*), o qual por ser muito limitado, permitia que diversos fabricantes incluissem modificações proprietárias que dificultavam a portabilidade. Desde a versão 2.2 do padrão CORBA, o BOA foi substituído pelo POA (*Portable Object Adapter*). Todas as implementações atuais de CORBA suportam ambos BOA e POA.

IDL (*Interface Definition Language*): A IDL é o ponto de partida para o desenvolvimento de aplicações em CORBA, a IDL é uma linguagem de definição, não uma linguagem de programação. Seu objetivo é definir de forma única e padronizada interfaces para objetos. Uma interface define um protocolo permitindo que componentes de software (objetos) possam se comunicar de forma padronizada. Vamos examinar a IDL com mais detalhes na seção seguinte pois ela tem importância fundamental na proposta desta tese.

***Stubs* e *Skeletons*:** Os termos acima não serão traduzidos devido ao significado não exprimir corretamente a sua função dentro da arquitetura CORBA. Como estas mesmas denominações são sempre utilizadas então os termos serão mantidos. Os *stubs* e *skeletons* são códigos gerados automaticamente após a IDL de um determinado sistema ser compilada com o compilador IDL. Os *stubs* e *skeletons* irão fazer parte da aplicação em desenvolvimento e deverão ser "linkados" junto com as implementações dos objetos e o programa principal cliente e servidor, conforme o caso.

O *stub* é o código que é responsável por tratar as invocações de clientes e o *skeleton* é o responsável por prover uma classe para que o servidor possa derivar e implementar os objetos. O programador não trabalha diretamente com *stubs* ou *skeletons*, ele apenas escreve a IDL e a compila, posteriormente escreve a implementação dos objetos e os programas cliente e servidor, mas sempre tarabalhando com objetos em alto nível e nunca com detalhes de *sockets* nem específicos de arquiteturas.

Existem diversos outros componentes importantes do padrão CORBA, tais como o repositório de interfaces, repositório de implementações, interface de invocação dinâmica, etc. Estas características não serão examinadas aqui pois foge ao escopo da tese apresentar o padrão CORBA completo. Por outro lado, apresentamos as características básicas e aquelas que são utilizadas no desenvolviemento do trabalho.

3.2.2 A IDL (*Interface Definition Language*)

A OMG IDL é o mecanismo fundamental de abstração do padrão CORBA para separar interfaces de objetos de suas implementações.

A OMG IDL estabelece um "contrato" entre cliente e servidor que descreve os tipos de dados e as interfaces utilizadas por uma aplicação. Esta descrição é independente da linguagem de programação a ser utilizada, então não é necessário que o cliente seja escrito na mesma liguagem de programação que o servidor.

As definições da IDL são compiladas para uma linguagem de programação em particular por um compilador IDL. Este compilador traduz a linguagem idependente da IDL para a linguagem específica. A forma de tradução da IDL para diferentes linguagens de programação são especificadas pelo padrão CORBA e são conhecidas como *language mappings*. Atualmente CORBA define *language mappings* para C, C++, SmallTalk, COBOL, Ada e Java. Esforços independentes existem para prover *language mappings* adicionais para Eiffel, Modula 3, Lisp, Perl, Tcl, Python, Dylan, Oberon, Visual Basic e Objective-C. Alguns destes *mappings* podem eventualmente se tornar padrões.

Pelo fato da IDL descrever interfaces, mas não implementações, ela é uma linguagem puramente declarativa.

Compilação:

Um compilador IDL produz código fonte que deve ser combinado com o código da aplicação para produzir programas executáveis do tipo cliente e/ou servidor.

O código fonte produzido é dividido nas denominações *stub* (utilizadas para programar clientes) e *skeletons* (utilizadas para programar servidores) vistas anteriormente e podem ser obtidas em arquivos separados ou não, isto vai depender do ORB (implementação de CORBA) utilizado.

Ex.: Um aquivo IDL chamado: `conta.idl`

```
interface Conta {
    void deposita(in unsigned long quantia);
```

```
    void retira(in unsigned long quantia);
    unsigned long saldo();
};
```

A IDL mostrada representa a interface de uma conta bancária onde se pode fazer as operações: deposita, retira e saldo. Ao ser compilada com o Mico [14] (implementação de CORBA gratuita) esta IDL irá gerar os arquivos: `conta.cc` e `conta.h`, que são arquivos com código fonte em C++ (única linguagem suportada pelo Mico) e que possuem os *stubs* e *skeletons*. Estes arquivos devem ser compilados junto com os programas desenvolvidos com esta interface.

Tipos de dados da IDL:

A Tabela 3.1 mostra os tipos básicos da IDL.

Tipo	Faixa	Tamanho
short	-2^{15} até $2^{15} - 1$	16 bits
long	-2^{31} até $2^{31} - 1$	32 bits
unsigned short	0 até $2^{16} - 1$	16 bits
unsigned long	0 até $2^{32} - 1$	32 bits
float	IEEE precisão simples	32 bits
double	IEEE precisão dupla	64 bits
char	ISO Latin-1	8 bits
string	ISO Latin-1, exceto ASCII NULL	variável
boolean	VERDADEIRO ou FALSO	não especificado
octet	0 - 255	8 bits
any	Tipo arbitrário identificado em tempo real	variável

Tabela 3.1: Tipos Básicos da IDL

Outros tipos podem ser definidos pelo usuário, e são muito parecidos com os tipos de dados do C++:

- `typedef`;
- `enum`;
- `struct`;
- `union`;
- `array` e `sequence` (tipo especial de *array*).

Os tipos mais importantes cujos nomes diferem do C++ mas possuem uma correspondência direta são:

`module` : O mesmo funcionamento do `namespace` em C++.

`interface` : O mesmo funcionamento de `class` em C++.

Com estes tipos de dados é possível então definir qualquer interface usando a IDL. As operações das interfaces (que correspondem aos métodos de uma classe em C++) possuem três tipos de especificadores adicionais para os seus argumentos:

`in` : O argumento é passado do cliente para o servidor e seu valor não é alterado.

`out` : O argumento é passado do servidor para o cliente (tal como o valor de retorno).

`inout` : O argumento é passado do cliente para o servidor e pode ser alterado, retornando o valor para o cliente.

O compilador IDL se encarregará de traduzir estes especificadores para o formato de passagem de parâmetro apropriado.

3.2.3 CORBA*services* e CORBA*facilities*

CORBA*services*:

A OMA (*Object Management Architecture*) da qual CORBA faz parte, define um certo número de serviços que são úteis para aplicações em geral. Estes serviços vão desde o indispensável *Naming Service* até serviços de altíssimo nível, tal como o *Transaction Service*.

Tal como as outras especificações (incluindo CORBA), a OMG não define quaisquer tipos de implementações para estes serviços, mas provê as interfaces pelas quais os serviços são oferecidos. É função dos fabricantes de produtos CORBA prover estas implementações.

Listamos em seguida os CORBA*services*. Entraremos em detalhes somente no *Naming Service* que é o serviço que utilizamos na proposta desta tese.

- *Concurrency Control Service*;
- *Event Service*;
- *Externalization Service*;

- *Licensing Service*;
- *Life Cycle Service*;
- *Naming Service*;
- *Object Trader Service*;
- *Persistent Object Service*;
- *Property Service*;
- *Query Service*;
- *Relationship Service*;
- *Security Service*;
- *Time Service*;
- *Transaction Service.*

CORBA*facilities*:

As CORBA*facilities* são divididas em *horizontal facilities* (características úteis para todos os tipos de aplicações em CORBA para várias insdústrias) e *vertical facilities* (funcionalidade que é especificamente útil para aplicações dentro de um mercado em particular).

As *Horizontal facilities* incluem interface de usuário e gerenciamento do sistema. Estas funcionalidades são úteis para muitas aplicações, sem importar o segmento em que são utilizadas.

Tal como os CORBA*services*, a OMG apenas especifica as interfaces para estas facilidades. As implementações, quando aplicáveis, são providas pelos fabricantes de produtos CORBA.

Algumas CORBA*facilities* apenas sugerem interfaces para serem usadas por serviços específicos ou tipos de aplicações.

3.2.4 *Naming Service*

O OMG *Naming Service* é o mais simples e o mais básico dos serviços CORBA. Ele provê um mapeamento de nomes para referências de objetos: dado um nome,

o serviço retorna a referência do objeto armazenada sob este nome. Este funcionamento é similar ao Internet *Domain Name Service* (DNS), o qual traduz nomes de domínio da Internet (tal como www.sigsec.org) para endereços IP (tal como 146.164.32.70). Ambos OMG *Naming Service* e o DNS implementam um simples mapeamento, tal como numa agenda telefônica.

Existem três conceitos que precisam ficar claros e são fundamentais no *Naming Service*:

***name binding*:** É uma associação nome-referência. Um servidor registra um objeto no *Naming Service* com um determinado nome, assim, uma outra aplicação (cliente ou servidor) pode obter uma referência (localização) deste objeto fazendo uma consulta no *Naming Service*.

A mesma referência de um objeto pode ser armazenada várias vezes sob diferentes nomes, mas cada nome deve identificar somente uma referência.

***naming context*:** Um contexto do servidor de nomes é um objeto que armazena *name bindings*. Em outras palavras, cada contexto é como se fosse uma tabela que contém mapeamentos de nome-referência ou outros contextos. Esta estrutura é similar a um sistema de arquivos, onde os contextos seriam os diretórios e os arquivos seriam as associações nome-referência. Isto significa que contextos podem ser conectados para formar hierarquias.

É importante notar que os contextos possuem referências pois eles foram registrados por aplicações, e é necessário sempre obter a referência para um determinado contexto quando se quer saber quais os objetos que estão registrados sob ele.

***naming graph*:** Um grafo do *Naming Service* é uma hierarquia de contextos (*naming contexts*) e associações nome-referência (*name bindings*). A Figura 3.5 mostra um exemplo.

Os nós escuros representam contextos e os claros representam objetos, as setas representam as referências, tanto para objetos como para contextos.

O *Naming Service* possui uma interface muito bem definida (através de uma IDL) na qual pode se fazer operações, tais como:

- Registrar um objeto;

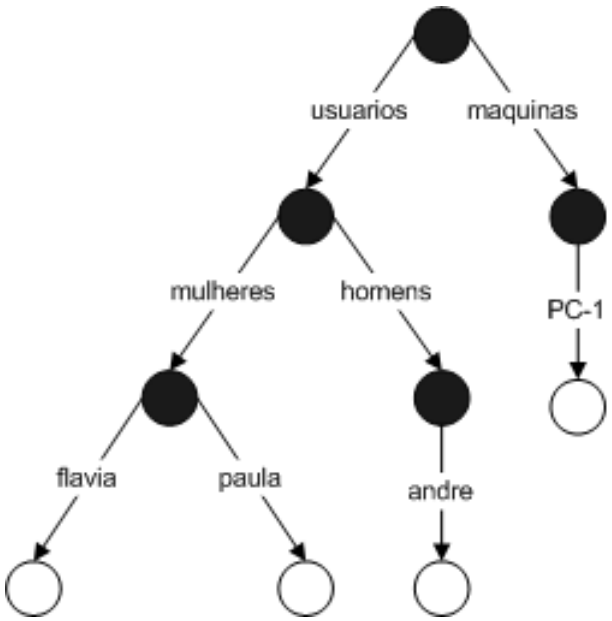


Figura 3.5: Exemplo de um Grafo do *Naming Service*

- Criar um contexto;
- Obter a referência de um objeto ou contexto;
- Apagar uma referência ou contexto;
- Listar as referências em um contexto.

No modelo de gerência proposto nesta tese o *Naming Service* é bastante utilizado e é uma peça fundamental para o correto funcionamento do sistema.

3.3 Implementações do Padrão CORBA

Existem muitas implementações do padrão CORBA, algumas gratuitas outras comerciais. Na Tabela 3.2 listamos algumas implementações, seus respectivos fabricantes e *sites* na Internet.

Atualmente podem ser encontradas mais de 50 implementações do padrão, em diferentes estágios de desenvolvimento, isto é, suportam determinados sistemas operacionais, possuem determinados *language mappings* e serviços.

Dentre as principais implementações, vistas na Tabela 3.2, escolhemos para o projeto desta tese a implementação chamada Mico, por apresentar um bom balanceamento entre qualidade, velocidade e recursos disponíveis, além de ser totalmente gratuita.

ORB	Fabricante	Site
Mico	gratuito	www.mico.org
Orbacus	IONA	www.ooc.com/ob
OmniORB	gratuito	omniorb.sourceforge.net
Orbix	IONA	www.iona.com
Visibroker	Borland	www.borland.com
TAO	gratuito	www.cs.wustl.edu/~schmidt/TAO
Orbit	gratuito	www.labs.redhat.com/orbit
JacORB	gratuito	www.jacorb.org

Tabela 3.2: Algumas Implementações do Padrão CORBA

Vejamos, como exemplo, as principais características desta implementação:

- C++ *mapping*;
- *Dynamic Invocation Interface* (DII);
- *Dynamic Skeleton Interface* (DSI);
- Repositório de interfaces (*Interface Repository*, IR);
- Navegador gráfico para o repositório de interfaces;
- IIOP como protocolo nativo (ORB preparado para suporte a multiprotocolos);
- *Portable Object Adapter* (POA);
- *Objects by Value* (OBV);
- *CORBA Components* (CCM);
- Suporte para utilizar MICO dentro de aplicações X WindoswS (Xt, Qt, Gtk e Tcl/Tk);
- *Dynamic Any*;
- *Interceptors*;
- Suporte para comunicação segura e autenticação utilizando SSL;
- Suporte para invocações de métodos aninhados;
- O tipo `any` oferece uma interface para inserir e extrair tipos construídos que não foram conhecidos em tempo de compilação;

- Implementação completa do BOA (adaptador de objetos básico), incluindo todos os modos de ativação, suporte para migração de objetos e repositório de implementação;
- O BOA pode carregar implementações de objetos em clientes em tempo real, utilizando módulos carregáveis;
- CORBA*services*:
  - *Naming Service*;
  - *Trading Service*;
  - *Event Service*;
  - *Relationship Service*;
  - *Property Service*;
  - *Time Service*.

Ao fazer a escolha de um determinado ORB é importante atentar para diversos fatores, de forma que o ORB se encaixe da melhor forma nos padrões da empresa ou organização. Estes fatores são:

- Linguagens de programação suportadas;
- Plataformas suportadas (sistema operacional e arquitetura);
- Compiladores suportados;
- Serviços CORBA disponíveis;
- Facilidades CORBA disponíveis;
- Velocidade;
- Custo;
- Suporte técnico.

Na Tabela 3.3 mostramos alguns testes de velocidade feitos com alguns ORBs. Este testes foram providos pelo *Open CORBA Benchmarking* [19], um projeto que realiza vários testes em diferentes ORBs.

O tempo de resposta apresentado neste teste é definido como tempo de invocação, realizado através de uma operação `void ping()` sem argumentos. Outras métricas definidas por TUMA e BUBLE [66] são também utilizadas nos testes do *Open CORBA Benchmarking* [19].

ORB	Plataforma	void Ping ()
Mico 2.3	Linux i386 (1x896MHz CPU)	196 µsec
Orbacus 4.1	Linux i386 (1x896MHz CPU)	245 µsec
Orbacus 4.0	Windows NT/2K i386 (2x797MHz CPU)	207 µsec
Orbix 2000 (1.2)	Linux i386 (1x896MHz CPU)	399 µsec
Visibroker 4.5	Linux i386 (2x798MHz CPU)	227 µsec
Visibroker 4.5	Windows NT/2K i386 (2x797MHz CPU)	122 µsec
OmniORB 3.0	Linux i386 (2x798MHz CPU)	174 µsec
TAO 1.1	Linux i386 (2x798MHz CPU)	193 µsec

Tabela 3.3: Teste de Velocidade de Alguns ORBs

Podemos verificar que as implementações gratuitas não deixam nada a dever em relação as comerciais neste teste.

3.4 Como e por que CORBA foi Utilizado

O padrão CORBA foi utilizado nesta tese por prover uma forma clara e bem definida das interfaces de comunicação utilizando a IDL. Assim, é possível padronizar as operações e tipos de dados da interface das ferramentas de segurança através de um modelo conhecido e bem fundamentado.

Outro motivo muito forte da utilização de CORBA foi a necessidade de se programar aplicações baseadas nas interfaces que pudessem ser facilmente desenvolvidas em vários sistemas operacionais e arquiteturas, pois estas aplicações (que chamaremos de *drivers*) devem traduzir os dados de entrada e saída das ferramentas de segurança (que existem hoje para as mais diferentes plataformas) para o formato padronizado pela IDL.

O uso do padrão CORBA no desenvolvimento do modelo proposto nesta tese foi moderado. Podemos citar algumas características importantes que foram utilizadas e outras que não foram.

- Repositório de Interfaces: Não foi necessário seu uso;
- Interface de invocação dinâmica: Não foi necessário seu uso;

- CORBA *security*: O uso do SSL foi testado mas não foi incluído no teste do protótipo, apesar de ser fortemente recomendado;
- *Naming Service*: Foi bastante utilizado;
- Adaptador de Objetos Portável (POA): Foi utilizado, mas não foram exploradas características de ativação dos objetos;
- C++ *mapping*: Foi bastante utilizado;
- Cliente/Servidor: Os drivers desenvolvidos foram programados para funcionar simultaneamente como cliente e servidor, a fim de permitir envio e recebimento de operações.
- *Event Service*: Este serviço poderia ser utilizado para permitir uma outra forma de implementação do sistema que dispensaria a solução descrita no item acima. A programação do serviço de eventos se mostrou demasiado complexa e o esforço e complexidade inseridas não justificaram seu uso.

Nos capítulos 4 e 5, onde serão mostrados o modelo de gerência e o desenvolvimento do protótipo, veremos mais detalhes sobre a utilização do padrão CORBA.

Capítulo 4

Modelo de Gerência de Segurança

Este capítulo descreve em detalhes o modelo de gerência proposto nesta tese, utilizando os conceitos fundamentais de ferramentas de segurança de redes e do padrão CORBA obtidos nos capítulos anteriores.

Primeiramente, apresentamos os componentes do modelo e uma visualização de como eles interagem. Partimos então para uma descrição usando a UML e a padronização das interfaces entre os objetos. Terminamos com um detalhamento da API e das estruturas de dados utilizadas. Esta linha de abordagem mostra primeiramente o sistema de uma visão mais abstrata, caminhando para uma abordagem mais a nível de implementação, que culminará no capítulo seguinte a este, onde a especificação do SIGSEC será detalhada.

4.1 Visão Geral e Componentes do Modelo

O objetivo deste modelo de segurança é padronizar as interfaces de ferramentas de segurança (inicialmente *firewalls* e IDSs) permitindo interoperabilidade entre estas ferramentas, flexibilidade e facilidade na sua gerência. Para alcançar este objetivo foi utilizado o padrão CORBA, para especificar claramente as interfaces entre as ferramentas de segurança e desenvolver programas (que chamaremos de *drivers*) que efetivamente traduzem e adaptam o formato de regras, *logs* e configuração ao formato proposto.

O modelo é composto de diagramas, padronizações das interfaces, e uma API detalhada para o seu correto uso dentro de um ambiente heterogêneo e distribuído. Graças ao uso do padrão CORBA, os componentes descritos podem estar em qualquer ponto de uma rede, podem ser instalados em qualquer sistema operacional ou plataforma de hardware, e ainda, existe a possibilidade de se escolher a linguagem

de programação, bastando que a implementação de CORBA utilizada suporte-a.

Na Figura 4.1 é apresentada a arquitetura do modelo de gerência de segurança, ela contém diversos componentes e interações que serão comentadas a seguir.

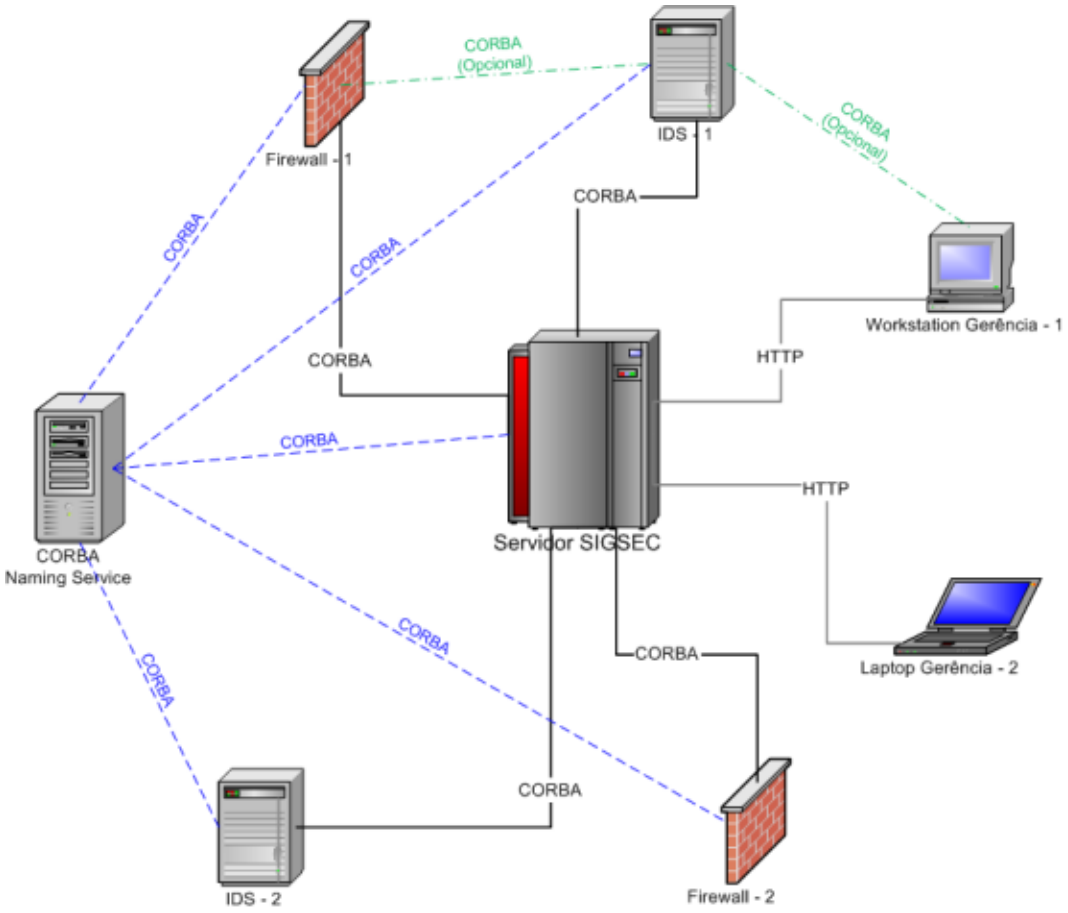


Figura 4.1: Arquitetura do Modelo de Gerência de Segurança

O primeiro aspecto importante a se observar é que os *firewalls* e IDSs espalhados pela rede se comunicam com o restante do sistema usando exclusivamente CORBA. O Servidor SIGSEC, que é onde se processam os *logs* e reside o banco de dados e servidor Web, também se comunica com o sistema usando CORBA, exceto os clientes de gerência, que acessam este sistema através da Web (com SSL), usando portanto o protocolo HTTPS.

Os *firewalls*, IDSs e o Servidor SIGSEC podem se encontrar facilmente graças ao uso do *Naming Service*. Assim, cada componente do sistema deve se registrar no *Naming Service* para que outros sistemas possam encontrá-lo, e consultar este serviço para encontrar outros. Cada componente registra alguns objetos que permitem o seu gerenciamento. O *Naming Service* pode ser instalado na mesma máquina onde

está o Servidor SIGSEC, ou em uma máquina dedicada, como na Figura 4.1.

Os clientes de gerência, apesar de não usarem CORBA, possuem também bastante flexibilidade, pois são usados apenas navegadores Web (que podem ser encontrados em qualquer plataforma), permitindo que o sistema possa ser acessado praticamente de qualquer lugar.

A forma bastante distribuída dos componentes no sistema permite que se adapte o modelo para as mais diversas situações, podendo por exemplo, permitir que *firewalls* e IDSs troquem informações entre si ou mesmo criar clientes de gerência que se comunicam diretamente com os os *firewalls* e IDSs usando CORBA, como mostram as linhas tracejadas onde se lê "opcional", na Figura 4.1.

Vamos examinar, a seguir, mais detalhadamente, cada componente do modelo.

4.1.1 Identificando os Componentes do Modelo

Firewalls: Os *firewalls* podem ser, a princípio, de qualquer modelo ou fabricante, a única restrição é que o sistema operacional onde o *firewall* será instalado suporte uma implementação de CORBA (Figura 4.2). Assim, é possível desenvolver um *driver* em CORBA para permitir que o *firewall* se comunique com o restante do sistema. No modelo proposto o *firewall* registra dois objetos: *PacketFilter* (filtro de pacotes) e *Nat* (tradução de endereços de rede), as quais são as funcionalidades de *firewall* mais utilizadas e aquelas suportadas atualmente pelo modelo.

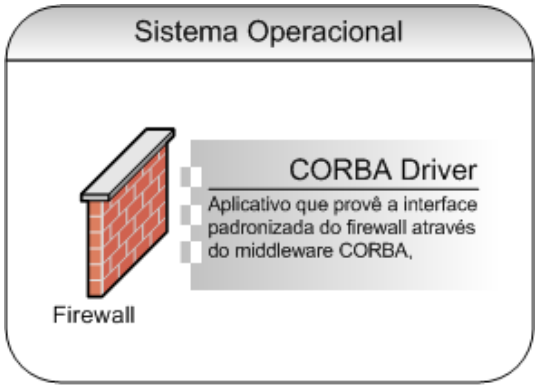


Figura 4.2: *Driver* CORBA do *Firewall*

IDS: Os IDSs podem ser, a princípio, de qualquer modelo ou fabricante, a única restrição é que o sistema operacional onde o IDS será instalado suporte uma

implementação de CORBA. Assim, é possível desenvolver um *driver* em CORBA (Figura 4.3) para permitir que o IDS se comunique com o restante do sistema. No modelo proposto, o IDS registra um objeto: *NIDS* (Sistema de Detecção de Intrusão Baseado em Rede), que é o tipo de IDS mais utilizado e o suportado atualmente pelo modelo.

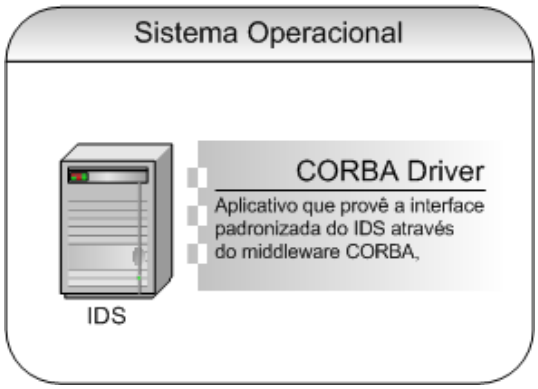


Figura 4.3: *Driver* CORBA do IDS

Servidor SIGSEC: O servidor SIGSEC é um sistema central que é responsável por receber e armazenar alertas e *logs* de *firewalls* e IDSs e autenticar clientes que desejem gerenciar estas ferramentas de segurança. O Servidor SIGSEC (Figura 4.4) é composto de:

- Servidor Web;
- CGI empregando CORBA;
- Banco de Dados SQL;
- *Daemon* receptor de *logs* e alertas (SIGSEC *Log Daemon*).

As ferramentas de segurança podem enviar seus *logs* e alertas (se comunicando de forma padronizada usando CORBA) para o SIGSEC *Log Daemon* este então, armazena os dados recebidos no banco de dados SQL.

Através do CGI, operando em conjunto com o servidor Web, clientes utilizando navegadores comuns podem se conectar, mediante senha, e gerenciar as ferramentas de segurança na rede, além disso, podem também ter acesso ao banco de dados SQL para verificação dos *logs* e alertas gravados.

No modelo proposto, o *Log Daemon* registra dois objetos: *FwLog* (log de *firewalls*) e *IDSAlert* (alertas de IDSs).

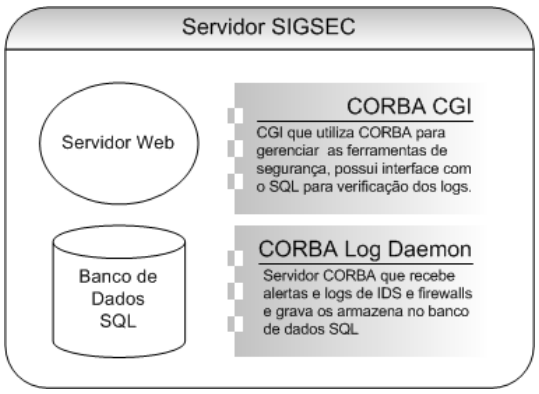


Figura 4.4: Servidor SIGSEC

Naming Service: O *Naming Service* é um serviço disponível no padrão CORBA que permite que objetos se registrem e sejam encontrados na rede através do seu nome, de forma similar ao serviço de nomes de domínio da Internet (DNS). Neste modelo, todo componente registra seu nome e objetos no *Naming Service*, e antes de qualquer ação, todo programa consulta o *Naming Service* para saber a localização do objeto desejado, obter uma referência para ele e se comunicar.

O uso deste serviço facilita muito, pois a única referência que os componentes do sistema precisam saber é a localização do *Naming Service*.

O *Naming Service* é portanto uma peça chave e essencial para que o sistema funcione, este pode estar sendo executado em uma máquina dedicada ou mesmo em quaisquer outras máquinas disponíves no sistema, mas deve-se alertar que seu não funcionamento invalida qualquer operação, devendo ser portanto instalado em uma plataforma de grande robustez (Figura 4.5).

A forma como o *Naming Service* foi utilizado, será examinada em detalhes mais adiante.

Clientes de Gerência: Os clientes de gerência (Figura 4.6) são as máquinas que os admistradores usam para se conectar ao sistema e gerenciar as ferramentas de segurança existentes. Para facilitar ao máximo este procedimento se optou pela simples interface Web, disponível praticamente em qualquer plataforma de hardware e software.

Este clientes se conectam e se autenticam no Servidor SIGSEC, onde podem então, dependendo do seu privilégio gerenciar as ferramentas de segurança,

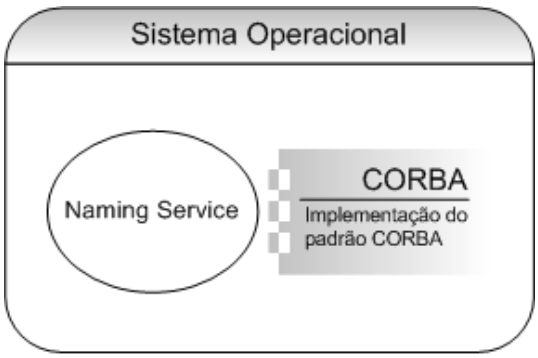


Figura 4.5: CORBA *Naming Service*

alterando regras, verificando *logs*, etc.

Além da autenticação, os clientes se comunicam com o servidor Web usando SSL para maior segurança.

Figura 4.6: Clientes de Gerência

Agora que verificamos cada componente do modelo, vamos analisar como ocorre a comunicação entre estes componentes, vamos iniciar com o serviço de nomes do CORBA (*Naming Service*), que participa do ínicio de todas as transações e que necessitou de uma modelagem e organização bem específica.

4.1.2 Modelagem do Naming Service

Antes que um *firewall* envie um *log*, ou um cliente de gerência deseje alterar uma regra de um IDS através do Servidor SIGSEC, por exemplo, é necessário que estes programas obtenham referências (saibam a localização) dos objetos com os quais desejam realizar estas operações.

Desta forma, ao iniciar, cada objeto se registra no servidor de nomes do CORBA (*Naming Service*) de modo que possa ser encontrado por algum programa que tenha

apenas a referência do *Naming Service*. Na Figura 4.7 é mostrado o grafo do *Naming Service* contendo um *firewall* chamado **PF-1** e um IDS chamado **Snort-1**.

Para que o modelo suportasse o gerenciamento de varias ferramentas de segurança adotou-se uma hierarquia no *Naming* na qual o primeiro nível é formado por contextos referentes aos tipos de ferramentas de segurança gerenciadas, no modelo atual temos os contextos:

- **Firewall**;
- **IDS**;
- **SigSecLog**.

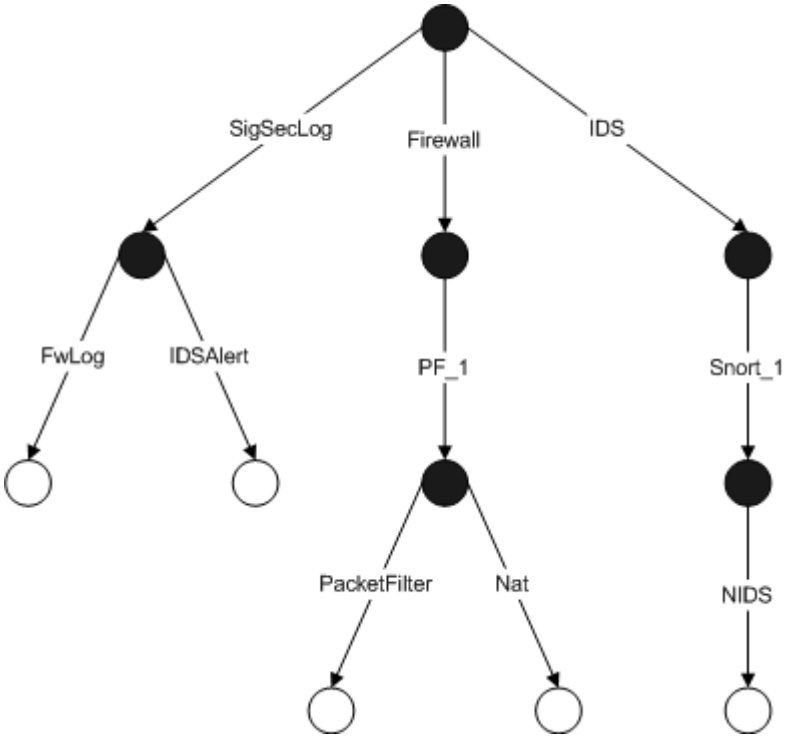


Figura 4.7: Grafo Montado no Servidor de Nomes (*Naming Service*)

O contexto SigSecLog é criado pelo programa SIGSEC *Log Daemon* e registra dois objetos que são responsáveis por receber os *logs* dos *firewalls* e os alertas dos IDSs, são eles: **FwLog** e **IDSAlert** respectivamente.

O contexto **Firewall** é criado pelo *driver* CORBA do primeiro *firewall* que seja colocado no ambiente de gerência. Sob este contexto, o *driver* do *firewall* registra outro contexto com um nome único seu na rede, de forma que o sistema possa

suportar vários *firewalls* de maneira organizada e escalável. Sob o contexto do nome do *firewall* (**PF-1** na figura) o *driver* registra dois objetos: **PacketFilter** e **Nat**.

Se um outro *firewall* entra na rede, digamos **PF-2**, ele não deve criar o contexto **Firewall** novamente, ele deve verificar sua existência, obter uma referência para ele e registrar o contexto **PF-2** sob ele.

O contexto **IDS** segue o mesmo esquema, ele é criado pelo *driver* CORBA do primeiro IDS que seja colocado no ambiente de gerência, sob este contexto, o *driver* do IDS registra outro contexto com um nome único seu na rede, de forma que o sistema possa suportar vários IDSs de maneira organizada e escalável. Sob o contexto do nome do IDS (**Snort-1** na figura) o *driver* registra um objeto chamado **NIDS**.

Se um outro IDS entra na rede, digamos **Snort-2**, ele não deve criar o contexto **IDS** novamente, ele deve verificar sua existência, obter uma referência para ele e registrar o contexto **Snort-2** sob ele.

4.1.3 Comunicação entre os Componentes do Sistema

Toda a comunicação entre as ferramentas de segurança e o Servidor SIGSEC são feitas através de CORBA, a única comunicação que não usa CORBA está no tráfego dos clientes de gerência com o Servidor SIGSEC (que usa HTTPS).

A comunicação através do *middleware* CORBA se dá através da chamada de operações padronizadas que constituem a interface de cada componente. Devido ao uso de CORBA, e o fato das interfaces suprirem as operações desejadas ao funcionamento da ferramenta de segurança, o desenvolvimento de um *driver* para um determinado *firewall* ou IDS é bastante facilitado, pois o programador não precisará se preocupar com as diferenças de arquiteturas e sistemas operacionais, e não precisará programar em baixo nível (*sockets*).

Antes de chamar as operações da interface, o programa deve obter a referência do objeto no qual ele deseja fazer a operação, como descrito na modelagem do *Naming Service* (sub-seção anterior).

As operações e estruturas de dados passadas e recebidas serão descritas nos diagramas em UML [17] na seção seguinte e uma descrição detalhada de cada operação será feita logo após.

4.2 Descrição do Modelo Usando a UML

Vamos utilizar a linguagem de modelagem UML para mostrar através de alguns diagramas as interações dos usuários com o sistema e especificar a interface e estruturas de dados utilizados.

4.2.1 Diagrama de Casos de Uso

O diagrama de casos de uso da Figura 4.8 representa as operações que um administrador (ou gerente) pode executar no sistema. Os casos de uso mostrado são válidos para IDSs ou *firewalls*.

Os casos de uso apresentados formaram a base para se construir o conjunto de operações necessárias para compor a interface. Os casos de uso "Insere/Altera Administrador" e "Remove Administrador" não geraram nenhuma operação nas interfaces de *firewalls* ou IDSs pois não pertencem exclusivamente à estes componentes, pertencem ao sistema como um todo.

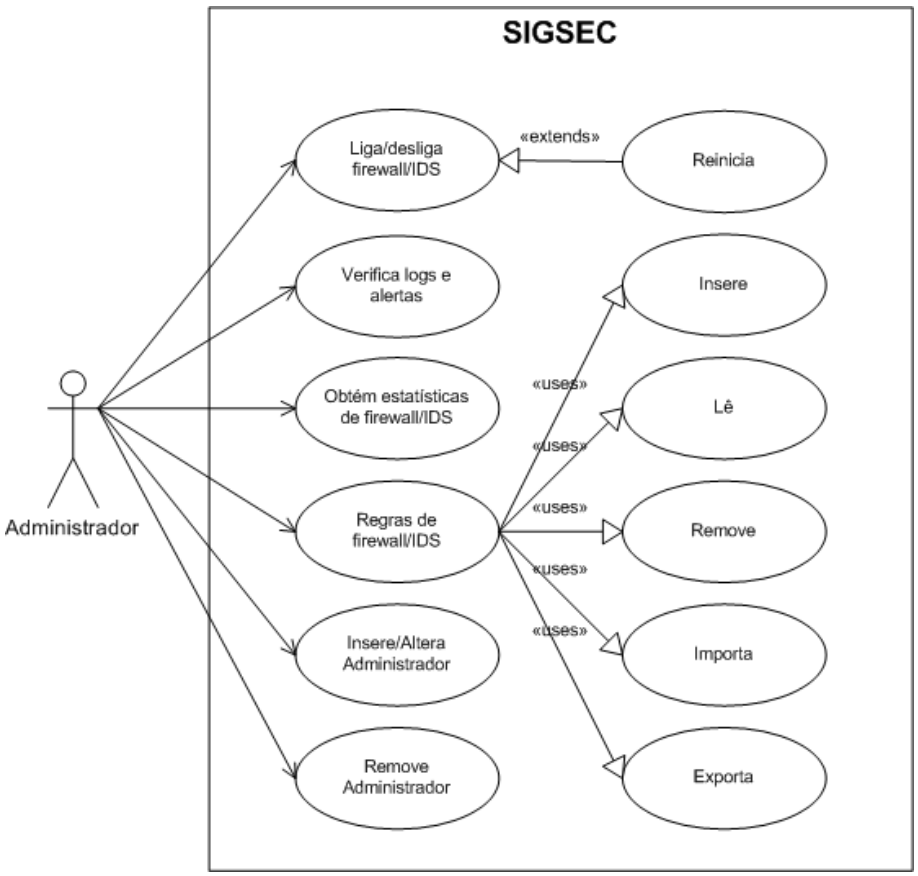


Figura 4.8: Diagrama de Casos de Uso

Na gerência das ferramentas de segurança em questão (*firewalls* e IDSs) foram quatro os requisitos básicos para se montar este diagrama de casos de uso, isto é, para ambas ferramentas, deveria ser possível:

- Controlar o estado (ligada/desligada);
- Obter estatísticas;
- Analisar os *logs* e alertas gerados;
- Configurar as regras.

Estes quatro requisitos estão resumidos no diagrama de casos de uso da Figura 4.8, sendo que para maior clareza, não estão duplicados os casos onde a configuração da regra é de NAT, filtro de pacotes ou regras de IDS, pois para todas as ferramentas (IDSs e *firewalls*) elas são regras, e possuem os mesmos casos de uso adicionais: inserir, remover, etc.

4.2.2 Diagrama de componentes

O diagrama de componentes da Figura 4.9 vai nos mostrar de uma forma resumida quais são os componentes que compõem o modelo e as interfaces providas por cada um deles. Na implementação, cada componente vai corresponder a um *driver* que trabalha associado a uma determinada ferramenta de segurança e que instancia e registra objetos CORBA, podendo receber as operações descritas na interface.

4.2.3 Diagramas de classes

Com base no diagrama de componentes visto anteriormente podemos então criar vários diagramas de classes que foram subdivididos por interface, para uma melhor visualização.

As escolhas dos atributos das estruturas de dados e operações das interfaces foram baseadas na pesquisa de alguns *firewalls* e IDSs existentes no mercado, bem como nos requerimentos básicos dos casos de uso.

Existiu então uma linha tênue entre abarcar a maior quantidade de possibilidades de configuração e não poluir a padronização com atributos de configuração muito específicos de um determinado *firewall* ou IDS. Portanto, a padronização através desta

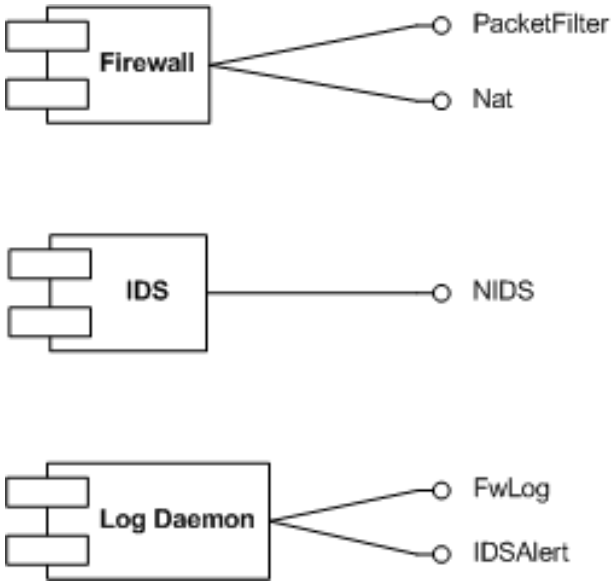


Figura 4.9: Diagrama de Componentes

modelagem não pretende suportar todas as configurações possíveis das ferramentas de segurança, mas sim suportar as configurações mais comuns entre elas.

Diagrama da interface *Firewall::PacketFilter*:

Este diagrama (Figura 4.10) pretende padronizar a interface de um filtro de pacotes de um *firewall*. A estrutura **Stats**, assim como a operação *GetStats()* não deveriam pertencer ao filtro de pacotes. O motivo da escolha foi por se achar desnecessária a criação de uma interface somente para estatísticas e também porque o filtro de pacotes está presente em quaquer *firewall*.

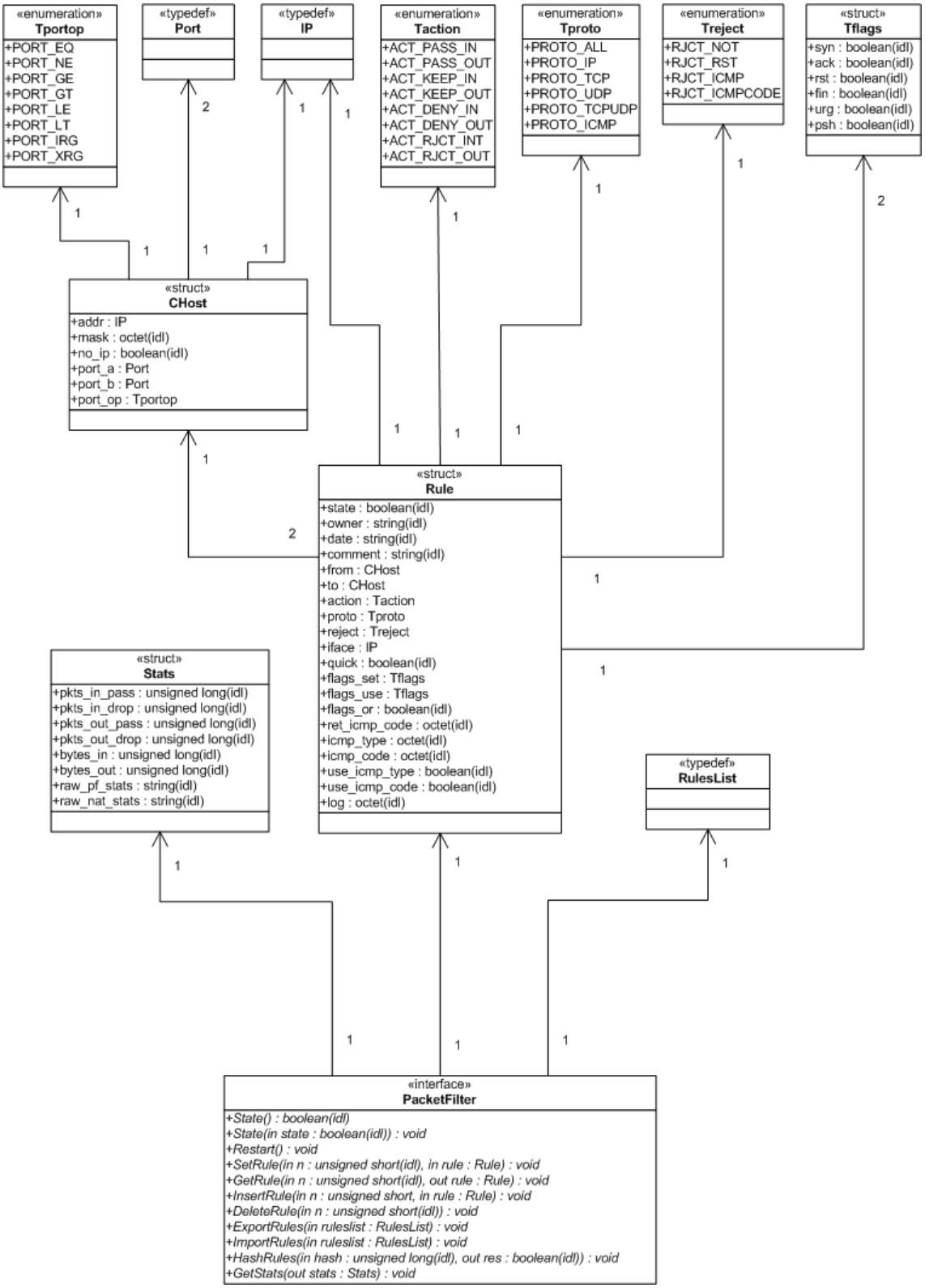


Figura 4.10: Diagrama da Interface *Firewall::PacketFilter*

Diagrama da interface *Firewall::Nat*:

Este diagrama (Figura 4.11) pretende padronizar a interface de NAT de um *firewall*. Algumas estruturas de dados utilizadas são exatamente as mesmas usadas na modelagem do filtro de pacotes.

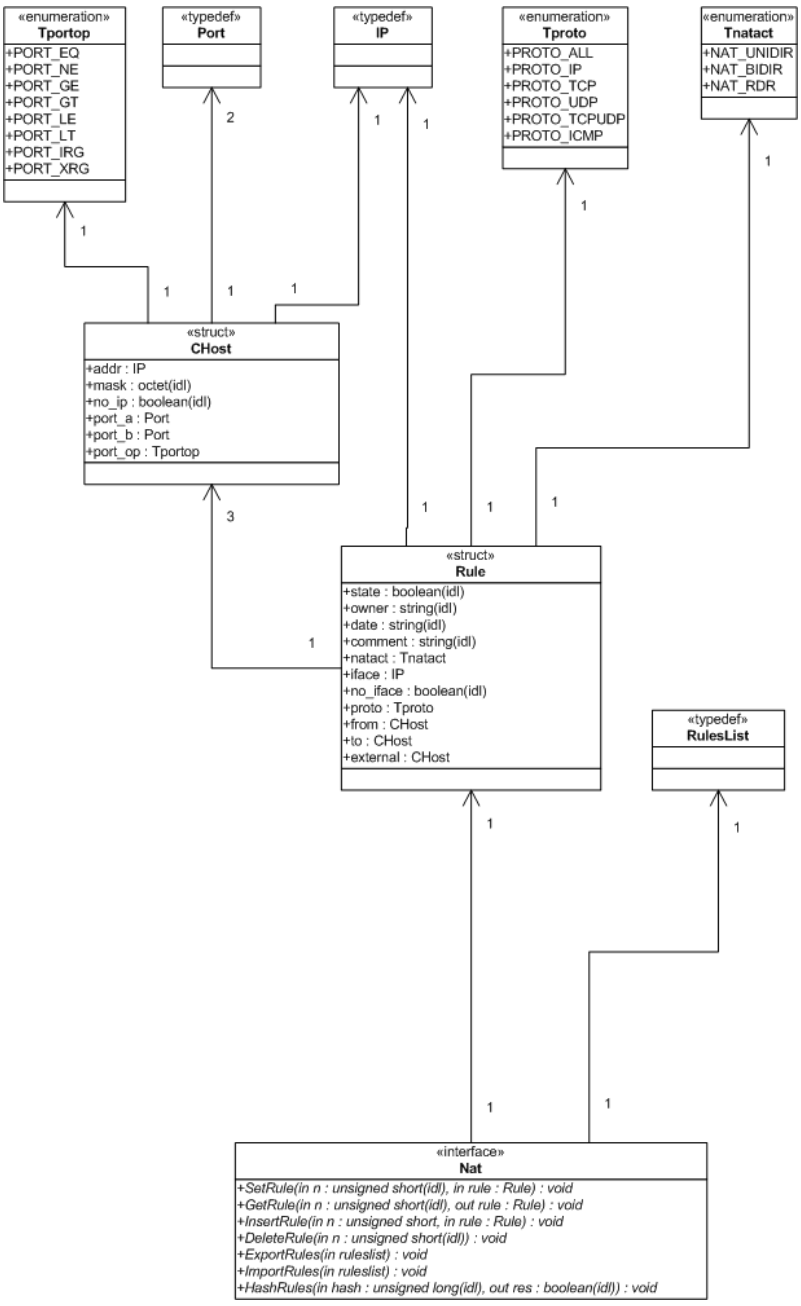


Figura 4.11: Diagrama da Interface *Firewall::Nat*

Diagrama da interface *Firewall::FwLog*:

Este diagrama (Figura 4.12) pretende padronizar a interface de como um *firewall* envia seus *logs*. Algumas estruturas de dados utilizadas são exatamente as mesmas usadas anteriormente. É importante notar, que na implementação do *driver* não existirá um objeto que suporte a operação *SendLog()*. Este objeto será instanciado pelo SIGSEC *Log Daemon*, e o *driver* do *firewall* é quem irá executar a operação *SendLog()*.

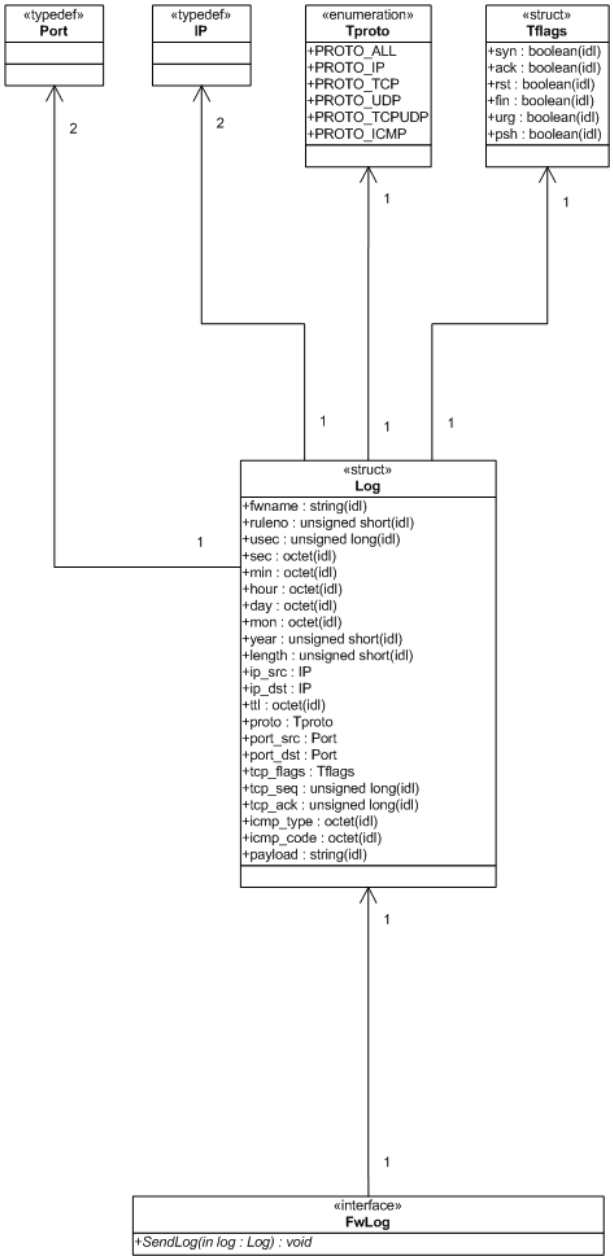


Figura 4.12: Diagrama da Interface *Firewall::FwLog*

Diagrama da interface *IDS::NIDS*:

Este diagrama (Figura 4.13) pretende padronizar a interface de um sensor baseado em rede de um IDS. O acrônimo NIDS significa: *Network Intrusion Detection System* (Sistema de detecção de intrusão baseado em rede). A estrutura **Stats**, assim como a operação *GetStats()* não deveriam pertencer ao NIDS. O motivo da escolha foi por se achar desnecessária a criação de uma interface somente para estatísticas e também porque o NIDS é o tipo de IDS mais utilizado.

No caso do IDS, foi interessante a criação de grupos de regras, por isso a inclusão de operações tais como: *ImportGroups()* e *ExportGroups()*, pois a quantidade de regras de um NIDS pode ser muito grande e faz sentido separar as regras em grupos que detectam tipos de ataques diferentes.

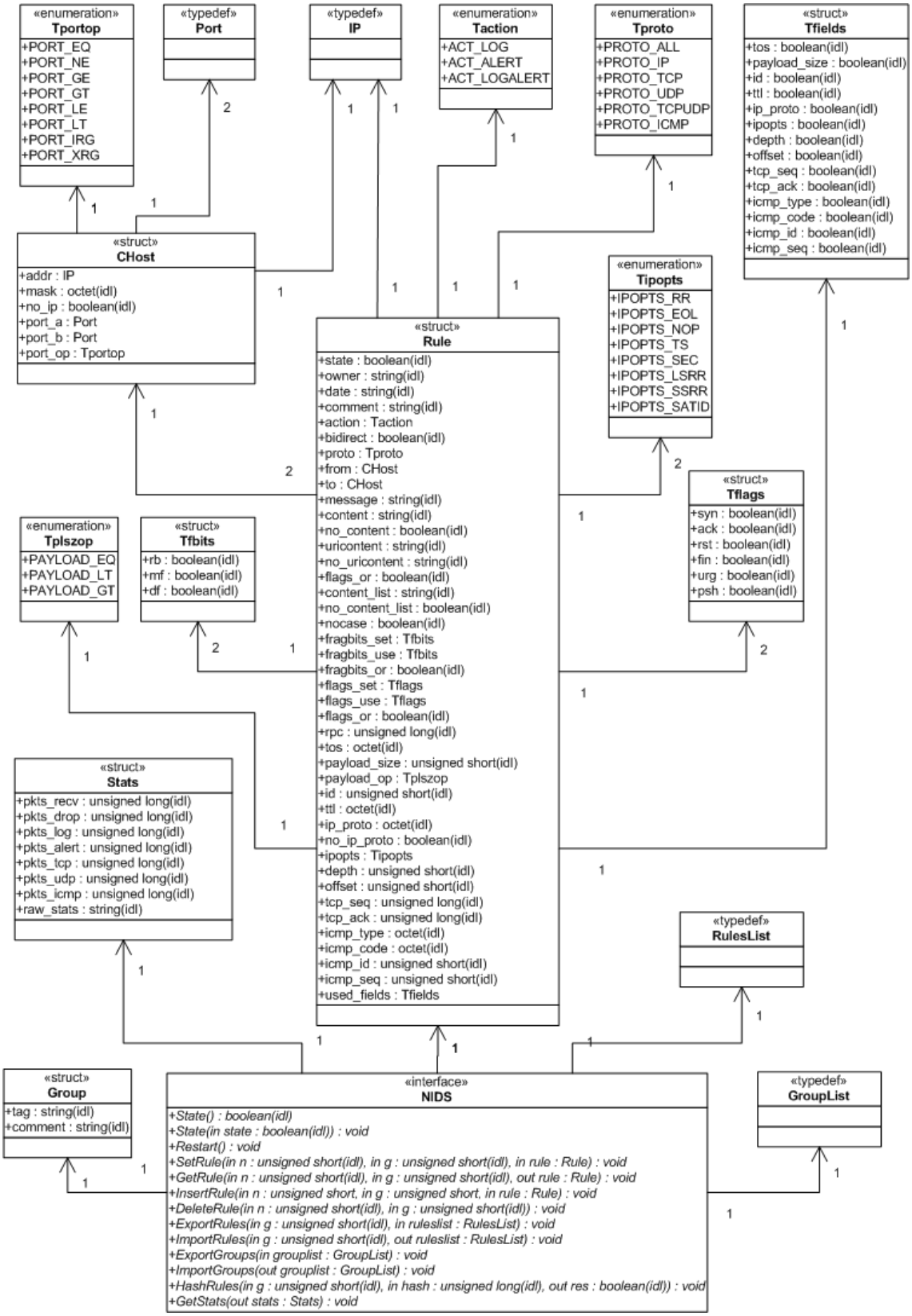


Figura 4.13: Diagrama da Interface *IDS::NIDS*

Diagrama da interface *IDS::IDSAlert*:

Este diagrama (4.14) pretende padronizar a interface de como um IDS envia seus alertas. Algumas estruturas de dados utilizadas são exatamente as mesmas usadas anteriormente. É importante notar, que na implementação do *driver* não existirá um objeto que suporte a operação *SendAlert()*, este objeto será instanciado pelo SIGSEC *Log Daemon* e o *driver* do IDS é que irá executar a operação *SendAlert()*.

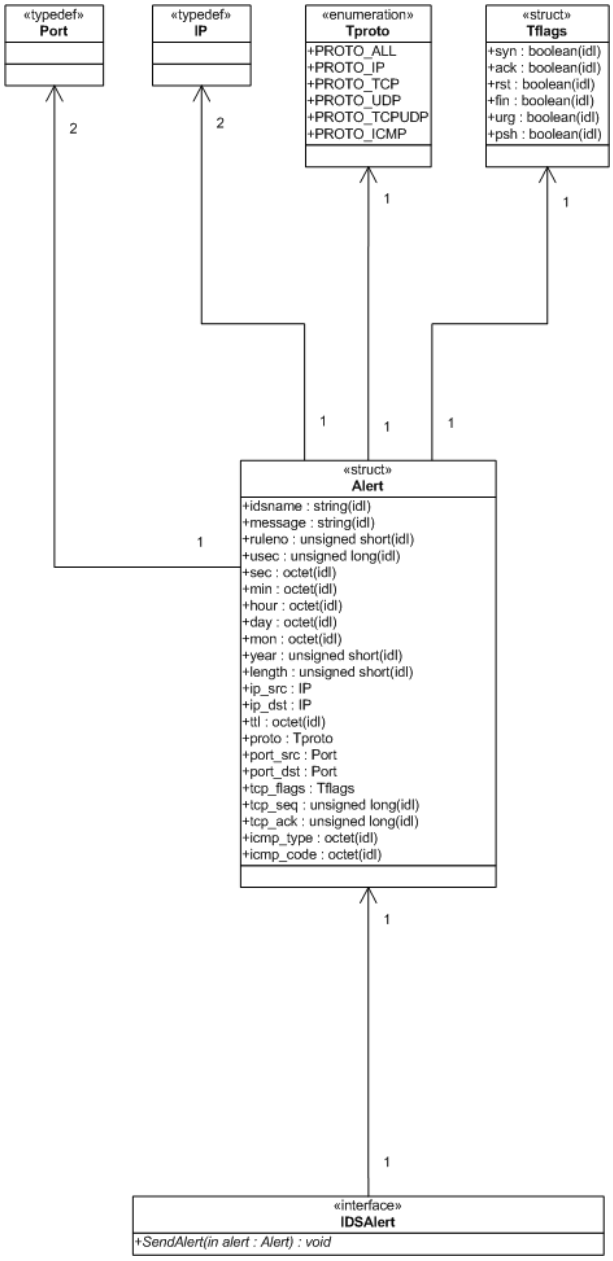


Figura 4.14: Diagrama da Interface *IDS::IDSAlert*

4.3 Padronização das Interfaces

A padronização das interfaces foi feita através do uso da linguagem de definição de interfaces (IDL) do padrão CORBA. As interfaces permitem um determinado conjunto de operações nos *firewalls* e IDSs. Este conjunto de operações foi escolhido baseando-se em pesquisas em alguns *firewalls* e IDSs existentes no mercado. As interfaces também determinam como as ferramentas enviam *logs* e alertas.

4.3.1 Arquivos de Linguagem de Definição de Interface

Os aquivos escritos na IDL do padrão CORBA são então utilizados para especificar cada ferramenta de segurança envolvida. Existe portanto, neste modelo, dois arquivos IDL: `sigsec_fw.idl` para a padronização de *firewalls* e `sigsec_ids.idl` para padronização de IDSs.

Evidentemente, outras ferramentas de segurança podem ser padronizadas e inseridas neste modelo, iniciando-se pela padronização de suas interfaces, ex.: VPNs.

Cada arquivo IDL deve conter um conjunto de interfaces e estruturas de dados relativos a ferramenta que está sendo tratada. Foi utilizada a palavra reservada `module` da IDL que define um espaço de nomes independente dos demais, tal como `namespace` em C++. Na verdade, quando se utiliza CORBA com a linguagem C++ a construção `module` é realmente mapeada para `namespace`.

Assim, em cada arquivo IDL temos apenas uma construção `module` e dentro desta temos as estruturas de dados, tipos definidos, interfaces e operações.
Ex.: Esboço do arquivo `sigsec_fw.idl`:

```
module Firewall {

    // Tipos definidos
    // Estruturas de dados gerais ...

    interface Packetfilter {
        // Estruturas ...
        // Operações ...
    };

    interface Nat {
        // Estruturas ...
        // Operações ...
    };

    interface FwLog {
```

```
        // Estruturas ...
        // Operações ...
    };

};
```

Ex.: Esboço do arquivo `sigsec_ids.idl`:

```
module IDS {

    // Tipos definidos
    // Estruturas de dados gerais ...

    interface NIDS {
        // Estruturas ...
        // Operações ...
    };

    interface IDSAlert {
        // Estruturas ...
        // Operações ...
    };

};
```

Nas subseções seguintes serão apresentadas respectivamente a IDL para *firewalls* e a IDL para IDSs, o detalhamento completo das estruturas e operações que podem ser realizadas serão vistos na próxima seção, intitulada "Detalhamento da API".

4.3.2 Padronização para Firewalls

Como visto na modelagem em UML da seção anterior o componente **Firewall** vai possuir três interfaces e uma certa quantidade de estruturas de dados de apoio. Veremos em seguida a IDL completa para o componente **Firewall**, definida através de um `module` em CORBA, que faz parte do arquivo `sigsec_fw.idl`.

```
module Firewall {

    typedef unsigned long  IP;
    typedef unsigned short Port;
    typedef unsigned long  u_long ;
    typedef unsigned short u_short;

    enum Tproto  { PROTO_ALL,
                   PROTO_IP,
                   PROTO_TCP,
                   PROTO_UDP,
```

```
                PROTO_TCPUDP,
                PROTO_ICMP };

enum Taction { ACT_PASS_IN,
                ACT_PASS_OUT,
                ACT_KEEP_IN,
                ACT_KEEP_OUT,
                ACT_DENY_IN,
                ACT_DENY_OUT,
                ACT_RJCT_IN,
                ACT_RJCT_OUT };

enum Tportop { PORT_EQ,
                PORT_NE,
                PORT_GE,
                PORT_GT,
                PORT_LE,
                PORT_LT,
                PORT_IRG,
                PORT_XRG };

enum Treject { RJCT_NOT,
                RJCT_RST,
                RJCT_ICMP,
                RJCT_ICMPCODE };

enum Tnatact { NAT_UNIDIR,
                NAT_BIDIR,
                NAT_RDR };

struct Tflags {
    boolean syn, ack, rst, fin, urg, psh;
};

struct CHost {
    IP      addr;
    octet   mask;
    boolean no_ip;
    Port    port_a;
    Port    port_b;
    Tportop port_op;
};

interface PacketFilter {

    struct Rule {
        boolean state;
        string  owner;
        string  date;
        string  comment;
```

```
        CHost   from;
        CHost   to;
        Taction action;
        Tproto  proto;
        Treject reject;
        IP      iface;
        boolean quick;
        Tflags  flags_set;
        Tflags  flags_use;
        boolean flags_or;
        octet   ret_icmp_code;
        octet   icmp_type;
        octet   icmp_code;
        boolean used_icmp_type;
        boolean used_icmp_code;
        octet   log;
    };

    typedef sequence<Rule> RulesList;

    struct Stats {
        u_long  pkts_in_pass;
        u_long  pkts_in_drop;
        u_long  pkts_out_pass;
        u_long  pkts_out_drop;
        u_long  bytes_in;
        u_long  bytes_out;
        string  raw_pf_stats;
        string  raw_nat_stats;
    };

    attribute boolean State;
    void Restart();
    void SetRule(in u_short n, in  Rule rule);
    void GetRule(in u_short n, out Rule rule);
    void InsertRule(in u_short n, in Rule rule);
    void DeleteRule(in u_short n);
    void ExportRules(in  RulesList ruleslist);
    void ImportRules(out RulesList ruleslist);
    void HashRules(in u_long hash, out boolean res);
    void GetStats(out Stats stats);
};


interface Nat {

    struct Rule {
        boolean state;
        string  owner;
        string  date;
```

```
        string  comment;
        Tnatact natact;
        IP      iface;
        boolean no_iface;
        Tproto  proto;
        CHost   from;
        CHost   to;
        CHost   external;
    };

    typedef sequence<Rule> RulesList;

    void SetRule(in u_short n, in  Rule rule);
    void GetRule(in u_short n, out Rule rule);
    void InsertRule(in u_short n, in Rule rule);
    void DeleteRule(in u_short n);
    void ExportRules(in  RulesList ruleslist);
    void ImportRules(out RulesList ruleslist);
    void HashRules(in u_long hash, out boolean res);
};


interface FwLog {

    struct Log {
        string  fwname;
        u_short ruleno;
        u_long  usec;
        octet   sec;
        octet   min;
        octet   hour;
        octet   day;
        octet   mon;
        u_short year;
        u_short length;
        IP      ip_src;
        IP      ip_dst;
        octet   ttl;
        Tproto  proto;
        Port    port_src;
        Port    port_dst;
        Tflags  tcp_flags;
        u_long  tcp_seq;
        u_long  tcp_ack;
        octet   icmp_type;
        octet   icmp_code;
        string  payload;
    };

    oneway void SendLog(in Log log);
```

```
    };

};
```

4.3.3 Padronização para IDSs

Como visto na modelagem em UML da seção anterior o componente **IDS** vai possuir duas interfaces e uma certa quantidade de estruturas de dados de apoio. Veremos em seguida a IDL completa para o componente **IDS**, definida através de um `module` em CORBA, que faz parte do arquivo `sigsec_ids.idl`.

```
module IDS {

    typedef unsigned long  IP;
    typedef unsigned short Port;
    typedef unsigned long  u_long ;
    typedef unsigned short u_short;

    enum Taction { ACT_LOG,
                   ACT_ALERT,
                   ACT_LOGALERT};

    enum Tproto  { PROTO_ALL,
                   PROTO_IP,
                   PROTO_TCP,
                   PROTO_UDP,
                   PROTO_TCPUDP,
                   PROTO_ICMP };

    enum Tportop { PORT_EQ,
                   PORT_NE,
                   PORT_GE,
                   PORT_GT,
                   PORT_LE,
                   PORT_LT,
                   PORT_IRG,
                   PORT_XRG };

    enum Tplszop { PAYLOAD_EQ,
                   PAYLOAD_LT,
                   PAYLOAD_GT };

    enum Tipopts { IPOPTS_RR,
                   IPOPTS_EOL,
                   IPOPTS_NOP,
                   IPOPTS_TS,
                   IPOPTS_SEC,
```

```
                IPOPTS_LSRR,
                IPOPTS_SSRR,
                IPOPTS_SATID };

struct Tfields {
    boolean tos, payload_size, id, ttl, ip_proto, ipopts, depth, offset;
    boolean tcp_seq, tcp_ack, icmp_type, icmp_code, icmp_id, icmp_seq;
};

struct Tfbits {
    boolean rb, mf, df;
};

struct Tflags {
    boolean syn, ack, rst, fin, urg, psh;
};

struct CHost {
    IP      addr;
    octet   mask;
    boolean no_ip;
    Port    port_a;
    Port    port_b;
    Tportop port_op;
};


interface NIDS {

    struct Rule {
        boolean state;
        string  owner;
        string  date;
        string  comment;
        Taction action;
        boolean bidirect;
        Tproto  proto;
        CHost   from;
        CHost   to;
        string  message;
        string  content;
        boolean no_content;
        string  uricontent;
        boolean no_uricontent;
        string  content_list;
        boolean no_content_list;
        boolean nocase;
        Tfbits  fragbits_set;
        Tfbits  fragbits_use;
        boolean fragbits_or;
```

```
    Tflags  flags_set;
    Tflags  flags_use;
    boolean flags_or;
    u_long  rpc;
    octet   tos;
    u_short payload_size;
    Tplszop payload_op;
    u_short id;
    octet   ttl;
    octet   ip_proto;
    boolean no_ip_proto;
    Tipopts ipopts;
    u_short depth;
    u_short offset;
    u_long  tcp_seq;
    u_long  tcp_ack;
    octet   icmp_type;
    octet   icmp_code;
    u_short icmp_id;
    u_short icmp_seq;
    Tfields used_fields;
};

typedef sequence<Rule> RulesList;

struct Group {
    string tag;
    string comment;
};

typedef sequence<Group> GroupList;

struct Stats {
    u_long pkts_recv;
    u_long pkts_drop;
    u_long pkts_log;
    u_long pkts_alert;
    u_long pkts_tcp;
    u_long pkts_udp;
    u_long pkts_icmp;
    string raw_stats;
};

attribute boolean State;
void Restart();
void SetRule(in u_short n, in u_short g, in  Rule rule);
void GetRule(in u_short n, in u_short g, out Rule rule);
void InsertRule(in u_short n, in u_short g, in Rule rule);
void DeleteRule(in u_short n, in u_short g);
void ExportRules(in u_short g, in  RulesList ruleslist);
```

```
        void ImportRules(in u_short g, out RulesList ruleslist);
        void ImportGroups(out GroupList grouplist);
        void ExportGroups(in  GroupList grouplist);
        void HashRules(in u_short g, in u_long hash, out boolean res);
        void GetStats(out Stats stats);

    };


    interface IDSAlert {

        struct Alert {
            string  idsname;
            string  message;
            u_short ruleno;
            u_long  usec;
            octet   sec;
            octet   min;
            octet   hour;
            octet   day;
            octet   mon;
            u_short year;
            u_short length;
            IP      ip_src;
            IP      ip_dst;
            octet   ttl;
            Tproto  proto;
            Port    port_src;
            Port    port_dst;
            Tflags  tcp_flags;
            u_long  tcp_seq;
            u_long  tcp_ack;
            octet   icmp_type;
            octet   icmp_code;
        };

        oneway void SendAlert(in Alert alert);

    };

};
```

4.4 Detalhamento da API

Uma API (*Application Programming Interface*), é o conjunto de operações (funções, métodos) que podem ser utilizadas por um determinado software. No nosso caso teremos uma API para *firewalls* e outra para IDSs.

Antes de detalhar as operações da API iremos definir todas as estruturas de dados utilizadas, tanto para *firewalls* quanto para IDSs.

4.4.1 Estruturas Utilizadas na IDL para *firewalls*

Enum Firewall::Taction : Define a ação a ser tomada em uma determinada regra do filtro de pacotes.

- ACT_PASS_IN : O filtro permite um pacote que entra na interface de rede;
- ACT_PASS_OUT : O filtro permite um pacote que sai pela interface de rede;
- ACT_KEEP_IN : O filtro permite um pacote que entra na interface de rede, mantendo o estado para esta conexão;
- ACT_KEEP_OUT : O filtro permite um pacote sai pela interface de rede, mantendo o estado para esta conexão;
- ACT_DENY_IN : O filtro bloqueia um pacote que entra na interface na interface de rede sem enviar mensagem alguma;
- ACT_DENY_OUT : O filtro bloqueia um pacote que sai pela interface na interface de rede sem enviar mensagem alguma;
- ACT_RJCT_IN : O filtro bloqueia um pacote que entra na interface de rede, enviando uma mensagem para o endereço de origem;
- ACT_RJCT_OUT : O filtro bloqueia um pacote que sai pela interface de rede, enviando uma mensagem para o endereço de origem.

Enum Firewall::Tproto : Define os protocolos suportados.

- PROTO_ALL : Todos os protocolos: IP, TCP, UDP, ICMP;
- PROTO_IP : Protocolo IP;
- PROTO_TCP : Protocolo TCP;
- PROTO_UDP : Protocolo UDP;
- PROTO_TCPUDP : Protocolo TCP e UDP;

- PROTO_ICMP : Protocolo ICMP.

Enum Firewall::Tportop : Define as operações que podem ser feitas com uma ou duas portas em uma determinada regra.

- PORT_EQ : A porta é igual ao valor especificado;
- PORT_NE : A porta é diferente do valor especificado;
- PORT_GE : A porta é maior ou igual ao valor especificado;
- PORT_GT : A porta é maior do que o valor especificado;
- PORT_LE : A porta é menor ou igual ao valor especificado;
- PORT_LT : A porta é menor do que o valor especificado;
- PORT_IRG : A porta está entre os dois valores especificados;
- PORT_XRG : A porta está fora da faixa especificada por dois valores.

Enum Firewall::Treject : Define o tipo de mensagem de retorno para o endereço de origem quando o filtro bloqueou (ACT_RJCT_IN ou ACT_RJCT_OUT) um pacote.

- RJCT_NOT : Não envia mensagem alguma;
- RJCT_RST : Envia um pacote TCP RST (pacote TCP com a flag *reset*);
- RJCT_ICMP : Envia um pacote do tipo ICMP UNREACHABLE (normalmente com o código especificando porta não encontrada);
- RJCT_ICMPCODE : Envia um pacote do tipo ICMP UNREACHABLE com o código especificado.

Enum Firewall::Tnatact : Define a ação feita pelo NAT.

- NAT_UNIDIR : Nat unidirecional, o mais comum, o estado da conexão é mantida quando originado do primeiro *host* especificado;

- NAT_BIDIR : Nat bidirecional, o estado da conexão é mantida quando originado de quaisquer dos *hosts* especificados;
- NAT_RDR : Redirecionamento, utilzado para redirecionar conexões que entram em uma determinada interface para outras máquinas, podendo mapear também a porta de destino.

Struct Firewall::Tflags : Define as flags do protocolo TCP. Todas são variáveis *boolean*, podendo estar ligadas ou não, em qualquer combinação.

- syn : Flag SYN;
- ack : Flag ACK;
- rst : Flag RST;
- fin : Flag FIN;
- urg : Flag URG;
- psh : Flag PSH.

Struct Firewall::CHost : Define os dados mais comuns de um determinado *host*.

- addr : Endereço IP do *host*, armazenado como um *unsigned long* (32 bits);
- mask : Máscara de rede, armazenada como um número de 0 a 32 (máscara no formato 10.10.0.0/24, onde 24 é a máscara);
- no_ip : Valor *boolean* que indica que o endereço deve ser negado na regra (todos endereços, menos este);
- port_a : Porta, para os protocolos TCP e UDP, armazenado como um *unsigned short* (16 bits);
- port_b : Porta, para os protocolos TCP e UDP (relevante apenas quando especificada uma faixa de portas, PORT_IRG ou PORT_XRG), armazenado como um *unsigned short* (16 bits);
- port_op : Operação lógica na regra com a(s) porta(s). Este tipo de dado é um Tportop.

Struct Firewall::PacketFilter::Rule : Define uma regra do filtro de pacotes de um *firewall*.

- state : Valor *boolean* que indica que a regra deve ser processada ou não;
- owner : *String* que contém o nome do administrador que configurou a regra;
- date : *String* que contém a data de criação/alteração da regra;
- comment : *String* com um comentário sucinto a respeito da regra;
- from : Estrutura CHost que especifica o *host* de origem;
- to : Estrutura CHost que especifica o *host* de destino;
- action : Ação a ser feita pela regra. Este tipo é um Taction;
- proto : Protocolo utilizado na regra. Este tipo é um Tproto;
- reject : Tipo de mensagem de retorno se a regra é de bloqueio e retorna mensagem. Este tipo é um Treject;
- iface : Endereço IP da interface de rede, armazenada como um *unsigned long* (32 bits);
- quick : Valor *boolean* que indica se o filtro deve ou não continuar a processar o restante das regras para um determinado pacote caso haja uma coincidência desta regra com este pacote;
- flags_set : Configuração das flags TCP para esta regra. Este tipo é um Tflags;
- flags_use : Configuração que indica quais flags TCP configuradas acima devem ser usadas. Este tipo é um Tflags;
- flags_or : Valor *boolean* que indica que a operação lógica entre as flags acima é OR e não AND;
- ret_icmp_code : Valor de 0 a 255 (octet) que representa o código ICMP de retorno no caso da regra ser de bloqueio e se o campo reject for RJCT_ICMPCODE;
- icmp_type : Valor de 0 a 255 (octet) que representa o tipo do pacote ICMP;

- icmp_code : Valor de 0 a 255 (octet) que representa o código do pacote ICMP;
- used_icmp_type : Valor *boolean* que indica que o icmp_type deve ser utilizado (pois icmp_type = 0 existe e seria utilizado na regra mesmo se não fosse especificado);
- used_icmp_code : Valor *boolean* que indica que o icmp_code deve ser utilizado (pois icmp_code = 0 existe e seria utilizado na regra mesmo se não fosse especificado);
- log : Valor de 0 a 255 (normalmente uma escala bem menor é usada) que indica o nível (quantidade de informações) de *log* que é gerada. O valor zero significa sem geração de *log* e quanto maior o valor maior é a quantidade de informações de *log*.

Struct Firewall::PacketFilter::Stats : Define as estatísticas básicas de um *firewall*. Os seis primeiros campos são do tipo *unsigned long* (32 bits).

- pkts_in_pass : Quantidade de pacotes que entraram e passaram pelo filtro;
- pkts_in_drop : Quantidade de pacotes que entraram e foram bloqueados pelo filtro;
- pkts_out_pass : Quantidade de pacotes que sairam e passaram pelo filtro;
- pkts_out_drop : Quantidade de pacotes que sairam e foram bloqueados pelo filtro;
- bytes_in : Total de *bytes* que entraram;
- bytes_out : Total de *bytes* que sairam;
- raw_pf_stats : *String* que contém estatísticas de filtro de pacotes de forma "crua" (específica do modelo do *firewall*);
- raw_nat_stats : *String* que contém estatísticas de NAT de forma "crua" (específica do modelo do *firewall*).

Struct Firewall::Nat::Rule : Define uma regra de NAT de um *firewall*.

- state : Valor *boolean* que indica que a regra deve ser processada ou não;
- owner : *String* que contém o nome do administrador que configurou a regra;
- date : *String* que contém a data de criação/alteração da regra;
- comment : *String* com um comentário sucinto a respeito da regra;
- natact : Ação a ser feita pela regra, este tipo é um Tnatact;
- iface : Endereço IP da interface de rede, armazenada como um *unsigned long* (32 bits);
- no_iface : Valor *boolean* que indica que a interface deve ser negada na regra (todas as interfaces, menos esta);
- proto : Protocolo utilizado na regra. Este tipo é um Tproto;
- from : Estrutura CHost que especifica o *host* de origem;
- to : Estrutura CHost que especifica o *host* de destino;
- external : Estrutura CHost que especifica o endereço para qual a origem é traduzida (geralmente o endereço da interface externa ligada a internet).

Struct Firewall::FwLog::Log : Define a estrutura de um *log* gerado pelo *firewall*.

- fwname : *String* que contém um nome único do *firewall* na rede;
- ruleno : Número da regra que gerou o *log* (*unsigned short*, 16 bits);
- usec : Instante do *log*, microsegundos (*unsigned long*, 32 bits);
- sec : Instante do *log*, segundos (octet);
- min : Instante do *log*, minutos (octet);
- hour : Instante do *log*, horas (octet);
- day : Instante do *log*, dia (octet);
- mon : Instante do *log*, mês (octet);
- year : Instante do *log*, ano (*unsigned short*, 16 bits);

- length : Tamanho do pacote, (*unsigned short*, 16 bits);
- ip_src : Porta de origem (*unsigned long*, 32 bits);
- ip_dst : Porta de destino (*unsigned long*, 32 bits);
- ttl : Tempo de vida do pacote (*Time To Live*). Valor de 0 a 255 (octet);
- proto : Protocolo do pacote. Este tipo é um Tproto;
- port_src : Porta de origem (*unsigned short*, 16 bits);
- port_dst : Porta de destino (*unsigned short*, 16 bits);
- tcp_flags : Estado das flags TCP. Este tipo é um Tflags;
- tcp_seq : Número de sequência TCP (*unsigned long*, 32 bits);
- tcp_ack : Número de *ack* TCP (*unsigned long*, 32 bits);
- icmp_type : Tipo do pacote ICMP, 0 a 255 (octet);
- icmp_code : Código do pacote ICMP, 0 a 255 (octet);
- payload : *String* que contém alguns bytes do início da camada de aplicação do pacote.

Tipos de dados definidos (*typedefs*) :

- Firewall::IP : Representa um IP, (*unsigned long*, 32 bits);
- Firewall::Port : Representa uma porta, (*unsigned short*, 16 bits);
- Firewall::u_long : Representa um número de 32 bits, (*unsigned long*);
- Firewall::u_short : Representa um número de 16 bits, (*unsigned short*);
- Firewall::PacketFilter::RulesList : Lista de regras de um filtro de pacotes, (*sequence<Rule>*);
- Firewall::Nat::RulesList : Lista de regras de NAT, (*sequence<Rule>*).

Descreveremos a seguir o significado de cada uma das operações da interface do *firewall*, seus argumentos e valores de retorno.

4.4.2 Operações das Interfaces do Firewall

As Tabelas 4.1 e 4.2 descrevem de forma resumida as operações providas pelas interfaces do *firewall*.

Operação	Descrição
State(st)	Liga/Desliga o *firewall* (st é *boolean*)
State()	Obtém estado do *firewall* (valor de retorno é *boolean*)
Restart()	Reinicia o *firewall*
SetRule(n, rule)	Especifica regra da posição *n* do filtro de pacotes
GetRule(n, rule)	Obtém regra da posição *n* do filtro de pacotes
InsertRule(n, rule)	Insere regra na posição *n* do filtro de pacotes
DeleteRule(n, rule)	Remove regra da posição *n* do filtro de pacotes
ExportRules(ruleslist)	Exporta lista de regras para o filtro de pacotes
ImportRules(ruleslist)	Importa lista de regras do filtro de pacotes
HashRules(hash, res)	Compara *hash* enviado e obtém resposta em *res*
Getstats(stats)	Obtém estatísticas do *firewall* na estrutura Stat

Tabela 4.1: Operações Providas pela Interface PacketFilter do *Firewall*

Operação	Descrição
SetRule(n, rule)	Especifica regra da posição *n* do NAT
GetRule(n, rule)	Obtém regra da posição *n* do NAT
InsertRule(n, rule)	Insere regra na posição *n* do NAT
DeleteRule(n, rule)	Remove regra da posição *n* do NAT
ExportRules(ruleslist)	Exporta lista de regras para o NAT
ImportRules(ruleslist)	Importa lista de regras do NAT
HashRules(hash, res)	Compara *hash* enviado e obtém resposta em *res*
SendLog(log)	Envia *log* do *firewall* para o concentrador de *logs*

Tabela 4.2: Operações Providas pelas Interfaces NAT e FwLog do *Firewall*

Abaixo são detalhadas cada uma destas operações.

boolean PacketFilter::State() :

Examina o estado do *firewall*, o valor de retorno é uma variável *boolean*, podendo significar:

- verdadeiro : *Firewall* ativado;
- falso : *Firewall* desativado.

void PacketFilter::State(in boolean val) :

Altera o estado do *firewall*, o argumento passado (val) é uma variável *boolean*, que indica:

- verdadeiro : Ativar *firewall*;
- falso : Desativar *firewall*.

A ativação de um *firewall* que já se encontra no estado ativo ou a desativação de um *firewall* que já se encontra no estado desativado não implica em qualquer ação ou mensagem de erro.

void PacketFilter::Restart() :

Reinicia o *firewall*. Esta operação não envia nem retorna argumentos. O propósito dela é que seja possível processar as regras do *firewall* sem a necessidade de desativá-lo e ativá-lo novamente, o que interromperia as conexões existentes. Como muitos *firewalls* suportam esta operação, então ela foi incluída.

Se o *firewall* não suportar tal funcionalidade, deve-se chamar os códigos de *State(0)* e *State(1)* respectivamente.

void PacketFilter::SetRule(in u_short n, in Rule rule) :

Altera uma determinada regra de filtro de pacotes. Ela envia dois argumentos: o número da regra (*unsigned short*) e os dados da regra que estão especificados na estrutura Rule da IDL para *firewalls*.

O número da regra inicia de 1 até o número de regras existentes no *firewall*. Se o número de regra especificado for igual a zero ou maior que o número de regras existentes a operação retorna sem tomar nenhuma ação. Futuramente exceções podem ser geradas para informar o tipo e informações do erro.

- n : Número da regra (1 <= n <= regras existentes);
- Rule : Estrutura Rule definida na IDL para *firewalls*.

void PacketFilter::GetRule(in u_short n, out Rule rule) :

Obtém uma determinada regra de filtro de pacotes, enviando o número da regra (*unsigned short*) como argumento e retornando a regra através de uma referência como definida na estrutura Rule da IDL.

O número da regra inicia de 1 até o número de regras existentes no *firewall*. Se o número de regra especificado for igual a zero ou maior que o número de regras existentes, a operação retorna sem tomar nenhuma ação. Futuramente exceções podem ser geradas para informar o tipo e informações do erro.

- n : Número da regra (1 <= n <= regras existentes);
- Rule : Estrutura Rule definida na IDL para *firewalls*.

void PacketFilter::InsertRule(in u_short n, in Rule rule) :

Insere uma regra no filtro de pacotes, enviando dois argumentos, o número da regra (*unsigned short*) e os dados da regra que estão especificados na estrutura Rule da IDL para *firewalls*.

A regra inserida fica na posição determinada por n e todas as regras seguintes (incluindo a antiga da posição n) são deslocadas uma posição a frente.

O número da regra inicia de 1 até o número de regras existentes mais um (assim é possível inserir depois da última regra), se o número de regra especificado for igual a zero ou maior que o número de regras existentes mais um, a operação retorna sem tomar nenhuma ação. Futuramente exceções podem ser geradas para informar o tipo e informações do erro.

- n : Número da regra (1 <= n <= regras existentes + 1);
- Rule : Estrutura Rule definida na IDL para *firewalls*.

void PacketFilter::DeleteRule(in u_short n) :

Remove uma determinada regra de filtro de pacotes, enviando o número da regra (*unsigned short*) como argumento. Todas as regras seguintes a posição n são deslocadas uma posição para trás.

O número da regra inicia de 1 até o número de regras existentes no *firewall*. Se o número de regra especificado for igual a zero ou maior que o número de regras existentes a operação retorna sem tomar nenhuma ação. Futuramente exceções podem ser geradas para informar o tipo e informações do erro.

- n : Número da regra (1 <= n <= regras existentes).

void PacketFilter::ExportRules(in RulesList ruleslist) :

Envia a lista de todas as regras de filtro de pacotes para o *firewall*, através de um argumento. Esta lista é um tipo sequence<Rule>, como definido na IDL para *firewalls*.

- ruleslist : Lista de regras de filtro de pacotes, é um sequence<Rule>.

void PacketFilter::ImportRules(out RulesList ruleslist) :

Obtém a lista de todas as regras de filtro de pacotes do *firewall*, retornando-a através de uma referência. Esta lista é um tipo sequence<Rule>, como definido na IDL para *firewalls*.

- ruleslist : Lista de regras de filtro de pacotes, é um sequence<Rule>.

void PacketFilter::HashRules(in u_long hash, out boolean res) :

Calcula um *hash* de todas as regras de filtro de pacotes no *firewall* e compara com um *hash* enviado (de regras armazenadas localmente, por exemplo). Retorna através de uma referência uma variável *boolean* verdadeira caso o *hash* seja igual (as regras estejam idênticas), ou falsa caso contrário. Qualquer algoritmo pode ser utilizado para calcular o *hash*, podemos por exemplo, somar todos os campos de cada regra e depois fazer um XOR entre as somas.

- hash : Número de 32 bits (*unsigned long*);
- res : Retorno *boolean* que indica *hash* correto (true) ou errado (false).

void PacketFilter::GetStats(out Stats stats) :

Obtém as estatísticas básicas do *firewall*, retornando através de uma referência a estrutura **Stats** como definida na IDL do *firewall*.

void Nat::SetRule(in u_short n, in Rule rule) :

Altera uma determinada regra de NAT. Envia dois argumentos: o número da regra (*unsigned short*) e os dados da regra, que estão especificados na estrutura Rule da IDL para *firewalls*.

O número da regra inicia de 1 até o número de regras existentes no *firewall*. Se o número de regra especificado for igual a zero ou maior que o número de

regras existentes a operação retorna sem tomar nenhuma ação. Futuramente exceções podem ser geradas para informar o tipo e informações do erro.

- n : Número da regra (1 <= n <= regras existentes);
- Rule : Estrutura Rule definida na IDL para *firewalls*.

void Nat::GetRule(in u_short n, out Rule rule) :

Obtém uma determinada regra de NAT, enviando o número da regra (*unsigned short*) como argumento e retornando a regra através de uma referência como definida na estrutura Rule da IDL.

O número da regra inicia de 1 até o número de regras existentes no *firewall*. Se o número de regra especificado for igual a zero ou maior que o número de regras existentes a operação retorna sem tomar nenhuma ação. Futuramente exceções podem ser geradas para informar o tipo e informações do erro.

- n : Número da regra (1 <= n <= regras existentes);
- Rule : Estrutura Rule definida na IDL para *firewalls*.

void Nat::InsertRule(in u_short n, in Rule rule) :

Insere uma regra de NAT no *firewall*, enviando dois argumentos, o número da regra (*unsigned short*) e os dados da regra que estão especificados na estrutura Rule da IDL para *firewalls*.

A regra inserida fica na posição determinada por n e todas as regras seguintes (incluindo a antiga da posição n) são deslocadas uma posição a frente.

O número da regra inicia de 1 até o número de regras existentes mais um (assim é possível inserir depois da última regra), se o número de regra especificado for igual a zero ou maior que o número de regras existentes mais um, a operação retorna sem tomar nenhuma ação. Futuramente exceções podem ser geradas para informar o tipo e informações do erro.

- n : Número da regra (1 <= n <= regras existentes + 1);
- Rule : Estrutura Rule definida na IDL para *firewalls*.

void Nat::DeleteRule(in u_short n) :

Remove uma determinada regra de NAT do *firewall*, enviando o número da regra (*unsigned short*) como argumento. Todas as regras seguintes a posição n são deslocadas uma posição para trás.

O número da regra inicia de 1 até o número de regras existentes no *firewall*. Se o número de regra especificado for igual a zero ou maior que o número de regras existentes a operação retorna sem tomar nenhuma ação. Futuramente exceções podem ser geradas para informar o tipo e informações do erro.

- n : Número da regra (1 <= n <= regras existentes).

void Nat::ExportRules(in RulesList ruleslist) :

Envia a lista de todas as regras de NAT para o *firewall* como um argumento. Esta lista é um tipo sequence<Rule>, como definido na IDL para *firewalls*.

- ruleslist : Lista de regras de NAT, é um sequence<Rule>.

void Nat::ImportRules(out RulesList ruleslist) :

Obtém a lista de todas as regras de NAT do *firewall*, retornando-a através de uma referência. A lista é um tipo sequence<Rule>, como definido na IDL para *firewalls*.

- ruleslist : Lista de regras de NAT, é um sequence<Rule>.

void Nat::HashRules(in u_long hash, out boolean res) :

Calcula um *hash* de todas as regras de NAT no *firewall* e compara com um *hash* enviado (de regras armazenadas localmente, por exemplo). Retorna através de uma referência uma variável *boolean* verdadeira caso o *hash* seja igual (as regras estejam idênticas), ou falsa caso contrário. Qualquer algoritmo pode ser utilizado para calcular o *hash*. Podemos por exemplo, somar todos os campos de cada regra e depois fazer um XOR entre as somas.

- hash : Número de 32 bits (*unsigned long*);
- res : Retorno *boolean* que indica *hash* correto (true) ou errado (false).

void FwLog::SendLog(in Log log) :

Envia um *log*. Esta operação é a única iniciada pelo *firewall*. Como esta operação, por natureza, pode acontecer muitas vezes em intervalos curtos de tempo, e considerando que um objeto que tenha que processar esta operação também irá processar a mesma vinda de outros *firewalls*, é interessante utilizar a característica do melhor esforço ("*best effort*") ao implementar esta operação. Isto pode ser imposto na IDL ao se utilizar a diretiva *oneway*, que é programada nas implementações de CORBA para utilizar o protocolo UDP ao invés de TCP.

A estrutura **Log** definida na IDL do *firewall* é passada como argumento desta operação.

- log : Estrutura que especifica o *log*, definida na IDL do *firewall*.

4.4.3 Estruturas Utilizadas na IDL para IDSs

Enum IDS::Taction : Define a ação a ser tomada em uma determinada regra do IDS.

- ACT_LOG : Grava localmente os *logs* dos ataques detectados;
- ACT_ALERT : Gera alertas dos ataques detectados;
- ACT_LOGALERT : Grava localmente os *logs* e gera alertas dos ataques detectados.

Enum IDS::Tproto : Define os protocolos suportados.

- PROTO_ALL : Todos os protocolos: IP, TCP, UDP, ICMP;
- PROTO_IP : Protocolo IP;
- PROTO_TCP : Protocolo TCP;
- PROTO_UDP : Protocolo UDP;
- PROTO_TCPUDP : Protocolo TCP e UDP;
- PROTO_ICMP : Protocolo ICMP.

Enum IDS::Tportop : Define as operações que podem ser feitas com uma ou duas portas em uma determinada regra.

- PORT_EQ : A porta é igual ao valor especificado;
- PORT_NE : A porta é diferente do valor especificado;
- PORT_GE : A porta é maior ou igual ao valor especificado;
- PORT_GT : A porta é maior do que o valor especificado;
- PORT_LE : A porta é menor ou igual ao valor especificado;
- PORT_LT : A porta é menor do que o valor especificado;
- PORT_IRG : A porta está entre os dois valores especificados;
- PORT_XRG : A porta está fora da faixa especificada por dois valores.

Enum IDS::Tplszop : Define as operações que podem ser feitas em relação ao tamanho da camada de aplicação do pacote (*payload*).

- PAYLOAD_EQ : Tamanho da camada de aplicação é igual ao valor especificado;
- PAYLOAD_LT : Tamanho da camada de aplicação é menor que o valor especificado;
- PAYLOAD_GT : Tamanho da camada de aplicação é maior que o valor especificado.

Enum IDS::Tipopts : Define os tipos disponíveis no campo de opções do protocolo IP.

- IPOPTS_RR : *Record route*;
- IPOPTS_EOL : *End of list*;
- IPOPTS_NOP : *No op*;
- IPOPTS_TS : *Time stamp*;

- IPOPTS_SEC : *Ip security option*;
- IPOPTS_LSRR : *Loose source routing*;
- IPOPTS_SSRR : *Strict source routing*;
- IPOPTS_SATID : *Stream identifier*.

Struct IDS::Tfields : Define alguns campos que serão utilizados na composição das regras do IDS, pois estes campos não podem ser desabilitados ao se especificar o valor zero. Todos são variáveis *boolean*. O campo será utilizado se o valor for veradeiro e não será utilizado se for falso.

- tos : Tipo de serviço (TOS);
- payload_size : Tamanho da camada de aplicação;
- id : Identificação do pacote;
- ttl : Tempo de vida do pacote (TTL);
- ip_proto : Protocolo que está encapsulado dentro do IP;
- ipopts : Opções do protocolo IP;
- depth : Quantidade de bytes que serão analisados na camada de aplicação;
- offset : Posição a partir da qual a camada de aplicação será analisada;
- tcp_seq : Número de sequência do protocolo TCP;
- tcp_ack : Número de *ack* do protocolo TCP;
- icmp_type : Tipo do pacote ICMP;
- icmp_code : Código do pacote ICMP;
- icmp_id : Número identificador do pacote ECHO do ICMP;
- icmp_seq : Número de sequência do pacote ECHO do ICMP.

Struct IDS::Tfbits : Define os bits de fragmentação do protocolo IP. Todos são variáveis *boolean*, podendo estar ligadas ou não, em qualquer combinação.

- rb : Bit reservado (*Reserved Bit*);
- mf : Mais fragmentos (*More Fragments*);
- df : Não fragmente (*Don't Fragment*).

Struct IDS::Tflags : Define as flags do protocolo TCP. Todas são variáveis *boolean*, podendo estar ligadas ou não, em qualquer combinação.

- syn : Flag SYN;
- ack : Flag ACK;
- rst : Flag RST;
- fin : Flag FIN;
- urg : Flag URG;
- psh : Flag PSH.

Struct IDS::CHost : Define os dados mais comuns de um determinado *host*.

- addr : Endereço IP do *host*, armazenado como um *unsigned long* (32 bits);
- mask : Máscara de rede, armazenada como um número de 0 a 32 (máscara no formato 10.10.0.0/24, onde 24 é a máscara);
- no_ip : Valor *boolean* que indica que o endereço deve ser negado na regra (todos endereços, menos este);
- port_a : Porta, para os protocolos TCP e UDP, armazenado como um *unsigned short* (16 bits);
- port_b : Porta, para os protocolos TCP e UDP (relevante apenas quando especificada uma faixa de portas, PORT_IRG ou PORT_XR G), armazenado como um *unsigned short* (16 bits);
- port_op : Operação lógica na regra com a(s) porta(s). Este tipo de dado é um Tportop.

Struct IDS::NIDS::Rule : Define uma regra do sensor baseado em rede de um IDS.

- state : Valor *boolean* que indica que a regra deve ser processada ou não;
- owner : *String* que contém o nome do administrador que configurou a regra;
- date : *String* que contém a data de criação/alteração da regra;
- comment : *String* com um comentário sucinto a respeito da regra;
- action : Ação a ser feita pela regra. Este tipo é um Taction;
- bidirect : Valor *boolean* que indica que a regra é bidirecional;
- proto : Protocolo utilizado na regra. Este tipo é um Tproto;
- from : Estrutura CHost que especifica o *host* de origem;
- to : Estrutura CHost que especifica o *host* de destino;
- message - Mensagem a ser enviada, relatando o motivo do alerta;
- content : Busca a *string* especificada da camada de aplicação. A *string* pode conter uma representação hexadecimal dos bytes desejados, mas neste caso, eles devem estar contidos entre um par de caracteres "|". Ex.: "|33 FB C4 03 6D|". Esta representação pode ser misturada à *string* normal. Ex.: "bug123|FF 33 C3 6D|attack";
- no_content : O pacote não contém os dados do campo *content*;
- uricontent : Busca a *string* especificada na porção relativa a URI na camada de aplicação;
- no_uricontent : O pacote não contém os dados do campo *uricontent*;
- content_list : Busca as *strings* especificadas em um arquivo cujo nome está em *content_list* na camada de aplicação;
- no_content_list : O pacote não contém os dados do arquivo especificado em *content_list*;

- nocase : Quando especificada como verdadeira, o conteúdo do campo *content* é comparado como se estivesse com letras maiúsculas e minúsculas;
- fragbits_set : Configuração dos bits de fragmentação do protocolo IP. Este tipo é um Tfbits;
- fragbits_use : Configuração que indica quais bits de fragmentação configurados acima devem ser usados. Este tipo é um Tfbits;
- fragbits_or : Valor *boolean* que indica que a operação lógica entre os bits de fragmentação acima é OR e não AND;
- flags_set : Configuração das flags TCP para esta regra. Este tipo é um Tflags;
- flags_use : Configuração que indica quais flags TCP configuradas acima devem ser usadas. Este tipo é um Tflags;
- flags_or : Valor *boolean* que indica que a operação lógica entre as flags acima é OR e não AND;
- rpc : Valor que representa o número da aplicação RPC.;
- tos : Tipo de serviço (campo TOS do protocolo IP);
- payload_size : Tamanho da camada de aplicação;
- payload_op : Operação a ser feita com o tamanho da camada de aplicação especificada e a do pacote. Este tipo é um Tplszop;
- id : Identificação do pacote (campo ID do protocolo IP);
- ttl : Tempo de vida do pacote (campo TTL do protocolo IP);
- ip_proto : Protocolo que está encapsulado dentro do IP;
- no_ip_proto : O pacote não possui o protocolo que está especificado no campo *ip_proto*;
- ipopts : Opções do protocolo IP. Este tipo é um Tipopts;
- depth : Quantidade de bytes que serão analisados na camada de aplicação;
- offset : Posição a partir da qual a camada de aplicação será analisada;

- tcp_seq : Número de sequência do protocolo TCP;
- tcp_ack : Número de *ack* do protocolo TCP;
- icmp_type : Valor de 0 a 255 (octet) que representa o tipo do pacote ICMP;
- icmp_code : Valor de 0 a 255 (octet) que representa o código do pacote ICMP;
- icmp_id : Número identificador do pacote ECHO do ICMP;
- icmp_seq : Número de sequência do pacote ECHO do ICMP;
- used_fields : Campos que serão utilizados nesta regra. Este tipo é um Tfields. Esta informação é utilizada pois alguns campos não podem ser desabilitados colocando-se zero, pois zero é um valor válido para estes campos, assim, estes campos podem ser desabilitados colocando-se a variável *boolean* falsa no campo correspondente na estrutura Tfields.

Struct IDS:NIDS:Group : Define os grupos de regras que compõem o IDS.

- tag : *String* que identifica o grupo;
- comment : *String* que contém uma descrição do grupo.

Struct IDS::NIDS::Stats : Define as estatísticas básicas de um IDS. Os sete primeiros campos são do tipo *unsigned long* (32 bits).

- pkts_recv : Quantidade de pacotes que foram processadas pelo IDS;
- pkts_drop : Quantidade de pacotes que foram descartadas pelo IDS;
- pkts_log : Quantidade de pacotes que ficaram gravadas em *log*;
- pkts_alert : Quantidade de pacotes que foram transmitidas por alerta;
- pkts_tcp : Quantidade de pacotes TCP;
- pkts_udp : Quantidade de pacotes UDP;
- pkts_icmp : Quantidade de pacotes ICMP;

- raw_stats : *String* que contém estatísticas do de forma crua (específica do modelo do IDS).

Struct IDS::IDSAlert::Alert : Define a estrutura de um alerta gerado pelo IDS.

- idsname : *String* que contém um nome único do IDS na rede;
- message : *String* que contém a mensagem de alerta enviada;
- ruleno : Número da regra que gerou o *log* (*unsigned short*, 16 bits);
- usec : Instante do *log*, microsegundos (*unsigned long*, 32 bits);
- sec : Instante do *log*, segundos (octet);
- min : Instante do *log*, minutos (octet);
- hour : Instante do *log*, horas (octet);
- day : Instante do *log*, dia (octet);
- mon : Instante do *log*, mês (octet);
- year : Instante do *log*, ano (*unsigned short*, 16 bits);
- length : Tamanho do pacote, (*unsigned short*, 16 bits);
- ip_src : Porta de origem (*unsigned long*, 32 bits);
- ip_dst : Porta de destino (*unsigned long*, 32 bits);
- ttl : Tempo de vida do pacote (*Time To Live*). Valor de 0 a 255 (octet);
- proto : Protocolo do pacote. Este tipo é um Tproto;
- port_src : Porta de origem (*unsigned short*, 16 bits);
- port_dst : Porta de destino (*unsigned short*, 16 bits);
- tcp_flags : Estado das flags TCP, este tipo é um Tflags;
- tcp_seq : Número de sequência TCP (*unsigned long*, 32 bits);
- tcp_ack : Número de *ack* TCP (*unsigned long*, 32 bits);

- icmp_type : Tipo do pacote ICMP, 0 a 255 (octet);
- icmp_code : Código do pacote ICMP, 0 a 255 (octet).

Tipos de dados definidos (*typedefs*) :

- IDS::IP : Representa um IP, (*unsigned long*, 32 bits);
- IDS::Port : Representa uma porta, (*unsigned short*, 16 bits);
- IDS::u_long : Representa um número de 32 bits, (*unsigned long*);
- IDS::u_short : Representa um número de 16 bits, (*unsigned short*);
- IDS::NIDS::RulesList : Lista de regras de um IDS, (*sequence<Rule>*);
- IDS::NIDS::GroupList : Lista os grupos de regras de um IDS, (*sequence<Group>*).

Descreveremos a seguir o significado de cada uma das operações da interface do IDS, seus argumentos e valores de retorno.

4.4.4 Operações das interfaces do IDS

A Tabela 4.3 descreve de forma resumida as operações providas pelas interfaces do IDS.

Abaixo são detalhadas cada uma destas operações.

boolean NIDS::State() :

Examina o estado do IDS, o valor de retorno é uma variável *boolean* podendo significar:

- verdadeiro : IDS ativo;
- falso : IDS inativo.

void NIDS::State(in boolean val) :

Altera o estado do IDS, o argumento passado (val) é uma variável *boolean*, que indica:

- verdadeiro : Ativar IDS;

Operação	Descrição
State(st)	Liga/Desliga o IDS (st é *boolean*)
State()	Obtém estado do IDS (valor de retorno é *boolean*)
Restart()	Reinicia o IDS
SetRule(n, g, rule)	Específica regra da posição *n*, grupo *g* do NIDS
GetRule(n, g, rule)	Obtém regra da posição *n*, grupo *g* do NIDS
InsertRule(n, g, rule)	Insere regra na posição *n*, grupo *g* do NIDS
DeleteRule(n, g, rule)	Remove regra da posição *n*, grupo *g* do NIDS
ExportRules(g, ruleslist)	Exporta lista de regras para o NIDS
ImportRules(g, ruleslist)	Importa lista de regras do NIDS
ExportGroups(grouplist)	Exporta a lista de grupos para o IDS
ImportGroups(grouplist)	Importa a lista de grupos do IDS
HashRules(hash, res)	Compara *hash* enviado e obtém resposta em *res*
Getstats(stats)	Obtém estatísticas do IDS na estrutura Stats
SendLog(alert)	Envia alerta do IDS para o concentrador de alertas

Tabela 4.3: Operações Providas pelas Interfaces NIDS e IDSAlert do IDS

- falso : Desativar IDS.

A ativação de um IDS que já se encontra no estado ativo ou a desativação de um IDS que já se encontra no estado desativado não implica em qualquer ação ou mensagem de erro.

void NIDS::Restart() :

Reinicia o IDS. Esta operação não envia nem retorna argumentos, o propósito dela é que seja possível processar as regras do IDS sem a necessidade de desativá-lo e ativá-lo novamente, o que interromperia a captura dos pacotes. Como muitos IDSs suportam esta operação então ela foi incluída.

Se o IDS não suportar tal funcionalidade deve-se chamar os códigos de *State(0)* e *State(1)* respectivamente.

void NIDS::SetRule(in u_short n, in u_short g, in Rule rule) :

Altera uma determinada regra do IDS, ela envia três argumentos, o número da regra e o número do grupo (ambos *unsigned short*) e os dados da regra que estão especificados na estrutura Rule da IDL para IDSs.

O número da regra inicia de 1 até o número de regras existentes no IDS, se o número de regra especificado for igual a zero ou maior que o número de regras existentes a operação retorna sem tomar nenhuma ação.

O número do grupo inicia de 1 até o número de grupos existentes no IDS, se o número do grupo especificado for igual a zero ou maior que o número de grupos existentes a operação retorna sem tomar nenhuma ação.

Futuramente exceções podem ser geradas para informar o tipo e informações do erro.

- n : Número da regra (1 <= n <= regras existentes);
- g : Número do grupo (1 <= n <= grupos existentes);
- Rule : Estrutura Rule definida na IDL para IDSs.

void NIDS::GetRule(in u_short n, in u_short g, out Rule rule) :

Obtém uma determinada regra do IDS, enviando o número da regra e do grupo (ambos *unsigned short*) como argumento e retornando a regra através de uma referência como definida na estrutura Rule da IDL.

O número da regra inicia de 1 até o número de regras existentes no IDS, se o número de regra especificado for igual a zero ou maior que o número de regras existentes a operação retorna sem tomar nenhuma ação.

O número do grupo inicia de 1 até o número de grupos existentes no IDS, se o número do grupo especificado for igual a zero ou maior que o número de grupos existentes a operação retorna sem tomar nenhuma ação.

Futuramente, exceções podem ser geradas para informar o tipo e informações do erro.

- n : Número da regra (1 <= n <= regras existentes);
- g : Número do grupo (1 <= n <= grupos existentes);
- Rule : Estrutura Rule definida na IDL para IDSs.

void NIDS::InsertRule(in u_short n, in u_short g, in Rule rule) :

Insere uma regra no IDS, enviando três argumentos, o número da regra, o número do grupo (ambos *unsigned short*) e os dados da regra que estão especificados na estrutura Rule da IDL para IDSs.

A regra inserida fica na posição determinada por n, e todas as regras seguintes (incluindo a antiga da posição n) são deslocadas uma posição a frente.

O número da regra inicia de 1 até o número de regras existentes mais um (assim é possível inserir depois da última regra) , se o número de regra especificado for igual a zero ou maior que o número de regras existentes mais um, a operação retorna sem tomar nenhuma ação.

O número do grupo inicia de 1 até o número de grupos existentes no IDS, se o número do grupo especificado for igual a zero ou maior que o número de grupos existentes a operação retorna sem tomar nenhuma ação.

Futuramente exceções podem ser geradas para informar o tipo e informações do erro.

- n : Número da regra (1 <= n <= regras existentes + 1);
- g : Número do grupo (1 <= n <= grupos existentes);
- Rule : Estrutura Rule definida na IDL para *firewalls*.

void NIDS::DeleteRule(in u_short n, in u_short g) :

Remove uma determinada regra do IDS, enviando o número da regra e o número do grupo (ambos *unsigned short*) como argumentos. Todas as regras seguintes a posição n são deslocadas uma posição para trás.

O número da regra inicia de 1 até o número de regras existentes no IDS, se o número de regra especificado for igual a zero ou maior que o número de regras existentes a operação retorna sem tomar nenhuma ação.

O número do grupo inicia de 1 até o número de grupos existentes no IDS, se o número do grupo especificado for igual a zero ou maior que o número de grupos existentes a operação retorna sem tomar nenhuma ação.

Futuramente exceções podem ser geradas para informar o tipo e informações do erro.

- n : Número da regra (1 <= n <= regras existentes);
- g : Número do grupo (1 <= n <= grupos existentes).

void NIDS::ExportRules(in u_short g, in RulesList ruleslist) :

Envia a lista de todas as regras de um grupo para o IDS, enviando o número do grupo e a lista de regras como argumento, o número do grupo é um *unsigned short* e a lista é um tipo sequence<Rule>, como definido na IDL para IDSs.

- ruleslist : Lista de regras do IDS, é um sequence<Rule>;
- g : Número do grupo (1 <= n <= grupos existentes).

void NIDS::ImportRules(in u_short g, out RulesList ruleslist) :

Obtém a lista de todas as regras de um grupo do IDS, enviando o número do grupo e retornando a lista de regras através de uma referência, o número do grupo é um *unsigned short* e a lista é um tipo sequence<Rule>, como definido na IDL para IDSs.

- ruleslist : Lista de regras do IDS, é um sequence<Rule>;
- g : Número do grupo (1 <= n <= grupos existentes).

void NIDS::ImportGroups(out GroupList grouplist) :

Obtém a lista de todos os grupos do IDS, retornando a lista de grupos através de uma referência, a lista é um tipo sequence<Group>, como definido na IDL para IDSs.

- grouplist : Lista de grupos do IDS, é um sequence<Group>.

void NIDS::ExportGroups(in GroupList grouplist) :

Envia a lista de todos os grupos para o IDS, através de um argumento, a lista é um tipo sequence<Group>, como definido na IDL para IDSs.

- grouplist : Lista de grupos do IDS, é um sequence<Group>.

void NIDS::HashRules(in u_short g, in u_long hash, out boolean res) :

Calcula um *hash* de todas as regras do sensor de rede de um IDS e compara com um *hash* enviado (de regras armazenadas localmente, por exemplo). Retorna através de uma referência uma variável *boolean* verdadeira caso o *hash* seja igual (as regras estejam identicas), ou falsa caso contrário. Qualquer algoritmo pode ser utilizado para calcular o *hash*, podemos por exemplo, somar todos os campos de cada regra e depois fazer um XOR entre as somas.

- hash : Número de 32 bits (*unsigned long*);
- res : Retorno *boolean* que indica *hash* correto (true) ou errado (false).

void NIDS::GetStats(out Stats stats) :

Obtém as estatísticas básicas do IDS, retornando através de uma referência a estrutura **Stats** como definida na IDL do IDS.

void IDSAlert::SendAlert(in Alert alert) :

Envia um alerta. Esta operação é a única iniciada pelo IDS. Como esta operação, por natureza, pode acontecer muitas vezes em intervalos curtos de tempo, e considerando que um objeto que tenha que processar esta operação também irá processar a mesma vinda de outros IDSs, é interessante utilizar a característica do melhor esforço ("*best effort*") ao implementar esta operação. Isto pode ser imposto na IDL ao se utilizar a diretiva *oneway*, que é programada nas implementações de CORBA para utilizar o protocolo UDP ao invés de TCP.

A estrutura **Alert** definida na IDL do IDS é passada como argumento desta operação.

- alert : Estrutura que especifica o alerta, definida na IDL do IDS.

Capítulo 5

SIGSEC - Implementação do Protótipo

Este capítulo descreve detalhes da implementação de um protótipo que utiliza o modelo e a API descritas no capítulo anterior. Primeiro nos concentramos no objetivo do protótipo, depois são justificadas as escolhas de software e então é descrito o funcionamento do sistema. Adicionalmente são apresentadas algumas observações e detalhes úteis da organização do código fonte do protótipo.

5.1 Objetivo

O objetivo do SIGSEC é gerenciar vários *firewalls* e IDSs diferentes de uma forma padronizada, permitindo maior clareza na definição de regras e análise de *logs*. Além disso, o sistema centraliza os *logs* das diversas ferramentas em uma base de dados única que pode ser monitorada. Ainda, devido à padronização, é possível (mas não implementada) a interoperabilidade destas ferramentas, permitindo por exemplo, que um determinado *firewall* atualize regras automaticamente em função de alertas detectados de um determinado IDS.

Para o desenvolvimento deste protótipo foi escolhido um sistema operacional, um *firewall*, um IDS, um banco de dados SQL e uma implementação de CORBA compatíveis com o sistema.

Foram então desenvolvidos os *drivers* para o *firewall* (OpenBSD PF) e para o IDS (Snort), o SIGSEC Log Daemon e o *script* CGI que permite que clientes se conectem via Web para gerenciar o sistema.

Esta implementação está totalmente baseada no modelo proposto no capítulo anterior e é também um guia para o desenvolvimento de *drivers* e aplicativos para outros sistemas operacionais e ferramentas de segurança.

Um outro objetivo deste protótipo, não menos importante, é mostrar a praticidade do modelo proposto no capítulo anterior.

5.2 Considerações Sobre a Plataforma e Softwares Utilizados

Listaremos agora a plataforma e softwares utilizados, fazendo algumas considerações sobre a escolha e facilidades providas por cada um.

Todos os softwares (incluindo o sistema operacional) utilizadados são *Open Source*, ou seja, são de domínio público, podendo ser utilizados, alterados e até comercializados na maioria dos casos.

Sistema Operacional - OpenBSD : O OpenBSD é considerado por muitos como um dos sistemas operacionais mais seguro disponível de forma gratuita. Não só o código fonte do OpenBSD, mas também os principais aplicativos são auditados continuamente em busca de bugs e falhas de segurança.

O *site* do projeto do OpenBSD fica em *http://www.openbsd.org* e é mantido por voluntários com sede no Canadá, onde as regras de exportação de criptografia são menos rígidas.

O OpenBSD foi escolhido também por suportar todos requisitos de software necessários e por ser mais "enxuto" que outros sistemas de código aberto (Linux, por exemplo). A compatibilidade com outros sistemas BSD e outro sistemas UNIX também foi um fator importante.

A versão utilizada inicialmente foi a 3.0. Recentemente foi lançada a versão 3.1 na qual o protótipo não apresentou problema algum. Como o SIGSEC não utiliza praticamente nenhuma característica particular do OpenBSD a instalação em outro ambiente UNIX requer mínimas alterações.

Implementaçao de CORBA - Mico : O Mico é uma implementação *Open Source* do padrão CORBA, a filosofia do Mico é reforçar compatibilidade e clareza em detrimento da velocidade. Como a velocidade não é um fator crucial na maioria das operações da interface e o seu uso no OpenBSD se mostrou bastante satisfatório ele foi a implementação escolhida.

O *site* do projeto fica em *http://www.mico.org*, a versão atual é a 2.3.7 e o protótipo tem funcionado sem alterações desde o início de seu desenvolvimento,

da versão 2.3.5 até a versão atual.

Uma grande ajuda do aprendizado e desenvolvimento em CORBA usando o Mico foi obtido na documentação que acompanha o software e no livro [47] publicado por seus desenvolvedores.

Inicialmente o Mico não havia sido testado no OpenBSD. Como era objetivo o uso do OpenBSD e o Mico apresentava vantagens para o desenvolvimento, foi decidido testar esta implementação de CORBA. Foram necessários vários ajustes (*patches*) no código fonte do Mico para que este compilasse e funcionasse corretamente no OpenBSD. O conjunto destas adaptações (chamado *port*) foi submetido à equipe do OpenBSD, e após alguns meses e novas correções o *port* foi tornado oficial. Hoje, o *port* oficial do Mico no OpenBSD é mantido pelo autor desta monografia, e o código fonte das adaptações pode ser acessado na Web a partir do endereço: *http://www.openbsd.org/cgi-bin/cvsweb/ports/devel/mico*.

Servidor Web - Apache Web Server : O servidor Web Apache foi utilizado porque além de ser gratuito, possui ótimo desempenho, é de fato, o servidor Web mais utilizado na Internet, como o mostra o gráfico de uma pesquisa da Netcraft em Abril de 2002 (Figura 5.1).

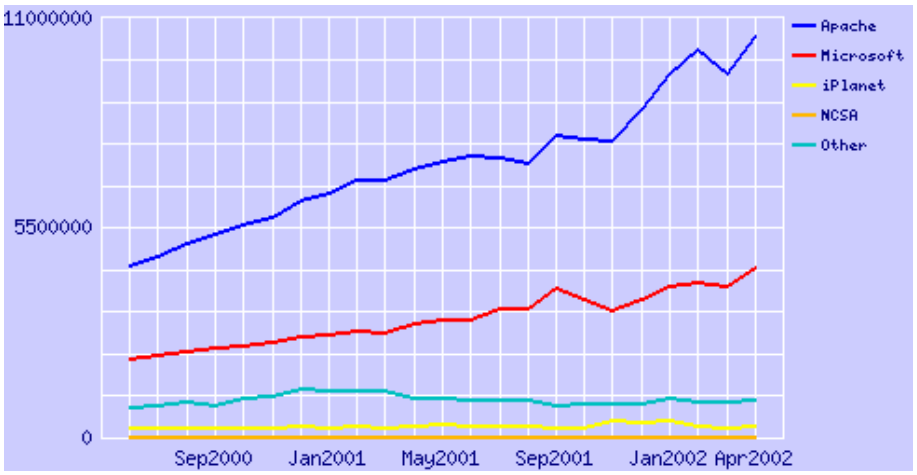


Figura 5.1: Pesquisa da Netcraft de Utilização de Servidores Web

O Apache também apresentou a facilidade de já estar disponível na instalação padrão do OpenBSD. O site do projeto fica em *http://www.apache.org*.

Gerenciador de Banco de Dados - MySQL : O SGBD MySQL é um dos

gerenciadores de bancos de dados SQL mais utilizados no mundo do *Open Source*, sua instalação e customização são muito simples, e como ele ja havia sido "portado" para o OpenBSD foi a escolha mais natural.

Firewall - OpenBSD PF : O OpenBSD PF é o *firewall* nativo do OpenBSD, com suporte a filtro de pacotes e NAT, o ideal para o teste do protótipo. No início do desenvolvimento o OpenBSD utilizava o IPF (*http://www.ipfilter.org*), mas devido a restrições de licença impostas pelo IPF o projeto do OpenBSD resoveu desenvolver seu próprio *firewall*, o PF.

A transição durante o desenvolvimento do protótipo do *driver* do IPF para o OpenBSD PF foi muito simples pois este manteve quase total compatibilidade a nivel de regras com o anterior.

IDS - Snort : O Snort é um sistema de detecção de intrusão baseado em rede, gratuito e muito utilizado, possuindo uma comunidade de usuários que contribuem continuamente com melhorias, *plugins* e atualizações de regras de detecção de ataques.

A instalação e a configuração do Snort no OpenBSD é muito simples pois este já faz parte do conjunto de pacotes disponíveis no OpenBSD a bastante tempo.

As regras de detecção de ataques do Snort podem ser divididas em grupos, por categoria de ataque, Ex.: DoS, Web CGI, FTP, etc.

Por estas características e pelo fato do Snort ser altamente customizável foi a escolha ideal para o protótipo.

Compilador - GNU g++ : A linguagem de programação utilizada foi o C++, para todo o sistema, até mesmo o CGI de gerência foi escrito em C++. Como o ORB escolhido foi o Mico, a linguagem de programação não teve possibilidades de escolha, visto que C++ é a única linguagem suportada por este ORB.

O compilador C++ padrão disponível no OpenBSD é o GNU g++, a versão utilizada foi a 2.95.3.

Ferramentas auxiliares - Lex e Yacc : O Lex e o Yacc são poderosas ferramentas (utilizadas até mesmo para programar compiladores) que foram utilizadas para fazer análise léxica e sintática das regras de IDSs e *firewalls*.

O **Lex** permitiu que se determinasse todo o conjunto de palavras chaves que podiam aparecer em um determinado conjunto de regras, especificando valores de retorno e identificando números, *strings*, etc.

O **Yacc** permitiu a análise da combinação das palavras-chave ao formar uma regra, proporcionando a interpretação desta regra e traduzindo-a para o formato padronizado nas estruturas da IDL.

O esforço no estudo do Lex e do Yacc foi muito recompensador, pois o tratamento das regras (que podem se apresentar em inúmeras possibilidades) se tornou muito mais consistente e confiável.

5.3 Descrição do Funcionamento do Sistema

Utilizando como base o capitulo anterior, listamos os componentes que devem fazer parte do protótipo:

- *Driver* CORBA para o *firewall* (OpenBSD PF);
- *Driver* CORBA para o IDS (Snort);
- Concentrador de *logs* (SIGSEC *Log Daemon*);
- CGI em CORBA para administração (SIGSEC CGI);
- Servidor Web (Apache);
- Banco de dados SQL (MySQL);
- Servidor de nomes do CORBA (Mico *Naming Service*);
- Clientes Web (Navegadores: Netscape, IE etc).

A Figura 5.2 mostra um diagrama lógico onde podem ser vistas as interações entre os diversos componentes do protótipo. O *Naming Service* foi propositalmente retirado do diagrama a fim de não complicar o entendimento. Esta omissão não prejudica de forma alguma as considerações seguintes, pois o *Naming Service* é uma aplicação comum, utilizado por todos os componentes CORBA no sistema para encontrar outros componentes.

Os *drivers* de *firewall* e IDS funcionam como servidores CORBA, permitindo que o *script* em CGI (SIGSEC CGI) realize as operações de configuração da interface,

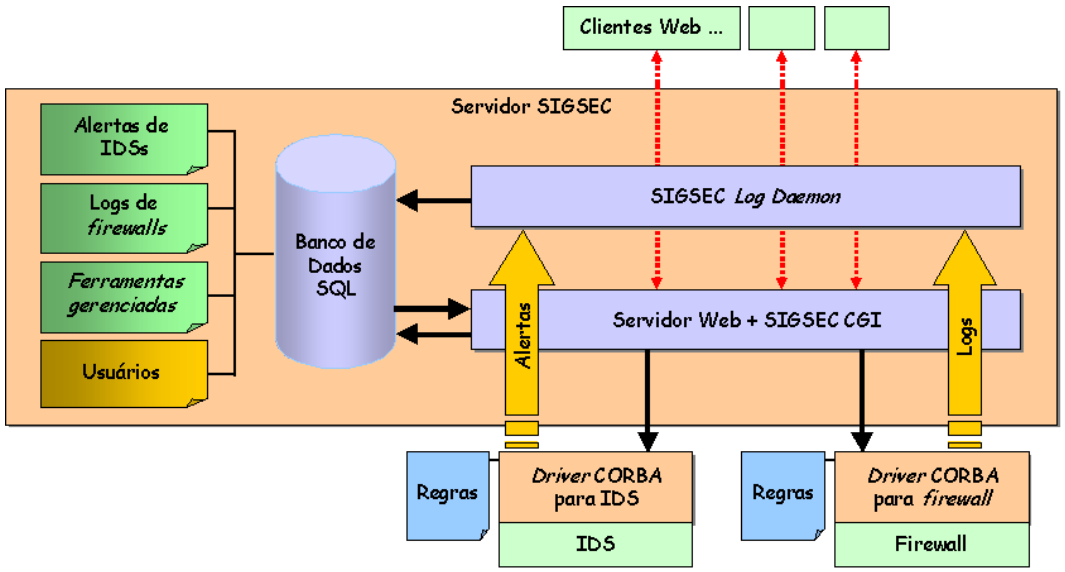


Figura 5.2: Diagrama Lógico do Protótipo SIGSEC

tais como: *Start()*, *SetRule()*, etc. (representadas pelas setas escuras descendentes na Figura 5.2). Estas operações, são disparadas pelos clientes Web que utilizam o CGI através do servidor Web (representadas pelas setas pontilhadas descendentes na Figura 5.2).

Os *drivers* de *firewall* e IDS também funcionam como clientes CORBA, realizando operações no concentrador de *logs* (SIGSEC *Log Daemon*). Estas operações são os envios dos *logs* e alertas (representadas pelas setas grossas ascendentes na Figura 5.2).

O SIGSEC *Log Daemon* é portanto, um servidor CORBA, que recebe os *logs* e alertas dos *firewalls* e IDSs e os envia ao banco de dados SQL (MySQL).

O SIGSEC CGI também é utilizado pelos clientes Web para verificar os *logs*, alertas e outras informações diretamente no banco de dados SQL, podendo eventualmente alterar estes dados.

O banco de dados SQL (MySQL) é reponsável por manter em tabelas distintas as seguintes informações:

- Alertas de vários IDSs;
- *Logs* de vários *firewalls*;
- Lista das ferramentas (*firewalls* e IDSs) para relacionamento com as tabelas anteriores;

- Lista dos usuários e seus privilégios de acesso no sistema.

Quando um *firewall* ou IDS inicia seu funcionamento ele primeiramente se registra no *Naming Service*, e então registra seus objetos e monitora continuamente até que ocorra um evento. Este evento pode ser uma operação vinda do SIGSEC CGI ou um *log* ou alerta que deve ser enviado, neste segundo caso, o *log* ou alerta é enviado para o SIGSEC *Log Daemon*, este, antes de armazenar, verifica se o *firewall* ou IDS está na lista de ferramentas, se não está ele é inserido.

Uma operação realizada pelo SIGSEC CGI é quase sempre precedida por uma consulta no *Naming Service* para encontrar a localização do objeto desejado. Apenas as consultas diretas ao MySQL não se utilizam do *Naming Service* nem CORBA, e nestes casos, a lista de ferramentas criada no MySQL é útil para se monitorar todas as ferramentas.

É importante notar que, apesar do protótipo suportar apenas um tipo de *firewall* (OpenBSD PF) e um IDS (Snort), ele pode gerenciar várias destas ferramentas simultaneamente.

5.4 Organização e Detalhes do Código Fonte

No capítulo anterior, após apresentada arquitetura do modelo de gerência, verificamos a necessidade de implementação de quatro componentes de software, são eles:

- *Driver* CORBA para o *firewall* (OpenBSD PF);
- *Driver* CORBA para o IDS (Snort);
- Concentrador de *logs* (SIGSEC *Log Daemon*);
- CGI em CORBA para administração (SIGSEC CGI).

Outros códigos auxiliares também são necessários:

- Os arquivos IDL com as padronizações para *firewalls* e IDSs;
- *Script* SQL com as definições do banco de dados;
- *Makefiles* para agilizar o processo de compilação;

- *Scripts* para inicializar todo o sistema e o *Naming Service*;
- Página HTML inicial para o SIGSEC CGI;
- Arquivo de configuração do sistema.

O código fonte foi dividido nos seguintes diretórios:

IDLs : Arquivos das IDLs padronizadas para *firewalls* e IDSs;

PF_Driver : Código fonte do *driver* do *firewall* OpenBSD PF;

Snort_Driver : Código fonte do *driver* do IDS Snort;

Tools : Código fonte do SIGSEC CGI, SIGSEC *Log Daemon*, *scripts* SQL e de inicialização.

Os diretórios acima serão examinados separadamente, especificando o seu conteúdo. O código fonte parcial pode ser visto no apêndice C.

5.4.1 IDLs

Neste diretório existem os arquivos:

`sigsec_fw.idl` : IDL padronizada para *firewalls*;

`sigsec_ids.idl` : IDL padronizada para IDSs;

`Makefile` : Arquivo usado para compilar as IDLs através do comando `make`.

5.4.2 PF_Driver

Neste diretório encontramos:

`sigsec_pfd.cc` : Programa principal do *driver* do OpenBSD PF;

`sigsec_pfd.h` : Classes, arquivos de inclusão e protótipos de funções para o arquivo `sigsec_pfd.cc`;

`sigsec_fw_impl.cc` : Implementação dos objetos do *driver* do OpenBSD PF;

`sigsec_fw_impl.h` : Classes, arquivos de inclusão e protótipos de funções para o arquivo `sigsec_fw_impl.cc`;

`Parser/` : Diretório que contém os arquivos: `scanner.l`, `parser.y` e `Makefile` que são responsáveis pela tradução das regras do OpenBSD PF para o formato padronizado pela IDL;

`Builder/` : Diretório que contém os arquivos: `builder.cc`, `builder.h` e `Makefile` que são responsáveis pela tradução do formato padronizado pela IDL para regras do OpenBSD PF;

`Logger/` : Diretório que contém os arquivos: `logger.cc` e `Makefile` que são responsáveis pela monitoração do sistema e envio dos *logs* para o SIGSEC *Log Daemon*;

`Makefile` : Arquivo usado para compilar o *driver* do OpenBSD PF através do comando `make`.

5.4.3 Snort_Driver

Neste diretório encontramos:

`sigsec_snortd.cc` : Programa principal do *driver* do Snort;

`sigsec_snortd.h` : Classes, arquivos de inclusão e protótipos de funções para o arquivo `sigsec_snortd.cc`;

`sigsec_ids_impl.cc` : Implementação dos objetos do *driver* do Snort;

`sigsec_ids_impl.h` : Classes, arquivos de inclusão e protótipos de funções para o arquivo `sigsec_ids_impl.cc`;

`Parser/` : Diretório que contém os arquivos: `scanner.l`, `parser.y` e `Makefile` que são responsáveis pela tradução das regras do Snort para o formato padronizado pela IDL;

`Builder/` : Diretório que contém os arquivos: `builder.cc`, `builder.h` e `Makefile` que são responsáveis pela tradução do formato padronizado pela IDL para regras do Snort;

`Logger/` : Diretório que contém os arquivos: `logger.cc` e `Makefile` que são responsáveis pela monitoração do sistema e envio dos *alertas* para o SIGSEC *Log Daemon*;

`Makefile` : Arquivo usado para compilar o *driver* do Snort através do comando `make`.

5.4.4 Tools

Neste diretório encontramos:

`sigsec_logd.cc` : Programa principal do SIGSEC *Log Daemon*;

`sigsec_logd.h` : Classes, arquivos de inclusão e protótipos de funções para o arquivo `sigsec_logd.cc`;

`sigsec_logd_impl.cc` : Implementação dos objetos do SIGSEC *Log Daemon*;

`sigsec_logd_impl.h` : Classes, arquivos de inclusão e protótipos de funções para o arquivo `sigsec_logd_impl.cc`;

`sigsec_cgi.cc` : Programa principal do SIGSEC CGI;

`sigsec_cgi.h` : Classes, arquivos de inclusão e protótipos de funções para o arquivo `sigsec_cgi.cc`;

`CGILib.cc` : Código principal da biblioteca CGI utilizada;

`CGILib.h` : Classes, arquivos de inclusão e protótipos de funções para o arquivo `CGILib.cc`;

`sigsec.conf` : Arquivo de configuração do sistema SIGSEC;

`sigsec.html` : Página HTML inicial de administração;

`sigsec.sql` : *Script* SQL para a criação do banco de dados;

`ctrl_logd` : *Script* de inicialização do SIGSEC *Log Daemon*;

`ctrl_snortd` : *Script* de inicialização do *driver* do Snort;

`ctrl_pfd` : *Script* de inicialização do *driver* do OpenBSD PF;

`start_nsd` : *Script* de inicialização do *Naming Service*;

`start_all` : *Script* de inicialização de todo o sistema;

`Makefile` : Arquivo usado para compilar todos componentes de software acima.

5.4.5 Obtendo o código fonte

O código fonte de todo o protótipo totaliza:

- 44 arquivos;
- 6537 linhas;
- 168334 caracteres.

O que poderia equivaler a mais de 100 páginas de código fonte. No apêncice C foram incluídos apenas os programas principais do código do protótipo. O código fonte completo pode ser obtido via Internet no endereço: *http://www.sigsec.org*.

5.5 Instalação e Utilização

5.5.1 Requerimentos

SIGSEC (Sistema Integrado de Gerência de Segurança Empregando CORBA):
sisgec-0.6.tar.gz.
Sigsec (*http://www.sigsec.org*).

ORB (Implementação do CORBA):
Qualquer uma.
Testado: Mico (*http://www.mico.org*).

SGDB (Gerenciador de Banco de Dados SQL) *Opcional
O banco de dados é utilizado apenas pelo concentrador de *logs*, suportando atualmente o MySQL.
Testado: MySQL (*http://www.mysql.com*).

Sistema Operacional :
Windows ou UNIX (teoricamente qualquer sistema operacional que suporte uma implementação de CORBA).
Testado: OpenBSD (*http://www.openbsd.org*).

Plataforma de hardware :
Teoricamente qualquer uma que rode um sistema operacional que suporte uma implementação de CORBA.
Testado: PC.

Firewalls e IDSs :

Atualmente o SIGSEC suporta:

- OpenBSD PF (*firewall* nativo do OpenBSD);
- Snort, famoso IDS baseado em rede (*http://www.snort.org*).

Obs.: Para desenvolvimento, as IDLs podem (teoricamente) serem usadas com qualquer *firewall* ou IDS.

É necessário o compilador C++ se está instalando os fontes para serem compilados. Também serão necessários o `lex` e o `yacc` (todos presentes na instalação completa do OpenBSD).

5.5.2 Instalando

Descompactar o arquivo sigsec-0.6.tar.gz :

```
host# tar -xzvf sigsec-0.6.tar.gz
host# cd sigsec-0.6
```

Compilar:

```
host# make
```

Serão criados:

- sigsec_pfd : *Driver* para o *firewall* do OpenBSD (PF);
- sigsec_snortd : *Driver* para o Snort;
- sigsec_logd : Concentrador de *logs* de firewalls e IDSs;
- sigsec_cgi : CGI que permite o gerenciamento de *firewalls* e IDSs.

Arquivos auxiliares:

- start_nsd : *Script* para iniciar o *Naming service*;
- ctrl_logd : *Script* para controlar o concentrador de *logs*. Opções: start | stop;
- ctrl_snortd : *Script* para controlar o *driver do snort*. Opções: start | stop;
- ctrl_pfd : *Script* para controlar o *driver* do PF. Opções: start | stop;

- start_all : *Script* para iniciar na ordem: MySQL, *Naming service*, concentrador de *logs*, PF *driver* e snort *driver*;
- sigsec.conf : Arquivo de configuração do SIGSEC;
- sigsec.html : Página inicial de administração.

Instale:

```
host# make install
```

- Os *drivers* e o concentrador serão instalados em /usr/local/bin;
- Os *scripts* de controle estarão também em /usr/local/bin;
- O sigsec_cgi será instalado em /var/www/cgi-bin;
- O sigsec.conf será instalado em /etc;
- O sigsec.html será instalado em /var/www/htdocs.

5.5.3 Configurando e Testando o Sistema

Obs 1: Os *scripts* de inicialização e controle contém as seguintes opções para o ORB:
- A referência para o *Naming Service* (*host* e porta)
- O *host* e a porta do próprio programa
Eles podem ser editados para se apropriarem às necessidades do sistema.

Obs 2: O arquivo de configuração sigsec.conf pode conter uma série de opções (algumas obrigatórias) que serão comentadas a seguir. Ele deve conter uma opção por linha, as opções correspondem a uma *tag* seguida do sinal de igual e o valor correspondente, sem espaços ou tabulações.
Ex.:
NAMING_PORT=9000

Iniciando o serviço de Naming do CORBA:

O serviço de *Naming* está disponível em praticamente todas as implementações do padrão CORBA. No Mico ele é ativado pelo programa chamado nsd (*Naming*

Service Daemon).

Sintaxe:
nsd -ORBIIOPAddr inet:host:9000 &

Onde:
host : deve ser substituído pelo nome da sua máquina.
9002 : porta desejada.

Esta é a sintaxe para o caso do Mico. Existe um *script* chamado start_nsd para iniciar o *Naming* de forma simples (o nome do *host* é obtido automaticamente):
`host# start_nsd`

Iniciando o concentrador de *logs* de firewalls e IDS do SIGSEC:

Este aplicativo é responsável por receber *logs* e alertas de *firewalls* e IDSs, ele registra um contexto no *Naming* chamado "SigSecLog" e dois objetos neste contexto: FwLog e IDSAlert.
O sigsec_logd, como é chamado, recebe então os *logs* e alertas e os armazena em um banco de dados SQL (MySQL).

Sintaxe:
`host# sigsec_logd`

Existe um *script* chamado ctrl_logd que pode ser usado para iniciar e parar o processo de forma simples:
`host# ctrl_logd start` (Inicia o processo concentrador de *logs*)
`host# ctrl_logd stop` (Termina o processo concentrador de *logs*)

O sigsec_logd coloca o seu **pid** no arquivo /var/run/sigsec_logd.pid, que é então utilizado pelo *script* de controle. Pode ser necessário remover este arquivo manualmente em caso de problemas.

Este serviço usa em /etc/sigsec.conf as opções:

- LOGD_PORT : Porta que o sigsec_logd está associado;
- DB_HOST : Máquina onde está instalado o MySQL;
- DB_USER : Usuário que tem a permissão para manipular o banco de dados do SIGSEC;
- DB_PASSWORD : Senha do banco de dados do SIGSEC;
- NAMING_HOST : Máquina onde está sendo executado o *Naming Service*;
- NAMING_PORT : Porta onde está sendo executado o *Naming Service*.

O parâmetro DB_PASSWORD é obrigatório, as outras cinco opções possuem os valores padrão:

- LOGD_PORT=9003
- DB_HOST=localhost
- DB_USER=root
- NAMING_HOST=<Nome da máquina local obtido automaticamente>
- NAMING_PORT=9000

Iniciando o *driver* do Firewall PF:

Este aplicativo é responsavel por fazer a interface entre o OpenBSD PF e o SIGSEC, provendo métodos para alteração das regras do filtro de pacotes, NAT e enviando Logs ao concentrador de *logs*.

Sintaxe:
`host# sigsec_pfd`

Existe um *script* chamado ctrl_pfd que pode ser usado para iniciar e parar o processo de forma simples:
`host# ctrl_pfd start` (Inicia o processo *driver* do PF)
`host# ctrl_pfd stop` (Termina o processo *driver* do PF)

O sigsec_pfd coloca o seu **pid** no arquivo /var/run/sigsec_pfd.pid, que é então utilizado pelo *script* de controle. Pode ser necessário remover este arquivo manualmente em caso de problemas.

Este serviço usa em /etc/sigsec.conf as opcoes:

- PF_NAME : Nome deste *firewall* (indicativo único do *firewall* na rede);
- PF_PORT : Porta que o sigsec_pfd está associado;
- PF_CONF : Arquivo de configuração das regras de filtro de pacotes;
- NAT_CONF : Arquivo de configuração das regras de NAT;
- FIX_IFACE : Nome da interface de rede que se deseje refrenciar pelo IP fixo 255.255.255.255, esta opção é um "*hack*" que permite que se utilize uma interface de IP dinâmico (Ex. tun0) não sendo necessário alterar as regras do *firewall* sempre que a interface adquirir um novo IP. Resumindo: Usando o IP 255.255.255.255 para referenciar esta interface é o mesmo que usar FIX_IFACE (que pode ser tun0 por exemplo);
- NAMING_HOST : Máquina onde está sendo executado o *Naming Service*;
- NAMING_PORT : Porta onde está sendo executado o *Naming Service*.

O parâmetro PF_NAME é obrigatório, as outras seis opções possuem os valores padrão:

- PF_PORT=9001
- PF_CONF=/etc/pf.conf
- NAT_CONF=/etc/nat.conf
- FIX_IFACE=NO
- NAMING_HOST=<Nome da máquina local obtido automaticamente>
- NAMING_PORT=9000

Iniciando o driver do Snort:

Este aplicativo é responsável por fazer a interface entre o Snort e o SIGSEC, provendo métodos para alteração das regras do IDS e enviando alertas ao concentrador de *logs*.

Sintaxe:
`host# sigsec_snortd`

Existe um *script* chamado ctrl_snortd que pode ser usado para iniciar e parar o processo de forma simples:
`host# ctrl_snortd start` (Inicia o processo *driver* do snort)
`host# ctrl_snortd stop` (Termina o processo *driver* do snort)

O sigsec_snortd coloca o seu **pid** no arquivo /var/run/sigsec_snortd.pid, que é então utilizado pelo *script* de controle. Pode ser necessário remover este arquivo manualmente em caso de problemas.

Este serviço usa em /etc/sigsec.conf as opções:

- SNORT_NAME : Nome deste IDS (indicativo único do IDS na rede);
- SNORT_PORT : Porta que o sigsec_snortd está associado;
- SNORT_PATH : Caminho completo onde se encontra o executável do snort;
- SNORT_CONF : Diretório onde estão as regras e o snort.conf;
- SNORT_ALERT : Caminho completo onde se encontra o arquivo de alertas do snort;
- NAMING_HOST : Máquina onde está sendo executado o *Naming Service*;
- NAMING_PORT : Porta onde está sendo executado o *Naming Service*.

O parâmetro SNORT_NAME é obrigatório, as outras seis opções possuem os valores padrão:

- SNORT_PORT=9002

- SNORT_PATH=/usr/local/bin/snort
- SNORT_CONF=/etc/snort
- SNORT_ALERT=/var/log/snort/alert
- NAMING_HOST=<Nome da máquina local obtido automaticamente>
- NAMING_PORT=9000

Usando a interface de gerência Web:

A interface de gerência Web está implementada através de um CGI escrito em C++, chamado sigsec_cgi. Este CGI obtém a localização do *Naming Service* lendo o arquivo de configuração sigsec.conf. O arquivo sigsec.html é a página inicial de administração.

O programa sigsec_cgi deve ser colocado no diretorio cgi-bin do servidor, no caso do OpenBSD: /var/www/cgi-bin.

O sigsec.html deve ser colocado em qualquer sub-diretório do servidor Web, no caso do OpenBSD: /var/www/htdocs.

Resumo:
- sigsec.html : Página inicial.
- sigsec_cgi : CGI.

Este serviço usa em /etc/sigsec.conf as opções:

- DB_HOST : Máquina onde está instalado o MySQL;
- DB_USER : Usuário que tem a permissão para manipular o banco de dados do SIGSEC;
- DB_PASSWORD : Senha do banco de dados do SIGSEC;
- NAMING_HOST : Máquina onde está sendo executado o *Naming Service*.

O parâmetro DB_PASSWORD é obrigatório, as outras três opções possuem os valores padrão:

- DB_HOST=localhost

- DB_USER=root
- NAMING_HOST=<Nome do host local obtido automaticamente>
- NAMING_PORT=9000

A interface de gerência pode ser acessada digitando-se a URL:
http://suamaquina.com.br/sigsec.html

Configurando a autenticação:
Deve ser criado um aquivo .htaccess com o seguinte conteúdo:

```
AuthName ‘‘SIGSEC Web Admin’’
AuthType Basic
AuthUserFile /var/www/.htpasswd
Require user andre
```

Este arquivo (.htaccess) deve ser colocado no seu diretório cgi-bin (no OpenBSD /var/www/cgi-bin). O arquivo .htpasswd é gerado pelo comando:
`htpasswd -mc .htpasswd andre` (usuário andre, senha será digitada).

E pode ser colocado em qualquer lugar fora da raiz do servidor Web (Ex.: /var/www).

Este tipo de configuração implica em modificar o arquivo de configuração do apache, httpd.conf (no OpenBSD, /var/www/conf/httpd.conf) e alterar a diretiva “AllowOverride None” para “AllowOverride All” abaixo da linha <Directory “/var/www/cgi-bin”>, no caso do OpenBSD.

Uma melhor solução, seria a criação de um diretório específico no servidor web, com diretivas ScriptAlias, e proteção .htaccess próprias, ou mesmo um VirtualHost, a fim de não ocupar todo o cgi-bin do servidor para o SIGSEC.

Iniciando todo o SIGSEC na inicialização do sistema

Todos os *drives*, *Naming Service*, MySQL e concentrador de *logs* podem ser inicializados através do *script* start_all.

Para colocar na inicialização do sistema coloque o conteúdo do arquivo start_all no arquivo rc.local, no caso do OpenBSD:

```
host# cat /usr/local/bin/start_all >> /etc/rc.local
```

As linhas acrescentadas no rc.local devem ser alteradas de acordo com as necesidades do sistema.

5.6 Testes e Avaliação do Protótipo

O protótipo SIGSEC foi testado nas versões 3.0 e 3.1 (atual) do OpenBSD, utilizando 2 PCs como servidores e vários PCs como clientes Web.

Servidor 1 : Pentium III 800MHz, OpenBSD 3.1;

Servidor 2 : Pentium III 600MHz, OpenBSD 3.0;

Clientes Web : Diversos PCs, com Windows 98 e Windows 2000, usando Netscape e Internet Explorer.

Os seguintes módulos de software foram testados:

- *Driver* para IDS (Snort);
- *Driver firewall* (OpenBSD PF);
- SIGSEC *Log Daemon*;
- SIGSEC CGI;
- *Naming Service*.

Diversas combinações dos módulos descritos anteriormente foram instalados nos servidores 1 e 2. Sendo que o *Naming Service* foi até mesmo testado em uma máquina Linux separada.

Em todos os testes o protótipo não apresentou nenhum erro de execução das operações da API. Os tempos de execução de algumas operações, do ponto de vista da interface de gerenciamento, podem ser visualizados no gráfico da Figura 5.3.

As medidas deste gráfico foram obtidas com o sistema funcionando em uma rede local com tráfego normal, isto é, em condições reais. As operações apresentadas pertencem a *firewalls* ou IDSs, em ambas os resultados foram muito similares.

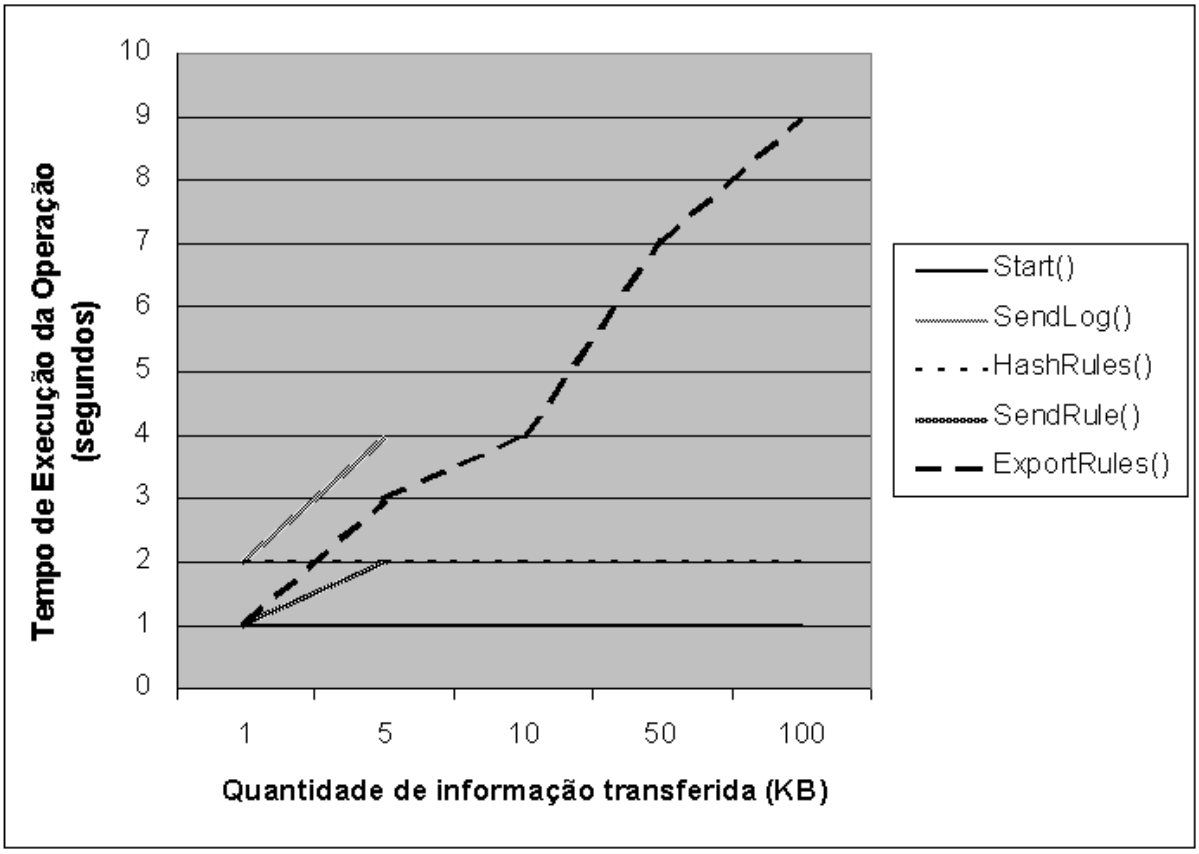


Figura 5.3: Tempos de execução de algumas operações da API

A implementação do CGI utilizando C++ ajudou bastante neste aspecto, pois se fosse utilizada uma linguagem interpretada (Ex. Perl) os tempos de execução seriam maiores.

A eficiência do uso do modelo não pode ser medida em função dos **tempos de execução** das operações da API, estes, podem variar muito, dependendo da quantidade de regras ou *logs* que estão sendo verificados. A eficiência do modelo está na **forma** como as operações da API estão montadas e como elas podem ser combinadas.

A Tabela 5.1 apresenta as operações e suas vantagens em termos de interoperabilidade. A intersecção entre as linhas e colunas mostra as possíveis funcionalidades entre *firewalls* e IDSs quando se utilizam duas operações da API em objetos distintos na rede. Ex.: Interoperabilidade entre *firewalls* e IDSs é conseguida quando

se utiliza a operação SendAlert() do IDSs para o sistema central (detecção de um ataque) e a operação SetRule() da central para o *firewall* (reconfiguração automática de regra).

		IDS			Firewall		
		SendAlert()	SendRule() GetRule()	ExportRules() ImportRules()	SendLog()	SendRule() GetRule()	ExportRules() ImportRules()
IDS	SendAlert()	Cooperação na detecção de ataques entre quaisquer IDSs	Cooperação na detecção de ataques entre quaisquer IDSs	-	-	Interoperabilidade entre firewalls e IDSs	-
	SendRule() GetRule()	Cooperação na detecção de ataques entre quaisquer IDSs	Cooperação entre quaisquer IDSs	-	Interoperabilidade entre firewalls e IDSs	-	-
	ExportRules() ImportRules()	-	-	Interoperabilidade de regras entre IDSs de diferentes fabricantes	-	-	-
Firewall	SendLog()	-	Interoperabilidade entre firewalls e IDSs	-	Cooperação na auditoria de logs entre quaisquer firewalls	Cooperação na auditoria de logs entre quaisquer firewalls	-
	SendRule() GetRule()	Interoperabilidade entre firewalls e IDSs	-	-	Cooperação na auditoria de logs entre quaisquer firewalls	Cooperação entre quaisquer firewalls	-
	ExportRules() ImportRules()	-	-	-	-	-	Interoperabilidade de regras entre firewalls de diferentes fabricantes

Tabela 5.1: Interoperabilidade das operações da API

A única ressalva a ser feita na implementação do protótipo foi a não utilização de SSL a fim de criptografar a comunicação entre os componentes CORBA. O SSL foi utilizado somente a nível de comunicação entre os clientes de gerência e o servidor SIGSEC.

Este detalhe é muito importante pois, usuários com acesso físico na rede de gerência interna poderiam monitorar os pacotes na rede com ferramentas de *sniffing* e obter detalhes de configuração e *logs* dos equipamentos, e ainda, poderiam até mesmo reconfigurar seus equipamentos com um pouco mais de perspicácia.

O uso do SSL na camada do *middleware* CORBA é portanto essencial em uma aplicação profissional. O SSL não foi utilizado no protótipo (a nível de CORBA) porquê a implementação utilizada (Mico) ainda possui algumas falhas no suporte ao SSL, além disso, seria necessário um esforço de programação muito maior, o que

também não justificaria, já que o objetivo do trabalho está na padronização das interfaces.

O propósito do protótipo, nesta tese, não é ser uma aplicação profissional, e sim, comprovar a utilidade do modelo e padronização propostos e o uso de CORBA como ferramenta de desenvolvimento de aplicações de gerência.

Capítulo 6

Conclusão

Este capítulo conclui finalmente o trabalho, destacando as principais vantagens obtidas, os pontos de maior importância e avaliações do trabalho como um todo. Posteriormente são apresentadas as contribuições do projeto e sugestões para trabalhos futuros. Finalmente são apresentados esforços para disseminar, aperfeiçoar e perpetuar o trabalho.

6.1 Conclusões

- O objetivo primordial do trabalho foi alcançado, que era propor um modelo de gerência de segurança empregando o padrão CORBA e permitindo que ferramentas de segurança pudessem se comunicar de forma padronizada, flexível e escalável;

- Dentro do conceito de gerenciamento de ferramentas de segurança, a escolha de *firewalls* e IDSs foi bastante relevante. A padronização das interfaces para estas ferramentas, utilizando a IDL do padrão CORBA, se mostrou bastante clara e funcional;

- O uso de CORBA na padronização das interfaces, e consequentemente no desenvolvimento dos *drivers* para *firewalls* e IDSs, proporcionou realmente um desenvolvimento mais preciso, organizado e acima de tudo mais portável e fácil. O uso de CORBA abstraiu a programação a nível de rede e a preocupação com detalhes intrínsecos de plataforma e sistema operacional;

- A utilização do *Naming Service* do CORBA permitiu ganhos adicionais, tornando o sistema bastante flexível e escalável;

- A implementação do protótipo veio comprovar todas as expectativas, transformando um sistema de *firewall* e um IDS sem nenhuma facilidade de gerência remota em sistemas com interfaces padronizadas, gerenciáveis e com possibilidade de interoperar. O protótipo SIGSEC foi totalmente baseado no modelo proposto e por isso goza de todos os benefícios descritos no modelo, podendo gerenciar vários *firewalls* e IDSs simultaneamente através de uma simples interface Web;

- A implementação do protótipo utiliza várias técnicas (Ex.: Lex, Yacc, SQL) que podem ser seguidas no desenvolvimento de outros *drivers* ou aplicações baseadas no modelo. A utilização de tais técnicas proporcionou sensíveis ganhos de desempenho, clareza de programação e confiabilidade durante o desenvolvimento;

- O protótipo pode ser realmente utilizado como uma ferramenta profissional, bastando que sejam incluídas algumas características não implementadas tais como: SSL no nível de CORBA e gerenciamento completo de administradores. Estas características não foram implementadas pelo fato de não serem o objetivo do protótipo;

- Uma preocupação constante durante todo o trabalho era manter uma documentação o mais clara, confiável e padronizada possível. A utilização, ainda que modesta, da UML e a descrição detalhada de toda a API sugerida, forneceu resultados bastante aceitáveis, permitindo uma clara compreensão das funcionalidades fornecidas e uma fase de implementação bem organizada e menos sujeita a erros;

- Comparativamente, em relação a outros sistemas de gerenciamento integrados (ISMS), o SIGSEC fornece uma alternativa viável, baseada em um padrão consagrado (CORBA) e permitindo modificações e adições de funcionalidades ao sistema, sem alteração do modelo original. Na Tabela 6.1 podemos verificar comparativamente algumas das principais características destes sistemas.

As seguintes características merecem uma definição quanto ao seu emprego na Tabela 6.1:

Extensível : Facilidade de suporte ao gerenciamento de outras ferramentas de segurança;

	SIGSEC	OPSEC	Active Security	CDSA
Fabricante / Desenvolvedor	André S. Barbosa	Checkpoint	Network Associates Inc.	Open Group
Gratuito	SIM	SIM	NÃO	SIM
Padrão aberto	SIM	SIM	NÃO	SIM
Objetivo	Gerência específica de Firewalls e IDSs	VPNs, Firewalls, IDSs, Anti-vírus, Gerência de Rede	Gerência de Scanners, Firewalls, Anti-vírus, IDSs	Criptografia, Autenticação, Certificados
Extensível	Facilmente	SIM	NÃO	Facilmente
Integração	Qualquer fabricante	Entre produtos Checkpoint com outros fabricantes	Somente entre produtos da NAI	Qualquer fabricante
Complexidade	Baixa	Baixa	Baixa	Alta
Consolidação	Nenhuma	Alta	Baixa	Baixa
Escalabilidade	Alta	Alta	Alta	Média
Portabilidade	Alta	Média	Baixa	Alta

Tabela 6.1: Comparação entre Sistemas Integrados de Gerência de Segurança

Integração : Capacidade do sistema de se integrar a tecnologias de outros fabricantes;

Complexidade : Quantidade de operações, estruturas e detalhes da API do sistema, que pode indicar o grau de dificuldade de utilização do sistema;

Consolidação : Utilização do sistema no mercado. Indica se o sistema já é usado por vários fabricantes, e se já se tornou consolidado como alternativa de ISMS;

Escalabilidade : Facilidade de adaptação do sistema para gerenciamento de um número maior de ferramentas de segurança de um mesmo tipo, ou inclusão de características adicionais;

Portabilidade : Grau de facilidade na implementação do sistema em sistemas operacionais e plataformas de hardware diferentes.

6.2 Contribuições

- A contribuição imediata do trabalho é servir como uma alternativa de gerenciamento de segurança, que pode ser implementada em qualquer ambiente com diversidade de ferramentas de segurança;
- A utilização do padrão CORBA na construção do modelo e no desenvolvimento do protótipo pode ser estudada e seus conceitos aplicados em outros sistemas. Atualmente, existe muita bibliografia a respeito de CORBA, mas uma quantidade ínfima aborda o lado prático do uso deste padrão. Este trabalho vem então ajudar a preencher esta lacuna. Esta, é a maior contribuição deste trabalho;
- A API proposta pode ser adotada por outros desenvolvedores de software gratuitos ou não, podendo ser alterada de acordo com a necessidade de cada um;
- O modelo também pode ser extendido e adaptado de acordo com as necessidades de gerenciamento, assim, ele pode ser utilizado todo ou em parte;
- Este trabalho visa também alertar aos problemas de segurança, em especial aos problemas de gerenciamento e falta de padronização que podem acontecer.

6.3 Sugestões Para Trabalhos Futuros

- Desenvolvimento de *drivers* para outros *firewalls* e IDSs;
- Inclusão no modelo de uma camada SSL sobre CORBA e sua implementação no protótipo. A camada SSL vai permitir a autenticação dos objetos e a confidencialidade da comunicação entre eles;
- Atualizando o sistema para que suporte outros ORBs e linguagens de programação;
- Extensão do modelo para gerenciar VPNs, padronizando sua interface e desenvolvendo *drivers*;
- Extensão do modelo para gerenciar Anti-vírus, padronizando sua interface e desenvolvendo *drivers*;

- Extensão do modelo para gerenciar qualquer outra ferramenta de segurança ou não.

6.4 Desenvolvimento Ativo do Projeto

O SIGSEC é um projeto *Open Source* que é coordenado pelo autor deste trabalho através do *site*: *http://www.sigsec.org*. O projeto publica as interfaces padronizadas das ferramentas de segurança, *drivers* para o gerenciamento das ferramentas e softwares adicionais.

O projeto está em constante desenvolvimento e o objetivo é disseminar seu uso de forma gratuita contando com a colaboração de desenvolvedores para:

- Desenvolver novos drivers para IDSs e *firewalls* disponíveis;
- Propor novas padronizações para outros mecanismos de segurança ou alterações nos já existentes;
- Suportar outros ORBs;
- Desenvolver interfaces/módulos de gerência e configuração;
- Propor novas idéias baseadas no modelo proposto de gerenciamento utilizando CORBA.

Referências Bibliográficas

[1] HENNING M., VINOSKI, S., *Advanced CORBA Programming with C++*, 1 ed., Addison-Wesley, 1999.

[2] STANIFORD-CHEN, S., PORRAS, P. A., KAHN, C, TUNG, B., "A Common Intrusion Detection Framework", *http://seclab.cs.ucdavis.edu/cidf*, Jul, 1998.

[3] CHUNG, T. M., "Integrated Security Management", *Real-Time Systems Laboratory*, Dec, 2000.

[4] OPEN GROUP, "Common Security: CDSA and CSSM, Version 2 (with corrigenda)", *The Open Group*, May, 2000.

[5] OMG, "Common Object Request Broker Architecture (CORBA/IIOP) Specification", *Object Management Group*, Dec, 2001.

[6] OMG, "The C++ Language Mapping Specification", *Object Management Group*, Jun, 1999.

[7] OMG, "Event Service Specification", *Object Management Group*, Mar, 2001.

[8] OMG, "Naming Service Specification", *Object Management Group*, Feb, 2001.

[9] OMG, "Security Service Specification", *Object Management Group*, Mar, 2002.

[10] OMG, "Unified Modeling Language Specification", *Object Management Group*, Sep, 2001.

[11] OMG, *Object Management Group*, `http://www.omg.org`

[12] *SecurityFocus Corporate Site*, `http://www.securityfocus.com`

[13] CSI, *Computer Security Institute*,
http://www.gocsi.com

[14] MICO, *Implementação gratuita do padrão CORBA*,
http://www.mico.org

[15] SIGSEC, *Sistema Integrado de Gerência de Segurança Empregando CORBA*,
http://www.sigsec.org

[16] MYSQL, *Sistema de Gerenciamento de Banco de Dados*,
http://www.mysql.com

[17] UML, *Unified Modeling language*,
http://www.uml.org

[18] CORBA, *Common Object Request Broker Architecture*,
http://www.corba.org

[19] OPEN CORBA BENCHMARKING, *Open CORBA Benchmarking Site*,
http://nenya.ms.mff.cuni.cz/~bench/

[20] OPENBSD, *Sistema Operacional BSD gratuito*,
http://www.openbsd.org

[21] SNORT, *Sistema de Detecção de Intrusão Baseado em Rede e Gratuito*,
http://www.snort.org

[22] ARACHNIDS, *Arachnids Snort Rules*,
http://www.whitehats.com/ids

[23] ACID, *Analysis Console for Intrusion Databases*,
//http://www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html

[24] DEMARC, *Sistema completo de gerência de sensores Snort*,
http://www.demarc.org

[25] STALLINGS, W., *Cryptography and Network Security: Principles and Practice*, 2 ed, Prentice Hall, 1998

[26] OPENSOURCE, *Organização não comercial dedicada a promover o Open Source*,
http://www.opensource.org

[27] CONOBOY, B., FICHTNER, E., "IP Filter Based Firewalls HOWTO", *IPFilter Project*, Mar, 2000.

[28] IPFILTER, *Firewall Gratuito, incluindo filtro de pacotes e NAT*, `http://www.ipfilter.org`

[29] CHECKPOINT, *Check Point Software Technologies, Ltd.*, `http://www.checkpoint.com`

[30] OPSEC, *Open Plataform for Secure Enterprise Connectivity*, `http://www.opsec.com`

[31] AKER, *Aker Security Solutions*, `http://www.aker.com.br`

[32] NETFILTER, *Firewall gratuito produzido para o Linux*, `http://www.netfilter.org`

[33] RFCs, *Request for Comments*, `http://www.rfc-editor.org`

[34] SHIREY, R., "Internet Security Glossary", RFC2828, May, 2000.

[35] POSTEL, J. "User Datagram Protocol", RFC768, Aug, 1980.

[36] POSTEL, J. "Internet Protocol", RFC791, Sep, 1981.

[37] POSTEL, J. "Internet Control Message Protocol", RFC792, Sep, 1981.

[38] POSTEL, J. "Transmission Control Protocol", RFC793, Sep, 1981.

[39] TCPDUMP, *Tcpdump/Libpcap Software*, `http://www.tcpdump.org`

[40] NFR, *Network Flight Recorder*, `http://www.nfr.net`

[41] ISS, *Internet Security Systems*, `http://www.iss.net`

[42] NAI, *Network Associates Inc.*, `http://www.nai.com`

[43] CERT, *Computer Emergency Response Team*, `http://www.cert.org`

[44] ICSA, *Internet Consortium Security Agency*, `http://www.icsa.net`

[45] MIZRAHI, V. V., *Treinamento em Linguagem C++ – Módulo 1*, 1 ed., Makron Books, 1995.

[46] MIZRAHI, V. V., *Treinamento em Linguagem C++ – Módulo 2*, 1 ed., Makron Books, 1995.

[47] PUDER, A., RÖMER, K., *Mico: An Open Source CORBA Implementation*, 3 ed., Academic Press and dpunkt-Verlag, 2000.

[48] OTTE, R., PATRICK, P., ROY, M., *Understanding CORBA: The Common Object Request Broker Architecture*, 1 ed., Prentice Hall, 1996.

[49] MOWBRAY, T. J., ZAHAVI, R., *The Essential CORBA: Systems Integration Using Distributed Objects*, 1 ed., John Wiley & Sons, Inc., 1995.

[50] PAGE-JONES, M., *O que Todo Programador Deveria Saber Sobre Projeto Orientado a Objeto*, 1 ed., Makron Books, 1997.

[51] FOWLER, M., SCOTT, K., *UML Essencial*, 2 ed., Bookman, 2002.

[52] LIBERTY J., *Aprenda em 24 horas C++*, 2 ed., Campus, 1997.

[53] BERNSTEIN, T., BHIMANI, A. B., SCHULTZ, E., SIEGEL, C. A., *Segurança na Internet*, 1 ed., Campus, 1997.

[54] LEE, W., STOLFO, S. J., MOK, K. W., "A Data Mining Framework for Building Intrusion Detection Models", *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, USA, pp. 120-132, May. 1999.

[55] NICOMETE, V., DESWARTE, Y., "An Authorization Scheme For Distributed Object Systems", *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, USA, pp. 21-30, May. 1997.

[56] SCHNEIER, B., *Applied Cryptography*, 2 ed., John Wiley & Sons, Inc., 1996.

[57] SCHNEIER, B., *Segurança.com: Segredos e Mentiras Sobre a Proteção na Vida Digital*, 1 ed., Campus, 2001.

[58] SATIR, G., BROWN, D., *Técnicas de Programação em C++*, 1 ed., O'Reilly & Associates, Inc., 1997.

[59] CHECK POINT, "Introduction to Firewall-1 Management (CCSA)", 1999.

[60] CHECK POINT, "Introduction to Advanced Firewall-1 Management (CCSE)", 1999.

[61] CHECK POINT, "Open Platform for Security (OPSEC^{TM})", 2000.

[62] GALVÃO, M., "IDS Conceitos", *Módulo Security*, 2000.

[63] ROSENBERGER, J., *Teach Yourself CORBA In 14 Days*, 1 ed., Macmillan Computer Publishing, 1999.

[64] CSI/FBI, "Computer Crime and Security Survey", 2002.

[65] RANUM, M. J., CURTIN, M., *Internet Firewalls: Frequently Asked Questions*, 2000.

[66] TUMA, P., BUBLE, A., *Open CORBA Benchmarking*, 2001.

Apêndice A

Glossário

Active Security : Sistema integrado de gerenciamento de segurança (ISMS) desenvolvido pela NAI (*Network Associates*).

ACL : Lista de controle de acesso, do inglês: *Access Control List*. Denominação utilizada principalmente em roteadores para indicar regras de filtro de pacotes.

Aker : Empresa brasileira fabricante do *firewall* de mesmo nome.

Anti-Vírus : Software especializado em detectar e remover vírus de computador.

Apache : Servidor Web mais utilizado no mundo. O apache é um projeto *Open Source* e pode ser encontrado em *http://www.apache.org*.

API : *Application Program Interface*, escrita também como *Application Programming Interface*, significa interface de programação de aplicação. É um método especificado por um sistema operacional ou aplicação pelo qual um programador pode programar operações de outro sistema operacional ou aplicação.

Backdoor : Conhecida como "porta dos fundos", é um software ou alteração de software feita por um *hacker* que permite a este acesso posterior, de forma facilitada, ao sistema invadido.

Backup : Procedimento de cópia de arquivos ou de todo o sistema que permite a sua posterior recuperação em caso de falhas ou perda de dados.

Bastião : Máquina que pode permitir o accesso de clientes externos à rede interna e vice e versa. Esta máquina é componente de uma arquitetura de *firewall*, sendo um pondo bastante crítico.

Boolean : Váriavel que pode assumir apenas dois valores: verdadeiro ou falso.

BOA : *Basic Object Adapter*. É o adaptador básico de objetos do padrão CORBA. Atualmente foi substituído pelo POA (*Portable Object Adapter*).

Buffer Overflow : Estado que ocorre quando um *buffer* de memória (geralmente utilizado por uma *string*), é sobrescrito com uma quantidade grande de dados, além do que o *buffer* comporta. Este estado permite desviar a execução do programa e executar código arbitrário, sendo portanto uma falha grave de segurança.

CGI : *Common Gateway Interface*. É uma forma padronizada para um servidor Web passar uma requisição de um usuário (geralmente através de um formulário) para uma aplicação e após o processamento dos dados responder de volta ao usuário.

Check Point : Empresa fabricante de produtos de segurança, em particular, do famoso Firewall-1.

CORBA : *Common Object Request Broker Architecture*. É uma arquitetura e especificação para criar, distribuir e gerenciar objetos disribuídos em uma rede. O padrão CORBA permite que programas em diferentes localizações e de diferentes fabricantes possam se comunicar. O padrão foi desenvolvido por um consórcio chamado OMG (*Object Management Group*), que é formado hoje por mais de 500 empresas.

Cracker : *Hacker* que utiliza seus conhecimentos para obter lucro e destruir informações e sistemas.

Criptoanálise : Técnicas de criptografia utilizadas para recuperar um texto cifrado sem saber a chave ou algoritmo com qual o texto foi cifrado. Estas técnicas também podem ser utilizadas para se descobrir fraquezas em algoritmos e analisar sua segurança e eficiência.

Criptografia : A criptografia comporta os estudos da criptologia e da criptoanálise. Atualmente a palavra criptografia é aplicada com o mesmo sentido da criptologia que estuda como cifrar e decifrar mensagens usando diferentes técnicas matemáticas e protocolos.

CSI : *Computer Security Institute*. É uma organização dedicada a servir e treinar profissionais da computação e de segurança de redes. Associada ao FBI, a CSI publica pesquisas e realiza conferências sobre segurança.

Daemon : Denominação dada a programas em ambientes UNIX que executam tarefas de forma permanente, sem interação com o usuário e portanto não necessitam estar associados ao terminal (dispositivos de teclado e vídeo).

DII : *Dynamic Invocation Interface*. É a interface de invocação dinâmica do padrão CORBA, na qual é possível determinar interfaces de objetos e invocar operações em tempo de execução.

DNS : *Domain Name System*. É o sistema de nomes de domínio da Internet, responsável por traduzir nomes de domínio em endereços IP e vice e versa.

DoS : *Denial of Service*. Conhecido como ataque de negação de serviço, é uma técnica utilizada por *crackers* para paralisar sistemas causando perdas financeiras e de dados.

Driver : Peça de software destinada a compatibilizar um determinado hardware a um software, ou mesmo entre dois softwares.

DSI : *Dynamic Specification Interface*.

Dual-homed bastion : Bastião interligado com duas interfaces de rede.

Ethernet : É a tecnologia de redes locais mais utilizada, especificada no padrão IEEE 802.3. A Ethernet foi originalmente desenvolvida pela Xerox e aperfeiçoada pela Intel, DEC e a própria Xerox.

FDDI : *Fiber Distributed Data Interface*. É um conjunto de padrões ANSI e ISO para transmissão em fibras óticas em uma rede local que pode ser extendido até 200Km. O protocolo FDDI é baseado no *Token Ring*.

Filtro de Conteúdo : Software que pode permitir ou bloquear tráfego baseado no conteúdo que está sendo acessado, geralmente utilizado para controlar o tráfego Web de usuários.

Filtro de Pacotes : Software utilizado para controlar tráfego baseado em regras préviamente determinadas. Estas regras são criadas baseando-se nos campos

dos cabeçalhos dos protocolos Internet, podendo bloquear ou permitir os pacotes na interface onde as regras se aplicam.

Firewall : Software ou conjunto de hardware e software que impõe políticas de controle de acesso entre uma rede e outra. Os *firewalls* possuem hoje várias funcionalidades, tais como: filtros pacotes, NAT, *proxies*, etc.

Firewall-1 : Famoso *firewall* comercializado pela Check Point.

FTP : *File Transfer Protocol*. Protocolo de transferência de arquivo da Internet. Utiliza as portas 20 e 21.

Gateway : Máquina responsável por concentrar o tráfego, roteando pacotes para determinados *links*. Esta máquina pode ser um roteador ou um computador com o sistema operacional propriamente configurado.

Hacker : Indivíduo com conhecimentos de computação acima da média, geralmente em programação e segurança de redes e sistemas. Os verdadeiros *hackers* não estão interessados em obter lucros ou destruir dados, mas sim em vencer desafios e aprender cada vez mais.

Hash : Cadeia de bits gerada por software específico que corresponde a um determinado arquivo. Este *hash* é gerado por algoritmos matemáticos fortes e pode ou não ter tamanho fixo. Não é possível manipular um arquivo para corresponder a um *hash* desejado.

Host : Computador conectado em rede.

HTTP : *Hiper-Text Transfer Protocol*. Protocolo de transferência de hipertexto. Utiliza a porta 80.

HTTPS : HTTP Seguro, utiliza geralmente SSL e porta 443.

ICMP : *Internet Control Message Protocol* [37].

IDL : *Interface Definition Language*. Linguagem de definição de interfaces do padrão CORBA.

IDS : *Intrusion Detection System*. Sistema de Detecção de Intrusão.

IP : *Internet Protocol* [36].

IPF : *Internet Packet Filter*. Projeto de código aberto de um *firewall* bastante utilizado. O mesmo que *IP Filter*.

ISMS : *Integrated Security Management System*. Sistema de gerenciamento de segurança integrado.

ISP : *Internet Service Provider*. Provedor de acesso Internet.

Kernel : Núcleo de um sistema operacional.

Lex : Ferramenta de software para análise léxica muito utilizada em compiladores.

Libpcap : Biblioteca para captura de pacotes para o ambiente UNIX.

Mico : Implementação gratuita do padrão CORBA.

MySQL : Sistema de gerenciamento de banco de dados gratuito.

Naming Service : Servidor de nomes do padrão CORBA.

NAT : *Network Address Translation*. Tradução de endereços de rede.

NIDS : *Network Intrusion Detection System*, sistema de detecção de intrusão baseado em rede.

OA : *Object Adapter*, adaptador de objetos do padrão CORBA.

OMA : *Object Management Architecture*, arquitetura de gerenciamento de objetos padronizada pela OMG.

OMG : *Object Management Group*. Organização formada por diversas empresas com o objetivo de criar uma padronização para sistemas distribuídos e o gerenciamento de objetos.

OpenBSD : Sistema operacional baseado no 4.4BSD, gratuito e bastante seguro na sua instalação padrão.

OpenBSD PF : *Firewall* nativo do OpenBSD.

ORB : Em um ambiente baseado no padrão CORBA, um *Object Request Broker* (ORB) é um programa ou biblioteca que age como um intermediário (*broker*)entre a requisição de um cliente por um serviço de um componente ou

objeto distribuído. Ter um ORB em um ambiente distribuído significa que o programa cliente pode requerer um serviço sem precisar saber onde o servidor está ou saber exatamente como é a sua interface.

Payload : Camada de aplicação de um protocolo.

POA : *Portable Object Adapter*, é o adaptador portável de objetos do padrão CORBA.

Port : Esta palavra possui dois significados: Pode indicar a porta de conexão (exemplo: HTTP é porta 80) ou pode significar um software que foi alterado para funcionar corretamente em um determinado sistema operacional (do verbo portar). O segundo significado é empregado nesta tese.

SIGSEC : Sistema Integrado de Gerência de Segurança Empregando CORBA.

Snort : Sistema de detecção de intrusão baseado em rede, gratuito e bastante utilizado.

SQL : *Structured Query Language*.

SSH : *Secure Shell*, alternativa ao telnet, provendo acesso a máquinas UNIX de forma segura.

SSL : *Secure Sockets Layer*, camada de segurança para aplicações em redes IP, sendo aplicada em HTTP, SMTP, CORBA e outros.

TCP : *Transmission Control Protocol* [38].

Token Ring : Tecnologia de rede local onde os computadores são conectados em configuração anel ou estrela e um esquema de *token* é utilizado para previnir colisão de dados entre dois computadores. Apenas a máquina que possui o *token* no momento pode enviar pacotes. Esta é a tecnologia mais utilizada para redes locais depois da Ethernet.

UDP : *User Datagram Protocol* [35].

UML : *Unified Modeling Language*, linguagem unificada de modelagem, padronizada pela OMG.

VPN : *Virtual Provate Networks*. Redes privadas virtuais, podendo ser construídas utilizando o protocolo IPSec e criando túneis criptografados, ou mesmo sem o uso da criptografia, provendo somente um ambiente confiável e independente do tráfego da Internet.

Yacc : Ferramenta de software para análise sintática muito utilizada em compiladores.

Apêndice B

Cabeçalhos dos Protocolos Básicos da Internet

B.1 IP (*Internet Protocol*)

Na Figura B.1 mostramos como é formado o cabeçalho IP, em seguida os campos são detalhados. A especificação completa pode ser obtida na RFC791.

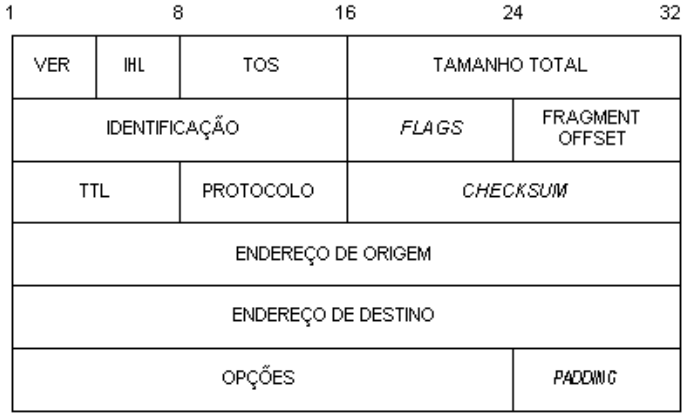


Figura B.1: IP (*Internet Protocol*)

VER (*Version*) : 4 bits. Indica a versão do protocolo IP utilizado, atualmente existem a versão 4 e a 6. A versão mais utilizada atualmente é a versão 4;

IHL (*Internet Header Length*) : 4 bits. Indica o tamanho do cabeçalho IP em palavras de 32 bits, o valor mínimo é 5;

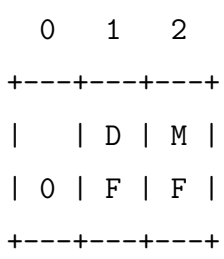
TOS (*Type of Service*) : 8 bits. Indica o tipo de qualidade de serviço desejado. Atualmente este campo é pouco utilizado;

Tamanho Total (*Total Length*) : 16 bits. Indica o tamanho total do pacote IP em bytes (datagrama IP). O tamanho máximo é de 65535 bytes;

Identificação (*Identification*) : 16 bits. Campo utilizado para identificar um pacote IP. Este campo ajuda na fragmentação e na montagem dos pacotes IP;

Flags : 3 bits. São bits utilizados para controle.

```
Bit 0: reservado, deve ser zero.
Bit 1: (DF) 0 = Pode fragmentar,  1 = Não pode fragmentar.
Bit 2: (MF) 0 = Último fragmento, 1 = Mais fragmentos.
```



Fragment Offset : 13 bits. Este campo indica qual a parte do pacote original que este fragmento pertence. O valor é medido em unidades de 8 bytes, o primeiro fragmento possui *offset* igual a zero;

TTL (*Time to Live*) : 8 bits. Campo utilizado para evitar que um pacote IP fique circulando eternamente na Internet por um problema de roteamento. Este campo é decrementado de 1 a cada *hop*. Quando este chega a 0 o pacote é descartado;

Protocolo (*Protocol*) : 8 bits. Campo utilizado para indicar qual é o protocolo utilizado no próximo nível. A completa lista pode ser obtida na RFC1700;

Checksum : 16 bits. Campo que indica o *checksum* do cabeçalho. Toda vez que o cabeçalho é alterado este campo é recalculado, o que ocorre pelo menos a cada *hop*;

Endereço de Origem (*Source Address*) : 32 bits. Campo designado para conter o endereço IP de origem do pacote;

Endereço de Destino (*Destination Address*) : 32 bits. Campo designado para conter o endereço IP de destino do pacote;

Opções (*Options*) : Tamanho variável. Este campo é utilizado para opções adicionais, que podem ou não estar presentes. Maiores detalhes estão descritos na RFC791;

Padding : Tamanho variável. Este campo é utilizado para garantir que o cabeçalho do pacote IP sempre tenha um tamanho multiplo de 32 bits;

B.2 TCP (*Transmission Control Protocol*)

Na Figura B.2 mostramos como é formado o cabeçalho TCP, em seguida os campos são detalhados. A especificação completa pode ser obtida na RFC793.

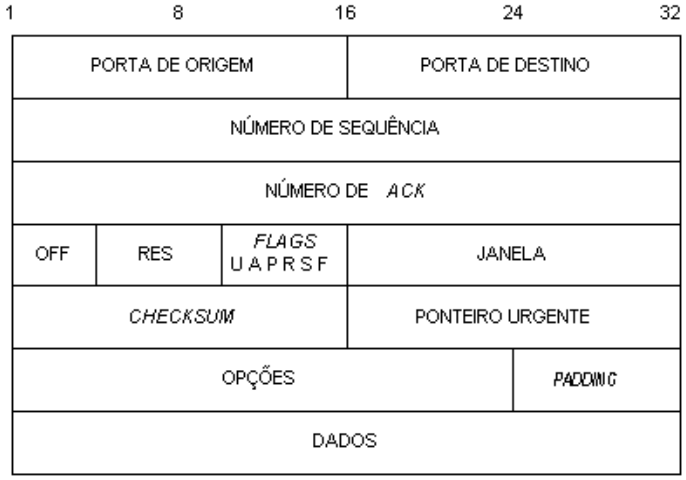


Figura B.2: TCP (*Transmission Control Protocol*)

Porta de Origem (*Source Port*) : 16 bits. Campo que transporta o número da porta origem da conexão;

Porta de Destino (*Destination Port*) : 16 bits. Campo que transporta o número da porta destino da conexão;

Número de Sequência (*Sequence Number*) : 32 bits. Número de sequência do envio de bytes de dados;

Número de *Ack* (*Acknowledgment Number*) : 32 bits. Se o bit ACK estiver ativo este campo indica o próximo número de sequência que o emissor da mensagem está esperando receber;

OFF (*Data Offset*) : 4 bits. Campo que indica o tamanho do cabeçalho do TCP em palavras de 32 bits;

RES (*Reserved*) : 6 bits. Reservado para uso futuro, deve ser igual a zero;

Flags : 6 bits. Da esquerda para direita:

- URG: *Urgent Pointer*;
- ACK: *Acknowledgment*;
- PSH: *Push Function*;
- RST: *Reset Connection*;
- SYN: *Synchronize*;
- FIN: *Finalize*;

Janela (*Window*) : 16 bits. Este campo indica a quantidade de bytes que podem ser recebidos pelo emissor sem que seja necessário enviar uma confirmação de recepção (ACK);

Checksum : 16 bits. Campo que indica o *checksum* do cabeçalho;

Ponteiro Urgente (*Urgent Pointer*) : 16 bits. Campo utilizado em conjunto com o bit URG para apontar o número de sequência do segmento que contém os dados marcados como "urgentes";

Opções (*Options*) : Tamanho variável. Campo reservado para opções adicionais;

Padding : Tamanho variável. Campo utilizado para garantir que o cabeçalho do TCP tenha um tamanho múltiplo de 32 bits;

B.3 UDP (*User Datagram Protocol*)

Na Figura B.3 mostramos como é formado o cabeçalho UDP, em seguida os campos são detalhados. A especificação completa pode ser obtida na RFC768.

Porta de Origem (*Source Port*) : 16 bits. Campo que transporta o número da porta origem;

Porta de Destino (*Destination Port*) : 16 bits. Campo que transporta o número da porta destino do serviço UDP;

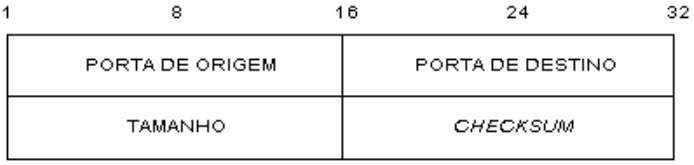


Figura B.3: UDP (*User Datagram Protocol*)

Tamanho (*Length*) : 16 bits. Indica o tamanho em bytes do cabeçalho mais os dados do pacote UDP;

Checksum : 16 bits. Campo que indica o *checksum* do cabeçalho UDP;

B.4 ICMP (*Internet Control Message Protocol*)

Na Figura B.4 mostramos como é formado o cabeçalho ICMP, em seguida os campos são detalhados. A especificação completa pode ser obtida na RFC792.

Figura B.4: ICMP (*Internet Control Message Protocol*)

Tipo (*Type*) : 8 bits. Identifica o tipo da mensagem ICMP;

Código (*Code*) : 8 bits. Identifica o tipo da mensagem ICMP;

Checksum : 16 bits. Campo que indica o *checksum* do cabeçalho ICMP;

Os pacotes ICMP são divididos em mensagens, cada mensagem possui algumas alterações no formato do cabeçalho. A mensagem é determinada por um campo chamado **tipo**, no início do cabeçalho, e pode ser:

- Tipo 0 - *Echo Reply Message*;
- Tipo 3 - *Destination Unreachable Message*;
- Tipo 4 - *Source Quench Message*;

- Tipo 5 - *Redirect Message*;
- Tipo 8 - *Echo Request Message*;
- Tipo 11 - *Time Exceeded Message*;
- Tipo 12 - *Parameter Problem Message*;
- Tipo 13 - *Timestamp Request Message*;
- Tipo 14 - *Timestamp Reply Message*;
- Tipo 15 - *Information Request*;
- Tipo 16 - *Information Reply Message*;

Cada mensagem pode possuir vários códigos, indicando situações diferentes ou tipos de resposta ou pergunta da mensagem. Detalhes específicos podem ser obtidos na RFC792.

Apêndice C

Código Fonte Parcial do Protótipo SIGSEC

O código fonte completo pode ser obtido em *http://www.sigsec.org*. O código completo não foi incluído aqui na sua totalidade por ser bastante extenso.

C.1 IDLs Padronizadas para *Firewalls* e IDSs

C.1.1 `sigsec_fw.idl`: IDL para *firewalls*

```
/* Firewall SIGSEC IDL 0.6 */

/*
* Copyright (c) 2002, André S. Barbosa <andre@infolink.com.br>
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* - Redistributions of source code must retain the above copyright
*   notice, this list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright
*   notice, this list of conditions and the following disclaimer in the
*   documentation and/or other materials provided with the distribution.
*
* - Neither the name of the Federal University of Rio de Janeiro nor the
*   names of its contributors may be used to endorse or promote products
*   derived from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
```

```
* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USES
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

module Firewall {

    typedef unsigned long  IP;
    typedef unsigned short Port;
    typedef unsigned long  u_long ;
    typedef unsigned short u_short;

    enum Tproto  { PROTO_ALL,
   PROTO_IP,
   PROTO_TCP,
   PROTO_UDP,
   PROTO_TCPUDP,
   PROTO_ICMP };

    enum Taction { ACT_PASS_IN,
   ACT_PASS_OUT,
   ACT_KEEP_IN,
   ACT_KEEP_OUT,
   ACT_DENY_IN,
   ACT_DENY_OUT,
   ACT_RJCT_IN,
   ACT_RJCT_OUT };

    enum Tportop { PORT_EQ,
   PORT_NE,
                   PORT_GE,
   PORT_GT,
   PORT_LE,
   PORT_LT,
   PORT_IRG,
   PORT_XRG };

    enum Treject { RJCT_NOT,
   RJCT_RST,
   RJCT_ICMP,
   RJCT_ICMPCODE };

    enum Tnatact { NAT_UNIDIR,
          NAT_BIDIR,
      NAT_RDR };

    struct Tflags {
        boolean syn, ack, rst, fin, urg, psh;
    };

    struct CHost {
IP      addr;
octet   mask;
        boolean no_ip;
Port    port_a;
Port    port_b;
```

```
Tportop port_op;
    };

    interface PacketFilter {

     struct Rule {
    boolean state;
    string  owner;
    string  date;
    string  comment;
    CHost   from;
    CHost   to;
    Taction action;
    Tproto  proto;
    Treject reject;
    IP     iface;
    boolean quick;
    Tflags  flags_set;
    Tflags  flags_use;
    boolean flags_or;
    octet   ret_icmp_code;
    octet   icmp_type;
    octet   icmp_code;
    boolean used_icmp_type;
    boolean used_icmp_code;
    octet   log;
     };

        typedef sequence<Rule> RulesList;

        struct Stats {
    u_long  pkts_in_pass;
    u_long  pkts_in_drop;
    u_long  pkts_out_pass;
    u_long  pkts_out_drop;
    u_long  bytes_in;
    u_long  bytes_out;
    string  raw_pf_stats;
    string  raw_nat_stats;
};

/* Estado do Firewall  0 -> desligado, 1 -> ligado */
attribute boolean State;
/* Reinicia o Firewall */
void Restart();
/* Modifica a regra 'n', enviando rule */
void SetRule(in u_short n, in  Rule rule);
/* Le a regra 'n' */
void GetRule(in u_short n, out Rule rule);
/* Insere uma regra(rule) na posicao 'n', todas as regras >= n sao deslocadas + 1 */
void InsertRule(in u_short n, in Rule rule);
/* Apaga a regra na posicao 'n', todas as regras > n sao deslocadas - 1 */
void DeleteRule(in u_short n);
/* Envia todas as regras em ruleslist para o Firewall */
void ExportRules(in  RulesList ruleslist);
/* Obtem toda a lista de regras do Firewall em ruleslist */
```

```
void ImportRules(out RulesList ruleslist);
/* Envia o hash das listas de regras no cliente e compara com o hash das regras */
/* do Firewall, devolve 1 se o hash coincide e 0 caso contrario */
void HashRules(in u_long hash, out boolean res);
/* Obtem todas as estatisticas do Firewall em stats */
        void GetStats(out Stats stats);
    };


    interface Nat {

struct Rule {
    boolean state;
    string  owner;
    string  date;
    string  comment;
            Tnatact natact;
    IP     iface;
    boolean no_iface;
    Tproto  proto;
    CHost   from;
    CHost   to;
    CHost   external;
};

        typedef sequence<Rule> RulesList;

/* Idem as funcoes do Filtro de Pacotes, so' que para Nat */
void SetRule(in u_short n, in  Rule rule);
void GetRule(in u_short n, out Rule rule);
void InsertRule(in u_short n, in Rule rule);
void DeleteRule(in u_short n);
void ExportRules(in  RulesList ruleslist);
void ImportRules(out RulesList ruleslist);
void HashRules(in u_long hash, out boolean res);
    };


    interface FwLog {

struct Log {
    string  fwname;
    u_short ruleno;
    u_long  usec;
    octet   sec;
    octet   min;
    octet   hour;
    octet   day;
    octet   mon;
    u_short year;
    u_short length;
    IP      ip_src;
    IP      ip_dst;
    octet   ttl;
    Tproto  proto;
    Port    port_src;
```

```
    Port    port_dst;
    Tflags  tcp_flags;
    u_long  tcp_seq;
    u_long  tcp_ack;
    octet   icmp_type;
    octet   icmp_code;
    string  payload;
};

oneway void SendLog(in Log log);

    };

};
```

C.1.2 sigsec_ids.idl: IDL para IDSs

```
/* IDS SIGSEC IDL 0.6 /*

/*
* Copyright (c) 2002, André S. Barbosa <andre@infolink.com.br>
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* - Redistributions of source code must retain the above copyright
*   notice, this list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright
*   notice, this list of conditions and the following disclaimer in the
*   documentation and/or other materials provided with the distribution.
*
* - Neither the name of the Federal University of Rio de Janeiro nor the
*   names of its contributors may be used to endorse or promote products
*   derived from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USES
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

module IDS {

    typedef unsigned long  IP;
    typedef unsigned short Port;
    typedef unsigned long  u_long ;
```

```
    typedef unsigned short u_short;

    enum Taction { ACT_LOG,
   ACT_ALERT,
   ACT_LOGALERT};

    enum Tproto  { PROTO_ALL,
   PROTO_IP,
   PROTO_TCP,
   PROTO_UDP,
   PROTO_TCPUDP,
   PROTO_ICMP };

    enum Tportop { PORT_EQ,
   PORT_NE,
   PORT_GE,
   PORT_GT,
   PORT_LE,
   PORT_LT,
   PORT_IRG,
   PORT_XRG };

    enum Tplszop { PAYLOAD_EQ,
   PAYLOAD_LT,
   PAYLOAD_GT };

    enum Tipopts { IPOPTS_RR,
   IPOPTS_EOL,
   IPOPTS_NOP,
   IPOPTS_TS,
   IPOPTS_SEC,
   IPOPTS_LSRR,
   IPOPTS_SSRR,
   IPOPTS_SATID };

    struct Tfields {
boolean tos, payload_size, id, ttl, ip_proto, ipopts, depth, offset;
boolean tcp_seq, tcp_ack, icmp_type, icmp_code, icmp_id, icmp_seq;
    };

    struct Tfbits {
        boolean rb, mf, df;
    };

    struct Tflags {
        boolean syn, ack, rst, fin, urg, psh;
    };

    struct CHost {
        IP      addr;
        octet   mask;
        boolean no_ip;
        Port    port_a;
        Port    port_b;
        Tportop port_op;
    };
```

```
    interface NIDS {

struct Rule {
    boolean state;
    string  owner;
    string  date;
    string  comment;
    Taction action;
    boolean bidirect;
    Tproto  proto;
    CHost   from;
    CHost   to;
    string  message;
    string  content;
    boolean no_content;
    string  uricontent;
    boolean no_uricontent;
    string  content_list;
    boolean no_content_list;
    boolean nocase;
    Tfbits  fragbits_set;
    Tfbits  fragbits_use;
    boolean fragbits_or;
    Tflags  flags_set;
    Tflags  flags_use;
    boolean flags_or;
    u_long  rpc;
    octet   tos;
    u_short payload_size;
    Tplszop payload_op;
    u_short id;
    octet   ttl;
      octet   ip_proto;
    boolean no_ip_proto;
    Tipopts ipopts;
    u_short depth;
      u_short offset;
    u_long  tcp_seq;
    u_long  tcp_ack;
    octet   icmp_type;
    octet   icmp_code;
    u_short icmp_id;
    u_short icmp_seq;
    Tfields used_fields;
        };

typedef sequence<Rule> RulesList;

struct Group {
    string tag;
    string comment;
};

typedef sequence<Group> GroupList;
```

```
        struct Stats {

    u_long pkts_recv;
    u_long pkts_drop;
    u_long pkts_log;
    u_long pkts_alert;
    u_long pkts_tcp;
    u_long pkts_udp;
    u_long pkts_icmp;
    string raw_stats;

};

        /* Estado do IDS  0 -> desligado, 1 -> ligado */
        attribute boolean State;
        /* Reinicia o IDS */
        void Restart();
        /* Modifica a regra 'n', enviando rule */
        void SetRule(in u_short n, in u_short g, in  Rule rule);
        /* Le a regra 'n' */
        void GetRule(in u_short n, in u_short g, out Rule rule);
        /* Insere uma regra(rule) na posicao 'n', todas as regras >= n sao deslocadas + 1 */
        void InsertRule(in u_short n, in u_short g, in Rule rule);
        /* Apaga a regra na posicao 'n', todas as regras > n sao deslocadas - 1 */
        void DeleteRule(in u_short n, in u_short g);
        /* Envia todas as regras em ruleslist para o IDS */
        void ExportRules(in u_short g, in  RulesList ruleslist);
        /* Obtem toda a lista de regras do IDS em ruleslist */
        void ImportRules(in u_short g, out RulesList ruleslist);
        /* Obtem toda a lista de grupos do IDS em grouplist */
        void ImportGroups(out GroupList grouplist);
        /* Envia todos os grupos em grouplist para o IDS */
        void ExportGroups(in  GroupList grouplist);
        /* Envia o hash das listas de regras no cliente e compara com o hash das regras */
        /* do IDS, devolve 1 se o hash coincide e 0 caso contrario */
        void HashRules(in u_short g, in u_long hash, out boolean res);
        /* Obtem todas as estatisticas do IDS em stats */
        void GetStats(out Stats stats);

    };


    interface IDSAlert {

        struct Alert {
            string  idsname;
            string  message;
            u_short ruleno;
            u_long  usec;
            octet   sec;
            octet   min;
            octet   hour;
            octet   day;
            octet   mon;
            u_short year;
```

```
            u_short length;
            IP      ip_src;
            IP      ip_dst;
            octet   ttl;
            Tproto  proto;
            Port    port_src;
            Port    port_dst;
            Tflags  tcp_flags;
            u_long  tcp_seq;
            u_long  tcp_ack;
            octet   icmp_type;
            octet   icmp_code;
        };

        oneway void SendAlert(in Alert alert);

    };

};
```

C.1.3 `Makefile`: Para compilação das IDLs

```
OBJS= sigsec_fw.o sigsec_ids.o
CFLAGS= -I. -O2 -Wall -Werror

all: start ${OBJS} end

start:
@echo
@echo "Changing to IDLs/, compiling IDLs(Firewall and IDS) for OpenBSD..."
@echo

sigsec_fw.cc: sigsec_fw.idl
idl sigsec_fw.idl

sigsec_ids.cc: sigsec_ids.idl
idl sigsec_ids.idl

sigsec_fw.o: sigsec_fw.cc sigsec_fw.h
mico-c++ ${CFLAGS} -c sigsec_fw.cc -o sigsec_fw.o

sigsec_ids.o: sigsec_ids.cc sigsec_ids.h
mico-c++ ${CFLAGS} -c sigsec_ids.cc -o sigsec_ids.o

end:
@echo "SigSec OpenBSD IDLs(Firewall and IDS) compiled!"

install:
@echo "Nothing to install in IDLs/"

clean:
rm -f *~ *core *.o sigsec_fw.cc sigsec_fw.h sigsec_ids.cc sigsec_ids.h
```

C.2 *Driver* CORBA para o OpenBSD PF

C.2.1 sigsec_pfd.cc: Programa principal do *driver* do OpenBSD PF

```
/* SigSec PF CORBA Driver  */
/* by André S. B. */

#include "sigsec_pfd.h"


int main(int argc, char *argv[]) {

    //Obtendo o hostname
    char hostname[50];
    gethostname(hostname, sizeof(hostname));

    // Abrindo arquivo de configuracao para obter os dados do firewall
    string line;
    string pfname="";
    string pfport="";
    string naminghost="";
    string namingport="";
    ifstream conf("/etc/sigsec.conf");
    while(conf) {
         conf >> line;
         if (line.substr(0,8) == "PF_NAME=") {
                pfname = line.substr(8);
         }
         if (line.substr(0,8) == "PF_PORT=") {
                pfport = line.substr(8);
         }
         if (line.substr(0,8) == "PF_CONF=") {
                pfconf = line.substr(8);
         }
         if (line.substr(0,9) == "NAT_CONF=") {
                natconf = line.substr(9);
         }
         if (line.substr(0,10) == "FIX_IFACE=") {
                fixiface = line.substr(10);
         }
 if (line.substr(0,12) == "NAMING_HOST=") {
                naminghost = line.substr(12);
         }
         if (line.substr(0,12) == "NAMING_PORT=") {
                namingport = line.substr(12);
         }
    }

    conf.close();

    if (pfport == "") pfport = "9001";
    if (pfconf == "") pfconf = "/etc/pf.conf";
    if (natconf == "") natconf = "/etc/nat.conf";
    if (fixiface == "")
```

```
       fixiface = "NO";
    else
       cerr << "Interface fixa: " << fixiface.c_str() << endl;
    if (naminghost == "") naminghost = hostname;
    if (namingport == "") namingport = "9000";
    if (pfname == "") {
        cerr << "No PF_NAME found in /etc/sigsec.conf" << endl;
        exit(0);
    }

    init_log();

    // Inicializacao do CORBA
    argv[argc] = new char[100];
    snprintf(argv[argc], 100, "-ORBIIOPAddr=inet:%s:%s", hostname, pfport.c_str());
    argc++;
    CORBA::ORB_var orb = CORBA::ORB_init( argc, argv );
    CORBA::Object_var poaobj = orb->resolve_initial_references ("RootPOA");
    PortableServer::POA_var poa = PortableServer::POA::_narrow (poaobj);
    PortableServer::POAManager_var mgr = poa->the_POAManager();

    PortableServer::ObjectId_var oid;
    CORBA::Object_var ref;

    // Resolvendo referencia para o Naming
    char refnaming[100];
    snprintf(refnaming, sizeof(refnaming), "corbaloc::%s:%s/NameService",
             naminghost.c_str(), namingport.c_str());
    CORBA::Object_var nsobj;
    try {
        nsobj = orb->string_to_object(refnaming);
    } catch (CORBA::ORB::InvalidName in){
        cerr << "Naming Service invalido!";
    } catch (CORBA::BAD_PARAM bp) {
        cerr << "Naming: parametro invalido!";
    }
    CosNaming::NamingContext_var nc = CosNaming::NamingContext::_narrow(nsobj);


    // Registrando no Naming
    // Contexto Firewall
    CosNaming::Name name;
    name.length(1);
    name[0].id = CORBA::string_dup("Firewall");
    name[0].kind = CORBA::string_dup("");
    CosNaming::NamingContext_var ncfw;
    try {
ncfw = nc->bind_new_context(name);
    } catch (CosNaming::NamingContext::AlreadyBound ab) {
cerr << "Contexto Firewall já criado, resolvendo..." << endl;
        CORBA::Object_var fwobj = nc->resolve(name);
        ncfw = CosNaming::NamingContext::_narrow(fwobj);
    }

    // Contexto do Firewall em questão (pf)
    name[0].id = CORBA::string_dup(pfname.c_str());
```

```
    name[0].kind = CORBA::string_dup("");
    CosNaming::NamingContext_var ncx = ncfw->new_context();
    try {
ncfw->bind_context(name, ncx);
    } catch (CosNaming::NamingContext::AlreadyBound ab) {
cerr << "Contexto Firewall/" << pfname.c_str() << " já criado, recriando..." << endl;
ncfw->rebind_context(name, ncx);
    }


    // Criacao de um objeto PacketFilter
    PacketFilter_impl * fw = new PacketFilter_impl;
    // Ativa objeto PacketFilter
    oid = poa->activate_object (fw);
    ref = poa->id_to_reference (oid.in());
    // Registrando Objeto PacketFilter
    name[0].id = CORBA::string_dup("PacketFilter");
    name[0].kind = CORBA::string_dup("");
    cerr << "Registrando Objeto PacketFilter..." << endl;
    ncx->rebind(name, ref);

    // Criacao de um objeto NAT
    Nat_impl * nat = new Nat_impl();
    // Ativa objeto NAT
    oid = poa->activate_object (nat);
    ref = poa->id_to_reference (oid.in());
    // Registrando Objeto NAT
    name[0].id = CORBA::string_dup("Nat");
    name[0].kind = CORBA::string_dup("");
    cerr << "Registrando Objeto Nat..." << endl;
    ncx->rebind(name, ref);

    // Resolvendo Contexto SigSecLog
    bool sigsec_on = TRUE;
    CORBA::Object_var obj;
    name[0].id = CORBA::string_dup("SigSecLog");
    name[0].kind = CORBA::string_dup("");
    try {
obj = nc->resolve(name);
    } catch (CosNaming::NamingContext::NotFound ab) {
        cerr << "Contexto SigSecLog nao encontrado..." << endl;
sigsec_on = FALSE;
    }

    // Resolvendo Objeto FwLog
    if (sigsec_on) {
     nc = CosNaming::NamingContext::_narrow(obj);
     name[0].id = CORBA::string_dup("FwLog");
      name[0].kind = CORBA::string_dup("");
      try {
      obj = nc->resolve(name);
      } catch (CosNaming::NamingContext::NotFound ab) {
            cerr << "Objeto FwLog nao encontrado..." << endl;
    sigsec_on = FALSE;
      }
    }
```

```
    Firewall::FwLog_var logfw;
    if (sigsec_on) logfw = Firewall::FwLog::_narrow(obj);


    mgr->activate ();

    // Create pid file  (stolen from Snort :) )
    FILE *pid_file = fopen(pid_filename.c_str(), "w");

    if(pid_file)
    {
        fprintf(pid_file, "%d\n", (int) getpid());
        fclose(pid_file);
    }

    signal(SIGTERM, Exit);
    signal(SIGQUIT, Exit);


    Firewall::FwLog::Log log;
    while (1) {
if (sigsec_on && get_log(&log)) {
    log.fwname = CORBA::string_dup(pfname.c_str());
    logfw->SendLog(log);
}
if (orb->work_pending())
         orb->perform_work();
    }

    poa->destroy (TRUE, TRUE);
    delete fw;
    delete nat;

    Exit(SIGQUIT);
}

void Exit(int sig) {
    cerr << "Finishing SIGSEC PF Driver..." << endl;
    close_log();
    unlink(pid_filename.c_str());
    exit(0);
}
```

C.2.2 sigsec_pfd.h: Classes, arquivos de inclusão e protótipos de funções para o arquivo sigsec_pfd.cc

```
/* SigSec PF CORBA Driver  */
/* by André S. B. */

#include "sigsec_fw.h"
#include "sigsec_fw_impl.h"
#include <sys/types.h>
#include <unistd.h>
```

```
string pfconf="";
string natconf="";
string fixiface="";

void Exit(int sig);
string pid_filename = "/var/run/sigsec_pfd.pid";
extern void init_log(void);
extern bool get_log(Firewall::FwLog::Log *);
extern void close_log(void);
```

C.2.3 `Makefile`: Para compilação do *driver* do OpenBSD PF

```
SUBDIRS= Parser Builder Logger
OBJS= ../IDLs/sigsec_fw.o sigsec_fw_impl.o Parser/parser.o Builder/builder.o
      Logger/logger.o sigsec_pfd.o
LIBS= -lmicocoss2.3.7 -lmico2.3.7 -ly -ll -lpcap
CFLAGS= -I../IDLs -O2 -Wall -Werror
INSTDIR=/usr/local/bin

all: start subdirs sigsec_pfd end

start:
@echo
@echo "Changing to PF_Driver/, compiling PF Driver for OpenBSD..."
@echo

subdirs:
@for i in ${SUBDIRS}; do cd $$i; ${MAKE}; cd ..; done

sigsec_fw_impl.o : sigsec_fw_impl.cc sigsec_fw_impl.h
mico-c++ ${CFLAGS} -c sigsec_fw_impl.cc -o sigsec_fw_impl.o

sigsec_pfd.o : sigsec_pfd.cc sigsec_pfd.h
mico-c++ ${CFLAGS} -c sigsec_pfd.cc -o sigsec_pfd.o

sigsec_pfd: ${OBJS}
mico-ld -I. ${OBJS} -o sigsec_pfd ${LIBS}

end:
@echo "SigSec OpenBSD PF Driver compiled!"

install:
@echo "Installing OpenBSD pf Driver..."
install -c sigsec_pfd ${INSTDIR}

clean:
set -e; for i in ${SUBDIRS}; do cd $$i; ${MAKE} clean; cd ..; done
rm -f *~ *core *.o sigsec_pfd
```

C.3 *Driver* CORBA para o IDS Snort

C.3.1 sigsec_snortd.cc: Programa principal do *driver* do Snort

```
/* SigSec Snort CORBA Driver  */
/* by André S. B. */

#include "sigsec_snortd.h"


int main(int argc, char *argv[]) {

    //Obtendo o hostname
    char hostname[50];
    gethostname(hostname, sizeof(hostname));

    // Abrindo arquivo de configuracao para obter os dados do firewall
    string line;
    string snortname="";
    string snortport="";
    string naminghost="";
    string namingport="";
    ifstream conf("/etc/sigsec.conf");
    while(conf) {
         conf >> line;
         if (line.substr(0,11) == "SNORT_NAME=") {
                snortname = line.substr(11);
         }
         if (line.substr(0,11) == "SNORT_PORT=") {
                snortport = line.substr(8);
         }
         if (line.substr(0,11) == "SNORT_PATH=") {
                snortpath = line.substr(11);
         }
         if (line.substr(0,11) == "SNORT_CONF=") {
                snortconf = line.substr(11);
         }
         if (line.substr(0,12) == "SNORT_ALERT=") {
                snortalert = line.substr(12);
         }
 if (line.substr(0,12) == "NAMING_HOST=") {
                naminghost = line.substr(12);
         }
         if (line.substr(0,12) == "NAMING_PORT=") {
                namingport = line.substr(12);
         }
    }

    conf.close();

    if (snortport == "") snortport = "9002";
    if (snortpath == "") snortpath = "/usr/local/bin/snort";
    if (snortconf == "") snortconf = "/etc/snort";
    if (snortalert == "") snortalert = "/var/log/snort/alert";
    if (naminghost == "") naminghost = hostname;
```

```
    if (namingport == "") namingport = "9000";
    if (snortname == "") {
        cerr << "No SNORT_NAME found in /etc/sigsec.conf" << endl;
        exit(0);
    }

    init_alert();

    // Inicializacao do CORBA
    argv[argc] = new char[100];
    snprintf(argv[argc], 100, "-ORBIIOPAddr=inet:%s:%s", hostname, snortport.c_str());
    argc++;
    CORBA::ORB_var orb = CORBA::ORB_init( argc, argv );
    CORBA::Object_var poaobj = orb->resolve_initial_references ("RootPOA");
    PortableServer::POA_var poa = PortableServer::POA::_narrow (poaobj);
    PortableServer::POAManager_var mgr = poa->the_POAManager();

    PortableServer::ObjectId_var oid;
    CORBA::Object_var ref;

    // Resolvendo referencia para o Naming
    char refnaming[100];
    snprintf(refnaming, sizeof(refnaming), "corbaloc::%s:%s/NameService",
             naminghost.c_str(), namingport.c_str());
    CORBA::Object_var nsobj;
    try {
        nsobj = orb->string_to_object(refnaming);
    } catch (CORBA::ORB::InvalidName in){
        cerr << "Naming Service invalido!";
    } catch (CORBA::BAD_PARAM bp) {
        cerr << "Naming: parametro invalido!";
    }
    CosNaming::NamingContext_var nc = CosNaming::NamingContext::_narrow(nsobj);


    // Registrando no Naming
    // Contexto IDS
    CosNaming::Name name;
    name.length(1);
    name[0].id = CORBA::string_dup("IDS");
    name[0].kind = CORBA::string_dup("");
    CosNaming::NamingContext_var ncids;
    try {
ncids = nc->bind_new_context(name);
    } catch (CosNaming::NamingContext::AlreadyBound ab) {
cerr << "Contexto IDS já criado, resolvendo..." << endl;
        CORBA::Object_var idsobj = nc->resolve(name);
        ncids = CosNaming::NamingContext::_narrow(idsobj);
    }

    // Contexto do IDS em questão (snort)
    name[0].id = CORBA::string_dup(snortname.c_str());
    name[0].kind = CORBA::string_dup("");
    CosNaming::NamingContext_var ncx = ncids->new_context();
    try {
ncids->bind_context(name, ncx);
```

```
    } catch (CosNaming::NamingContext::AlreadyBound ab) {
cerr << "Contexto IDS/" << snortname.c_str() << " já criado, recriando..." << endl;
ncids->rebind_context(name, ncx);
    }


    // Criacao de um objeto NIDS
    NIDS_impl * nids = new NIDS_impl;
    // Ativa objeto NIDS
    oid = poa->activate_object (nids);
    ref = poa->id_to_reference (oid.in());
    // Registrando Objeto NIDS
    name[0].id = CORBA::string_dup("NIDS");
    name[0].kind = CORBA::string_dup("");
    cerr << "Registrando Objeto NIDS..." << endl;
    ncx->rebind(name, ref);

    // Resolvendo Contexto SigSecLog
    bool sigsec_on = TRUE;
    CORBA::Object_var obj;
    name[0].id = CORBA::string_dup("SigSecLog");
    name[0].kind = CORBA::string_dup("");
    try {
obj = nc->resolve(name);
    } catch (CosNaming::NamingContext::NotFound ab) {
        cerr << "Contexto SigSecLog nao encontrado..." << endl;
sigsec_on = FALSE;
    }

    // Resolvendo Objeto IDSAlert
    if (sigsec_on) {
     nc = CosNaming::NamingContext::_narrow(obj);
     name[0].id = CORBA::string_dup("IDSAlert");
     name[0].kind = CORBA::string_dup("");
      try {
         obj = nc->resolve(name);
      } catch (CosNaming::NamingContext::NotFound ab) {
            cerr << "Objeto IDSAlert nao encontrado..." << endl;
         sigsec_on = FALSE;
      }
    }

    IDS::IDSAlert_var alertids;
    if (sigsec_on) alertids = IDS::IDSAlert::_narrow(obj);


    mgr->activate ();

    // Create pid file  (stolen from Snort :) )
    FILE *pid_file = fopen(pid_filename.c_str(), "w");

    if(pid_file)
    {
        fprintf(pid_file, "%d\n", (int) getpid());
        fclose(pid_file);
    }
```

```
    signal(SIGTERM, Exit);
    signal(SIGQUIT, Exit);

    IDS::IDSAlert::Alert alert;
    while (1) {
if (sigsec_on && get_alert(&alert)) {
    alert.idsname = CORBA::string_dup(snortname.c_str());
    alertids->SendAlert(alert);
}
if (orb->work_pending())
         orb->perform_work();
    }

    close_alert();
    poa->destroy (TRUE, TRUE);
    delete nids;

    Exit(0);
}

void Exit(int sig) {
    cerr << "Finishing SIGSEC Snort Driver..." << endl;
    unlink(pid_filename.c_str());
    exit(0);
}
```

C.3.2 sigsec_snortd.h: Classes, arquivos de inclusão e protótipos de funções para o arquivo sigsec_snortd.cc

```
/* SigSec Snort CORBA Driver  */
/* by André S. B. */

#include "sigsec_ids.h"
#include "sigsec_ids_impl.h"
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

string snortpath="";
string snortconf="";
string snortalert="";

void Exit(int sig);
string pid_filename = "/var/run/sigsec_snortd.pid";
extern void init_alert(void);
extern bool get_alert(IDS::IDSAlert::Alert *);
extern void close_alert(void);
```

C.3.3 Makefile: Para compilação do *driver* do Snort

```
SUBDIRS= Parser Builder Logger
OBJS= ../IDLs/sigsec_ids.o sigsec_ids_impl.o Parser/parser.o Builder/builder.o
      Logger/logger.o sigsec_snortd.o
LIBS= -lmicocoss2.3.7 -lmico2.3.7 -ly -ll
```

```
CFLAGS= -I../IDLs -O2 -Wall -Werror
INSTDIR=/usr/local/bin

all: start subdirs sigsec_snortd end

start:
@echo
@echo "Changing to Snort_Driver/, compiling Snort Driver for OpenBSD..."
@echo

subdirs:
@for i in ${SUBDIRS}; do cd $$i; ${MAKE}; cd ..; done

sigsec_ids_impl.o : sigsec_ids_impl.cc sigsec_ids_impl.h
mico-c++ ${CFLAGS} -c sigsec_ids_impl.cc -o sigsec_ids_impl.o

sigsec_snortd.o : sigsec_snortd.cc sigsec_snortd.h
mico-c++ ${CFLAGS} -c sigsec_snortd.cc -o sigsec_snortd.o

sigsec_snortd: ${OBJS}
mico-ld -I. ${OBJS} -o sigsec_snortd ${LIBS}

end:
@echo "SigSec OpenBSD Snort Driver compiled!"

install:
@echo "Installing Snort Driver..."
install -c sigsec_snortd ${INSTDIR}

clean:
set -e; for i in ${SUBDIRS}; do cd $$i; ${MAKE} clean; cd ..; done
rm -f *~ *core *.o sigsec_snortd
```

C.4 Concentrador de Logs (SIGSEC *Log Daemon*)

C.4.1 `sigsec_logd.cc`: Programa principal do SIGSEC *Log Daemon*

```
#include "sigsec_logd.h"

int main( int argc, char *argv[] )
{
    //Obtendo o hostname
    char hostname[50];
    gethostname(hostname, sizeof(hostname));

    // Abrindo arquivo de configuracao para obter os dados de condexao ao database
    string line;
    string dbhost="";
    string dbuser="";
    string dbpassword="";
    string naminghost="";
    string namingport="";
```

```
string logdport="";
ifstream conf("/etc/sigsec.conf");
while(conf) {
     conf >> line;
     if (line.substr(0,10) == "LOGD_PORT=") {
            logdport = line.substr(10);
     }
     if (line.substr(0,8) == "DB_HOST=") {
            dbhost = line.substr(8);
     }
     if (line.substr(0,8) == "DB_USER=") {
            dbuser = line.substr(8);
     }
     if (line.substr(0,12) == "DB_PASSWORD=") {
            dbpassword = line.substr(12);
     }
     if (line.substr(0,12) == "NAMING_HOST=") {
            naminghost = line.substr(12);
     }
     if (line.substr(0,12) == "NAMING_PORT=") {
            namingport = line.substr(12);
     }
}

conf.close();

if (dbhost == "") dbhost = "localhost";
if (dbuser == "") dbuser = "root";
if (naminghost == "") naminghost = hostname;
if (namingport == "") namingport = "9000";
if (logdport == "") logdport = "9003";
if (dbpassword == "") {
    cerr << "No DB_PASSWORD found in /etc/sigsec.conf" << endl;
exit(0);
}

CORBA::Object_var obj;
CosNaming::NamingContext_var nc;
CosNaming::Name name;

// Inicializacao do CORBA
argv[argc] = new char[100];
snprintf(argv[argc], 100, "-ORBIIOPAddr=inet:%s:%s", hostname, logdport.c_str());
argc++;
CORBA::ORB_var orb = CORBA::ORB_init( argc, argv );
CORBA::Object_var poaobj = orb->resolve_initial_references ("RootPOA");
PortableServer::POA_var poa = PortableServer::POA::_narrow (poaobj);
PortableServer::POAManager_var mgr = poa->the_POAManager();
PortableServer::ObjectId_var oid;
CORBA::Object_var ref;

// Resolvendo referencia para o Naming
char refnaming[100];
snprintf(refnaming, sizeof(refnaming), "corbaloc::%s:%s/NameService",
         naminghost.c_str(), namingport.c_str());
```

```
    CORBA::Object_var nsobj;
    try {
        nsobj = orb->string_to_object(refnaming);
    } catch (CORBA::ORB::InvalidName in){
        cerr << "Naming Service invalido!";
    } catch (CORBA::BAD_PARAM bp) {
        cerr << "Naming: parametro invalido!";
    }

    nc = CosNaming::NamingContext::_narrow(nsobj);

    // Registrando SigSecLog no Naming
    name.length(1);
    name[0].id = CORBA::string_dup("SigSecLog");
    name[0].kind = CORBA::string_dup("");
    CosNaming::NamingContext_var nclog;
    try {
        nclog = nc->bind_new_context(name);
    } catch (CosNaming::NamingContext::AlreadyBound ab) {
        cerr << "Contexto SigSecLog já criado, resolvendo..." << endl;
        CORBA::Object_var logobj = nc->resolve(name);
        nclog = CosNaming::NamingContext::_narrow(logobj);
    }

    // Criacao de um objeto FwLog
    FwLog_impl * fwlog = new FwLog_impl;
    // Ativa objeto FwLog
    oid = poa->activate_object (fwlog);
    ref = poa->id_to_reference (oid.in());
    // Registrando Objeto FwLog
    name[0].id = CORBA::string_dup("FwLog");
    name[0].kind = CORBA::string_dup("");
    cerr << "Registrando Objeto FwLog..." << endl;
    nclog->rebind(name, ref);

    // Criacao de um objeto IDSAlert
    IDSAlert_impl * idsalert = new IDSAlert_impl;
    // Ativa objeto IDSAlert
    oid = poa->activate_object (idsalert);
    ref = poa->id_to_reference (oid.in());
    // Registrando Objeto IDSAlert
    name[0].id = CORBA::string_dup("IDSAlert");
    name[0].kind = CORBA::string_dup("");
    cerr << "Registrando Objeto IDSAlert..." << endl;
    nclog->rebind(name, ref);

    mgr->activate ();


    // Conectando no MySQL
    mysql_init(&mysql);
    if (!mysql_real_connect(&mysql, dbhost.c_str(), dbuser.c_str() , dbpassword.c_str(),
        "sigsec", 0, NULL, 0)) {
cerr << "Impossivel conectar no MySQL SGBD" << endl;
cerr << mysql_errno(&mysql) << endl;
exit(1);
```

```
    }

    // Create pid file  (stolen from Snort :) )
    FILE *pid_file = fopen(pid_filename.c_str(), "w");

    if(pid_file)
    {
        fprintf(pid_file, "%d\n", (int) getpid());
        fclose(pid_file);
    }

    signal(SIGTERM, Exit);
    signal(SIGQUIT, Exit);

    orb->run();

    poa->destroy (TRUE, TRUE);
    delete fwlog;
    delete idsalert;

    Exit(SIGQUIT);
}

void Exit(int sig) {
    cerr << "Finishing SIGSEC Log Daemon..." << endl;
    unlink(pid_filename.c_str());
    mysql_close(&mysql);
    exit(0);
}
```

C.4.2 sigsec_logd.h: Classes, arquivos de inclusão e protótipos de funções para o arquivo sigsec_logd.cc

```
#include "sigsec_logd_impl.h"

char ipdot[15];

MYSQL mysql;
string pid_filename = "/var/run/sigsec_logd.pid";
const int NFWS = 200;
const int NIDS = 200;

void Exit(int sig);
```

C.5 *Script* CGI de gerência (SIGSEC CGI)

C.5.1 sigsec_cgi.cc: Programa principal do SIGSEC CGI

```
#include "sigsec_cgi.h"

extern const int NOBJS;
CGIData data;
CORBA::ORB_var orb;
```

```
int main(int argc, char **argv) {


  // Inicializacao do CORBA
  orb = CORBA::ORB_init( argc, argv );

  CGIParser parser;
  parser.parse();

  data = parser.getData();

  cout << "Content-type: text/html" << endl << endl;


  if      (data["action"] == "welcome") welcome();
  else if (data["action"] == "fwmenu")  firewall_menu();
  else if (data["action"] == "fwbody")  firewall_body();
  else if (data["action"] == "fwon")    firewall_on();
  else if (data["action"] == "fwoff")   firewall_off();
  else if (data["action"] == "fwres")   firewall_res();
  else if (data["action"] == "idsmenu") ids_menu();
  else if (data["action"] == "idsbody") ids_body();
  else if (data["action"] == "idson")   ids_on();
  else if (data["action"] == "idsoff")  ids_off();
  else if (data["action"] == "idsres")  ids_res();
  else if (data["action"] == "logmenu") logalert_menu();
  else if (data["action"] == "logbody") logalert_body();
  else cout << "<HTML><HEAD></HEAD><BODY><H3>Request Error!</H3>";

  cout << "</BODY></HTML>";
}

CosNaming::NamingContext_var
GetNamingRef() {

   CORBA::Object_var nsobj;
   CosNaming::NamingContext_var nc;

   //Obtendo o hostname
   char hostname[50];
   gethostname(hostname, sizeof(hostname));

   // Abrindo arquivo de configuracao para obter a porta do Naming
   string line;
   string naminghost="";
   string namingport="";
   ifstream conf("/etc/sigsec.conf");
   while(conf) {
conf >> line;
if (line.substr(0,12) == "NAMING_HOST=") {
naminghost = line.substr(12);
}
if (line.substr(0,12) == "NAMING_PORT=") {
namingport = line.substr(12);
}
   }
```

```
   conf.close();

   if (naminghost == "") naminghost = hostname;
   if (namingport == "") namingport = "9000";

   char refnaming[100];
   snprintf(refnaming, sizeof(refnaming), "corbaloc::%s:%s/NameService",
            naminghost.c_str(), namingport.c_str());


   //Resolvendo referencia para o Naming
   try {
     nsobj = orb->string_to_object(refnaming);
   } catch (CORBA::ORB::InvalidName in){
     cout << "Naming Service invalido!";
   } catch (CORBA::BAD_PARAM bp) {
     cout << "Naming: parametro invalido!";
   }

   try {
     nc = CosNaming::NamingContext::_narrow(nsobj);
   } catch (CORBA::COMM_FAILURE cf) {
     cout << "Naming: Falha na comunicacao!";
   }

   return nc;
}

CosNaming::NamingContext_var
GetFirewallRef() {
   CORBA::Object_var obj;
   CosNaming::NamingContext_var nc;
   CosNaming::Name name;

   nc = GetNamingRef();

   // Obtendo referencia para o contexto Firewall
   name.length(1);
   name[0].id = CORBA::string_dup("Firewall");
   name[0].kind = CORBA::string_dup("");
   try {
       obj = nc->resolve(name);
   }
   catch(CosNaming::NamingContext::NotFound nf) {
       cout << "Nenhum Firewall encontrado!" << endl;
   }
   catch(CosNaming::NamingContext::CannotProceed cp) {
       cout << "Erro em resolve (Firewall), impossivel continuar!" << endl;
   }

   nc = CosNaming::NamingContext::_narrow(obj);

   return nc;
}
```

```
CosNaming::NamingContext_var
GetIDSRef() {
   CORBA::Object_var obj;
   CosNaming::NamingContext_var nc;
   CosNaming::Name name;

   nc = GetNamingRef();

   // Obtendo referencia para o contexto IDS
   name.length(1);
   name[0].id = CORBA::string_dup("IDS");
   name[0].kind = CORBA::string_dup("");
   try {
       obj = nc->resolve(name);
   }
   catch(CosNaming::NamingContext::NotFound nf) {
       cout << "Nenhum IDS encontrado!" << endl;
   }
   catch(CosNaming::NamingContext::CannotProceed cp) {
       cout << "Erro em resolve (IDS), impossivel continuar!" << endl;
   }

   nc = CosNaming::NamingContext::_narrow(obj);

   return nc;
}

Firewall::PacketFilter_var
GetPacketFilterRef() {

  CORBA::Object_var obj;
  CosNaming::NamingContext_var nc;
  CosNaming::Name name;

  nc = GetFirewallRef();
  // obtendo referencia para o contexto do Firewall especifico
  name.length(1);
  name[0].id = data["fw"].c_str();
  name[0].kind = CORBA::string_dup("");
  try {
      obj = nc->resolve(name);
  }
  catch(CosNaming::NamingContext::NotFound nf) {
      cout << "Firewall : " << data["fw"] << "nao encontrado!" << endl;
  }
  catch(CosNaming::NamingContext::CannotProceed cp) {
      cout << "Erro em resolve (Firewall especifico), impossivel continuar!" << endl;
  }
  nc = CosNaming::NamingContext::_narrow(obj);

  name[0].id = CORBA::string_dup("PacketFilter");
  name[0].kind = CORBA::string_dup("");
  try {
      obj = nc->resolve(name);
  }
  catch(CosNaming::NamingContext::NotFound nf) {
```

```
        cout << "PacketFilter nao encontrado!" << endl;
    }
    catch(CosNaming::NamingContext::CannotProceed cp) {
        cout << "Erro em resolve (PacketFilter), impossivel continuar!" << endl;
    }

    return ( Firewall::PacketFilter::_narrow(obj) );

}

IDS::NIDS_var
GetNIDSRef() {
    CORBA::Object_var obj;
    CosNaming::NamingContext_var nc;
    CosNaming::Name name;

    nc = GetIDSRef();
    // obtendo referencia para o contexto do IDS especifico
    name.length(1);
    name[0].id = data["ids"].c_str();
    name[0].kind = CORBA::string_dup("");
    try {
        obj = nc->resolve(name);
    }
    catch(CosNaming::NamingContext::NotFound nf) {
        cout << "IDS : " << data["ids"] << "nao encontrado!" << endl;
    }
    catch(CosNaming::NamingContext::CannotProceed cp) {
        cout << "Erro em resolve (IDS especifico), impossivel continuar!" << endl;
    }
    nc = CosNaming::NamingContext::_narrow(obj);

    name[0].id = CORBA::string_dup("NIDS");
    name[0].kind = CORBA::string_dup("");
    try {
        obj = nc->resolve(name);
    }
    catch(CosNaming::NamingContext::NotFound nf) {
        cout << "NIDS nao encontrado!" << endl;
    }
    catch(CosNaming::NamingContext::CannotProceed cp) {
        cout << "Erro em resolve (NIDS), impossivel continuar!" << endl;
    }

    return ( IDS::NIDS::_narrow(obj) );

}

void firewall_menu() {
   CosNaming::NamingContext_var nc;

   header_top();

   nc = GetFirewallRef();

   if (!CORBA::is_nil(nc)) {
```

```
        // Lista bindings
        CosNaming::BindingIterator_var it;
        CosNaming::BindingList_var bl;
        nc->list(NOBJS, bl, it);

        cout << "<FORM METHOD=\"POST\" ACTION=\"/cgi-bin/sigsec_cgi\" TARGET=\"body\">";
        cout << "Managing Firewalls ";
        cout << "<SELECT NAME=\"fw\" ALIGN=\"top\">";

        for (CORBA::ULong i = 0; i < bl->length(); i++) {
            cout << "<OPTION VALUE=\"";
            cout << bl[i].binding_name[0].id << "\">" << bl[i].binding_name[0].id;
            cout << "</OPTION>";
        }

        cout << "</SELECT>";
        cout << " <INPUT TYPE=\"submit\" VALUE=\" GO \">";
        cout << "<INPUT TYPE=\"hidden\" NAME=\"action\" VALUE=\"fwbody\">";
        cout << "</FORM>";
    }
}

void ids_menu() {
    CosNaming::NamingContext_var nc;

    header_top();

    nc = GetIDSRef();

    if (!CORBA::is_nil(nc)) {

        // Lista bindings
        CosNaming::BindingIterator_var it;
        CosNaming::BindingList_var bl;
        nc->list(NOBJS, bl, it);

        cout << "<FORM METHOD=\"POST\" ACTION=\"/cgi-bin/sigsec_cgi\" TARGET=\"body\">";
        cout << "Managing IDS ";
        cout << "<SELECT NAME=\"ids\" ALIGN=\"top\">";

        for (CORBA::ULong i = 0; i < bl->length(); i++) {
            cout << "<OPTION VALUE=\"";
            cout << bl[i].binding_name[0].id << "\">" << bl[i].binding_name[0].id;
            cout << "</OPTION>";
        }

        cout << "</SELECT>";
        cout << " <INPUT TYPE=\"submit\" VALUE=\" GO \">";
        cout << "<INPUT TYPE=\"hidden\" NAME=\"action\" VALUE=\"idsbody\">";
        cout << "</FORM>";
    }

}

void ids_body() {
  IDS::NIDS_var nids;
```

```
  header_ids_body();

  nids = GetNIDSRef();

  try {
     if (nids->State()) {
  cout << "<B>Active</B>";
     } else {
          cout << "<B>Inactive</B>";
     }
  } catch ( CORBA::COMM_FAILURE cf) {
  cout << "<B>IDS not reponding!</B>";
  }
}

void firewall_body() {
  Firewall::PacketFilter_var pf;

  header_fw_body();

  pf = GetPacketFilterRef();

  try {
     if (pf->State()) {
  cout << "<B>Active</B>";
     } else {
          cout << "<B>Inactive</B>";
     }
  } catch ( CORBA::COMM_FAILURE cf) {
  cout << "<B>Firewall not reponding!</B>";
  }
}

void firewall_on() {
  Firewall::PacketFilter_var fw;

  fw = GetPacketFilterRef();

  fw->State(1);

  firewall_body();
}

void firewall_off() {
  Firewall::PacketFilter_var fw;

  fw = GetPacketFilterRef();

  fw->State(0);

  firewall_body();
}

void firewall_res() {
  Firewall::PacketFilter_var fw;
```

```
  fw = GetPacketFilterRef();

  fw->Restart();

  firewall_body();
}

void ids_on() {
  IDS::NIDS_var nids;

  nids = GetNIDSRef();

  nids->State(1);

  sleep(1);

  ids_body();
}

void ids_off() {
  IDS::NIDS_var nids;

  nids = GetNIDSRef();

  nids->State(0);

  ids_body();
}

void ids_res() {
  IDS::NIDS_var nids;

  nids = GetNIDSRef();

  nids->Restart();

  sleep(1);

  ids_body();
}

void logalert_menu() {

  header_top();

  mysql_init(&mysql);
  if (!mysql_real_connect(&mysql, "localhost", "root", "cat&odie", "sigsec", 0, NULL, 0)) {
     cerr << "Impossivel concetar no MySQL SGDB" << endl;
     cerr << mysql_errno(&mysql) << endl;
     exit(1);
  }

  MYSQL_RES    *res=0;
  MYSQL_ROW    row;
```

```
  if (mysql_query(&mysql,"SELECT objid, type, name FROM objects")) {
     cerr << mysql_error(&mysql) << endl;
  }

  res = mysql_use_result(&mysql);

  cout << "<FORM METHOD=\"POST\" ACTION=\"/cgi-bin/sigsec_cgi\" TARGET=\"body\">";
  cout << "Logs & Alerts ";
  cout << "<SELECT NAME=\"objid\" ALIGN=\"top\">";

  row = mysql_fetch_row(res);
  while(row != NULL) {
    cout << "<OPTION VALUE=\"";
    cout << row[0] << "\">" << row[1] << " : " << row[2];
    cout << "</OPTION>";
    row = mysql_fetch_row(res);
  }

  cout << "</SELECT>";
  cout << " <INPUT TYPE=\"submit\" VALUE=\" GO \">";
  cout << "<INPUT TYPE=\"hidden\" NAME=\"action\" VALUE=\"logbody\">";
  cout << "</FORM>";
  mysql_free_result(res);
  mysql_close(&mysql);
}

void logalert_body() {

  header_log_body();

  mysql_init(&mysql);
  if (!mysql_real_connect(&mysql, "localhost", "root", "cat&odie", "sigsec", 0, NULL, 0)) {
     cerr << "Impossivel concetar no MySQL SGDB" << endl;
     cerr << mysql_errno(&mysql) << endl;
     exit(1);
  }

  char query[1000];
  MYSQL_RES    *res=0;
  MYSQL_ROW    row;

  snprintf(query, sizeof(query), "SELECT * from idsalert WHERE objid=\"%s\"",
           data["objid"].c_str());
  if (mysql_query(&mysql,query)) {
     cerr << mysql_error(&mysql) << endl;
  }

  res = mysql_use_result(&mysql);

  cout << "<TABLE BORDER=1>";
  row = mysql_fetch_row(res);
  while(row != NULL) {
    cout << "<TR>";
    cout << "<TD>" << row[1] << "</TD>";
    cout << "<TD>" << row[2] << "</TD>";
    cout << "<TD>" << row[3] << "</TD>";
```

```
    cout << "<TD>" << row[4] << "</TD>";
    cout << "<TD>" << row[5] << "</TD>";
    cout << "<TD>" << row[6] << "</TD>";
    cout << "</TR>";
    row = mysql_fetch_row(res);
  }
  cout << "</TABLE>";

  mysql_free_result(res);
  mysql_close(&mysql);

}

void header_top() {
  cout << "<HTML><HEAD></HEAD><BODY BGCOLOR=\"#0000FF\" LINK=\"#BBBBFF\" VLINK=\"#BBBBFF\"
          ALINK=\"#FFFFFF\">";
  cout << "<FONT COLOR=\"#FFFFFF\" FACE=\"verdana,arial,helvetica\" SIZE=\"2\">";
  cout << "<H3>SIGSEC Web Admin Interface</H3>";

  cout << "[ <A HREF=\"/cgi-bin/sigsec_cgi?action=fwmenu\" TARGET=\"menu\"><B>
          Firewall</B></A> | ";
  cout << "<A HREF=\"/cgi-bin/sigsec_cgi?action=idsmenu\" TARGET=\"menu\"><B>
          IDS</B></A> | ";
  cout << "<A HREF=\"/cgi-bin/sigsec_cgi?action=logmenu\" TARGET=\"menu\"><B>
          Logs & Alerts</B></A> ]";
  cout << "<BR><BR>";
}

void header_fw_body() {
  cout << "<HTML><HEAD></HEAD><BODY BGCOLOR=\"#FFFFFF\" LINK=\"#BB1111\" VLINK=\"#BB1111\"
          ALINK=\"#0000FF\">";
  cout << "<FONT COLOR=\"#222222\" FACE=\"verdana,arial,helvetica\" SIZE=\"2\">";
  cout << "[ <A HREF=\"/cgi-bin/sigsec_cgi?action=fwon&fw=" << data["fw"];
  cout << "\" TARGET=\"body\"><B>On</B></A>";
  cout << " | <A HREF=\"/cgi-bin/sigsec_cgi?action=fwoff&fw=" << data["fw"];
  cout << "\" TARGET=\"body\"><B>Off</B></A>";
  cout << " | <A HREF=\"/cgi-bin/sigsec_cgi?action=fwres&fw=" << data["fw"];
  cout << "\" TARGET=\"body\"><B>Restart</B></A> ]";
  cout << "<BR>Firewall Manager :: " << data["fw"] << "<BR>";
}

void header_ids_body() {
  cout << "<HTML><HEAD></HEAD><BODY BGCOLOR=\"#FFFFFF\" LINK=\"#BB1111\" VLINK=\"#BB1111\"
          ALINK=\"#0000FF\">";
  cout << "<FONT COLOR=\"#222222\" FACE=\"verdana,arial,helvetica\" SIZE=\"2\">";
  cout << "[ <A HREF=\"/cgi-bin/sigsec_cgi?action=idson&ids=" << data["ids"];
  cout << "\" TARGET=\"body\"><B>On</B></A>";
  cout << " | <A HREF=\"/cgi-bin/sigsec_cgi?action=idsoff&ids=" << data["ids"];
  cout << "\" TARGET=\"body\"><B>Off</B></A>";
  cout << " | <A HREF=\"/cgi-bin/sigsec_cgi?action=idsres&ids=" << data["ids"];
  cout << "\" TARGET=\"body\"><B>Restart</B></A> ]";
  cout << "<BR>IDS Manager :: " << data["ids"] << "<BR>";
}

void header_log_body() {
  cout << "<HTML><HEAD></HEAD><BODY BGCOLOR=\"#FFFFFF\" LINK=\"#BB1111\" VLINK=\"#BB1111\"
```

```
          ALINK=\"#0000FF\">";
  cout << "<FONT COLOR=\"#222222\" FACE=\"verdana,arial,helvetica\" SIZE=\"2\">";
  //cout << "[ <A HREF=\"/cgi-bin/sigsec_cgi?action=" << data[""];
  //cout << "\" TARGET=\"body\"><B> </B></A>";
  //cout << " | <A HREF=\"/cgi-bin/sigsec_cgi?action=" << data[""];
  //cout << "\" TARGET=\"body\"><B> </B></A> ]";
  cout << "Alert & Log Manager:<BR>";
}

void welcome() {
  cout << "<HTML><HEAD></HEAD><BODY BGCOLOR=\"#FFFFFF\" LINK=\"#BBBBFF\" VLINK=\"#BBBBFF\"
          ALINK=\"#0000FF\">";
  cout << "<FONT COLOR=\"#222222\" FACE=\"verdana,arial,helvetica\" SIZE=\"2\">";
  cout << "<H3>Welcome to SIGSEC Web Admin Interface</H3>";
  cout << "<BR><BR>Credits:";
  cout << "<BR>Andre S. Barbosa <<I><A HREF=\"mailto:andre@ravel.ufrj.br\">
          andre@ravel.ufrj.br</A></I>>";
}
```

C.5.2 sigsec_cgi.h: Classes, arquivos de inclusão e protótipos de funções para o arquivo sigsec_cgi.cc

```
#include <CORBA.h>
#include <mico/CosNaming.h>
#include <iostream.h>
#include <unistd.h>
#include <fstream.h>
#include <mysql.h>
#include "sigsec_fw.h"
#include "sigsec_ids.h"
#include "CGILib.h"

const int NOBJS = 100;
MYSQL mysql;

void welcome(void);
void header_top(void);
void header_fw_body(void);
void header_ids_body(void);
void header_log_body(void);
void firewall_menu(void);
void firewall_body(void);
void firewall_on(void);
void firewall_off(void);
void firewall_res(void);
void ids_menu(void);
void ids_body(void);
void ids_on(void);
void ids_off(void);
void ids_res(void);
void logalert_menu(void);
void logalert_body(void);
CosNaming::NamingContext_var GetNamingRef(void);
CosNaming::NamingContext_var GetFirewallRef(void);
CosNaming::NamingContext_var GetIDSRef(void);
```

```
IDS::NIDS_var GetNIDSRef(void);
Firewall::PacketFilter_var GetPacketFilterRef(void);
```

C.6 *Script* SQL de Criação do Banco de Dados

C.6.1 sigsec.sql

```
# SIGSEC MySQL Database
# by Andre S. Barbosa
#
# Use: mysql -p < sigsec.sql

CREATE DATABASE sigsec;

CONNECT sigsec;

CREATE TABLE users (
    userid TINYINT UNSIGNED AUTO_INCREMENT NOT NULL,
    login VARCHAR(20),
    password VARCHAR(30),
    name VARCHAR(100),
    location VARCHAR(50),
    info VARCHAR(250),
    priv        VARCHAR(250),
    PRIMARY KEY (userid)
);

CREATE TABLE objects (
    objid SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    type enum('IDS', 'FIREWALL'),
    name VARCHAR(30),
    location VARCHAR(50),
    info VARCHAR(250),
    PRIMARY KEY (objid)
);

CREATE TABLE fwlog (
    objid SMALLINT UNSIGNED,
    ruleno SMALLINT UNSIGNED,
    dtime DATETIME,
    usec INT UNSIGNED,
    length SMALLINT UNSIGNED,
    ip_src INT UNSIGNED,
    ip_dst INT UNSIGNED,
    ttl TINYINT UNSIGNED,
    proto enum('ALL', 'IP', 'TCP', 'UDP', 'TCPUDP', 'ICMP'),
    port_src SMALLINT UNSIGNED,
    port_dst SMALLINT UNSIGNED,
    tcp_flags   SET('FIN', 'SYN', 'RST', 'PSH', 'ACK', 'URG'),
    tcp_seq INT UNSIGNED,
    tcp_ack INT UNSIGNED,
    icmp_type TINYINT UNSIGNED,
    icmp_code TINYINT UNSIGNED,
    payload     VARCHAR(100)
);
```

```
CREATE TABLE idsalert (
    objid SMALLINT UNSIGNED,
    message     VARCHAR(50),
    ruleno SMALLINT UNSIGNED,
    dtime DATETIME,
    usec INT UNSIGNED,
    length SMALLINT UNSIGNED,
    ip_src INT UNSIGNED,
    ip_dst INT UNSIGNED,
    ttl TINYINT UNSIGNED,
    proto enum('ALL', 'IP', 'TCP', 'UDP', 'TCPUDP', 'ICMP'),
    port_src SMALLINT UNSIGNED,
    port_dst SMALLINT UNSIGNED,
    tcp_flags   SET('FIN', 'SYN', 'RST', 'PSH', 'ACK', 'URG'),
    tcp_seq INT UNSIGNED,
    tcp_ack INT UNSIGNED,
    icmp_type TINYINT UNSIGNED,
    icmp_code TINYINT UNSIGNED
);

FLUSH PRIVILEGES;
```