Waldirio Manhães Pinheiro
wmp@sinope.com.br
Marcelle da Silva Teixeira Pinheiro
marcelle@sinope.com.br
Fabio Machado de Oliveira
fabiomac@gmail.com

DESENVOLVIMENTO DE UM SISTEMA DISTRIBUÍDO PARA DETECÇÃO DE INTRUSÃO BASEADO EM SOFTWARE LIVRE: NIDS HYDRA

Universidade Candido Mendes – Campos

Waldirio Manhães Pinheiro
Marcelle da Silva Teixeira Pinheiro
Fabio Machado de Oliveira

DESENVOLVIMENTO DE UM SISTEMA DISTRIBUÍDO PARA DETECÇÃO DE INTRUSÃO BASEADO EM SOFTWARE LIVRE: NIDS HYDRA

Monografia apresentada à banca de projeto final do Curso de Ciência da Computação da Universidade Candido Mendes – Campos, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação.

Prof. Dalessandro Soares Vianna (Orientador)

Prof.William Vianna (Co-Orientador)

Curso de Ciência da Computação
Universidade Candido Mendes
Campos dos Goytacazes, 2005.

AGRADECIMENTOS

Em especial, agradecemos a Deus por ter nos dado esta oportunidade e aos nossos pais por terem nos apoiado durante todos estes anos.

Agradecemos aos amigos que nos deram forças nos momentos difíceis de nossa vida, pois são nestes momentos que se destacam as grandes amizades.

Agradecemos também aos professores William Vianna e Dalessandro Vianna por confiarem e aceitarem a orientação do nosso trabalho de conclusão de curso.

RESUMO

Com o advento da *Internet*, as redes de computadores tornaram-se uma grande malha que agrega milhares de máquinas e empresas interconectadas, realizando desde simples troca de e-mail a transações econômicas, o que é um chamariz considerável a intrusos. Pensando assim, foi realizado uma pesquisa onde é apresentado diversas vulnerabilidades, atacantes e formas de ataque utilizadas contra a Segurança da Informação.

Foi projetado, implementado e testado uma ferramenta NIDS (*Network Intrusion Detection System*) denominado HYDRA. Este sistema detecta tentativas de ataques remotos a partir de certos padrões baseados em regras de assinaturas. Foram utilizados *Software* livres como MySQL, PHP, Web Server Apache. Como scanners de vulnerabilidade o Nessus e o Nmap e como base de desenvolvimento a linguagem de programação C, incluindo bibliotecas de MySQL, PCAP, dentre outras.

ABSTRACT

With the advent of the Internet, the computer networks they had become a great mesh that adds thousand of machines and interconnected companies, carrying through since simple exchange of email the economic transactions, what it is a considerable decoy the intruders. Thus thinking, a research was carried through where it is presented diverse vulnerabilities, aggressors and used forms of attack against the Security of the Information.

He was projected, implemented and tested a tool NIDS (Network Intrusion Detection System) called HYDRA. This system detects attempts of remote attacks from certain standards based on rules of signatures. They had been used free Software as MySQL, PHP and Apache Web server. As scanners of vulnerability the Nessus and the Nmap and as development base the programming language C, including libraries of MySQL, PCAP, amongst others.

Sumário

Lista de Figuras

Lista de Abreviaturas e Siglas

API ..... Aplication Program Interface
ARP ..... Address Resolution Procotol
ASCII ..... American Standart Code for Information Interchange
ATM ..... Asynchronous Transfer Mode
CGI ..... Common Gateway Interface
CIDF ..... Commom Intrusion Detection Framework
CISL ..... Common Instrusion Specification Language
COMM ..... Communication in the Common Intrusion Detection Framework
CORBA ..... Common Object Request Broker Architeture
DMZ ..... DeMilitarized Zone (Zona Desmilitarizada)
DNS ..... Domain Name System
GIDO ..... General Intrusion Detection Object
GPL ..... General Public License
GRE ..... Graphics Engine
GSSAPI ..... Generic Security Services Application Programming Interface
HTML ..... HyperText Markup Language
HTTP ..... Hypertext Terminal Protocol
IAP ..... Intrusion Alert Protocol
ICMP ..... Internet Control Message Protocol
ICSA ..... International Computer Security Association
IDMEF ..... Intrusion Detection Message Exchange Format
IDS ..... Intrusion Detection System
IETF ..... Internet Enginnering Task Force
IGRP ..... Interior Gateway Routing Protocol
IHL ..... Internet Header Length
IMAP ..... Internet Message Access Protocol
IP ..... Internet Protocol
IPX ..... Internetwork Packet Exchange
ISO ..... International Organizaton for Standardization
NASL ..... Nessus Attack Scripting Language
NIDS ..... Network Intrusion Detection System
NMAP ..... Network Mapper
NSA ..... National Security Agency
OSI ..... Open Systems Interconnection
OSPF ..... Open Shortest-Path First Interior Gateway Protocol
PHP ..... Personal Home Page
POP3 ..... Post Office Protocol 3
PPP ..... Point to Point Procol
RARP ..... Reverse Address Resolution Protocol
RFC ..... Request for Comments
RIP ..... Routing Information Protocol
RPC ..... Remote Procedure Call
SALT ..... Society for Applied Learning Technology®
SGDB ..... Sistema Gerenciador de Banco de Dados
SLIP ..... Serial Line Internet Protocol
SMB ..... Server Message Block

SMTP ..... Simple Mail Transfer Protocol
SNMP ..... Simple Network Management Protocol
SPAN ..... Switched Port Analizer
SQL ..... Structure Query Language
SSH ..... Secure Shell
SSL ..... Secure Socket Layer
S/MIME ..... Secure/Multipurpose Internet Mail Extensions
TCP ..... Transfer Control Protocol
TLS ..... Transport Layer Security
UDP ..... User Datagram Protocol
VPN ..... Virtual Private Network

Capítulo 1

Apresenta algumas noções introdutórias, descrevendo objetivos e alguns fatores que motivaram o desenvolvimento do Sistema.

1 INTRODUÇÃO

Com o advento da *Internet*, as redes de computadores tornaram-se uma grande malha que agrega milhares de máquinas e empresas interconectadas, realizando desde simples troca de *e-mail* a transações econômicas, o que é um chamariz considerável à intrusos.

Baseado nas informações de várias entidades de pesquisa em segurança, tais como CERT (*Computer Emergency Response Team Coordination Center*) e ICSA (*International Computer Security Association*), pode-se afirmar que o número de incidentes reportados foi de aproximadamente 377 por dia no ano de 2003, bem maior do que o comparado com o ano de 2000, que foi de aproximadamente 60 incidentes por dia.[1] [CERI] Estas estatísticas levam a uma enorme necessidade de poder rastrear e identificar estes ataques. Os sistemas que possuem essa capacidade são chamados de IDS (*Intrusion Detection System*) e NIDS (*Network Intrusion Detection System*).

O termo *ataque,* conforme a RFC (*Request for Comments*) 2828, é uma ação inteligente que ameaça a segurança de um sistema, podendo ter sucesso ou não. Ele estará explorando vulnerabilidades no sistema alvo ou inerentes aos protocolos. Um ataque bem sucedido pode caracterizar uma invasão ou até mesmo a negação de serviços no sistema alvo (*DoS - Denial of Service*). [RFC2828]

Atualmente, os administradores de sistemas e de redes devem estar a par dos principais recursos disponíveis para buscar a implementação de um ambiente seguro, com algum grau de proteção contra os perigos mais comuns existentes em redes de computadores. Grande quantidade de pesquisas originada nestes últimos anos evidencia a crescente importância do estudo dos IDS.

O IDS visa detectar tentativas de ataques contra sistemas e redes de computadores ou, em linhas gerais contra sistemas de informação. De fato, é difícil prover um sistema de informação seguro e mantê-lo em determinado estado de segurança durante seu tempo de vida e utilização.

Algumas vezes, uma restrição legada ou operacional nem sempre permite a definição de um sistema de informação completamente seguro. Por esta razão o IDS tem a tarefa de

[1] Informações retiradas do banco de dados do CERT, órgão responsável pela segurança da *Internet*.

monitorar a utilização de tais sistemas de informação para detectar a aparição de condições inseguras. Eles detectam tentativas e atividades de abusos que podem ser de usuários legítimos do sistema de informação ou de partes externas, por abusar de seus privilégios ou explorar vulnerabilidades de segurança.

As ferramentas para segurança de computadores e redes são necessárias para proporcionar transações eletrônicas seguras. Geralmente as instituições concentravam suas defesas em ferramentas preventivas somente nos firewalls que não implementavam o IDS, ignorando assim as ferramentas de detecção de intrusão. Estes ambientes estão mudando com o surgimento de vários IDS comerciais para sistemas baseados em UNIX e Windows.

Projetar e implementar um sistema completamente seguro ainda é uma realidade distante e migrar a base de sistemas instalados para um estado confiável tomará muito tempo.

Sistemas vulneráveis que podem ser atacados a qualquer momento fazem parte do cenário atual. Logo, se ataques estão ocorrendo nos sistemas, a descoberta deve ocorrer o mais cedo possível, preferencialmente em tempo real.

Uma das formas mais comum para descobrir intrusões é a utilização dos dados das auditorias gerados pelos sistemas operacionais e ordenados em ordem cronológica de acontecimento, sendo possível à inspeção manual destes registros. Mas isto não é uma prática viável, pois estes arquivos de *logs* apresentam tamanhos consideráveis.

1.1 OBJETIVOS

Tem-se como objetivo estudar as técnicas mais comuns de ameaça às Informações, assim como estudar algumas das ferramentas existentes atualmente no mercado para prevenção da mesma, fazendo um comparativo e desenvolver uma ferramenta NIDS.

Diante do contexto citado acima, tem-se como objetivo específico nesse trabalho:

- estudar técnicas de ataque, assim como vulnerabilidades existentes que possibilitem uma invasão;
- estudar técnicas mais comuns de implementação física voltada para Segurança da Informação;
- desenvolver um sistema de computação capaz de capturar os pacotes que estão a trafegar na rede, baseado em uma série de regras pré-definidas, e armazená-los em banco de dados para verificação utilizando-se uma ferramenta gráfica via WEB;

- desenvolver uma ferramenta visual e dinâmica (WEB) para análise dos dados capturados visando uma melhor forma de visualização e conseqüentemente a análise do ocorrido.

1.2 JUSTIFICATIVA

O IDS automatiza a tarefa de analisar dados da auditoria. Estes dados são extremamente úteis, pois podem ser usados para estabelecer a culpabilidade do atacante (muito utilizado em perícias forenses) que é freqüentemente o único modo de descobrir uma atividade sem autorização, detectar a extensão dos danos e prevenir tal ataque no futuro, tornando desta forma o IDS uma ferramenta extremamente valiosa para análises em tempo real e também após a ocorrência de um ataque.

É importante então registrar informações sobre invasões que possam ocorrer para que medidas preventivas possam ser tomadas e o invasor possa ser descoberto, assim como possam ser detectadas as alterações promovidas por ele no sistema que foi invadido. A implementação de *firewalls* e outros sistemas de segurança podem vir a falhar por má configuração, pela exploração de falhas, ou pela própria característica da rede.

Um sistema NIDS baseado em sensores *sniffer* de pacotes TCP, UDP e ICMP, e ferramenta para análise dos dados visando detecção de ataque cobre uma das atribuições de um IDS. Um IDS monitora o tráfego de uma rede ininterruptamente, a fim de detectar anomalia no mesmo. [2] [STR2002]

O armazenamento das informações em uma base de dados auxilia posteriormente, pois permite uma manipulação mais eficiente e precisa. A utilização de plataforma *Open* [3] e desenvolvimento do sistema baseado em *software livre* garantem manutenção e flexibilidade para portabilidade, pois pode ser compilado em qualquer outro Sistema Operacional *OpenSource*. Para instituições públicas e empresas privadas, esse é um modelo atualmente mais barato.

[2] Matthew Strebe, consultor e escritor voltado para TI.
[3] Sistemas Operacionais sem custo de licenciamento para utilização e/ou reprodução.

Capítulo 2

Apresenta alguns tipos de atacantes e ataques mais comuns em segurança da informação, políticas de Segurança e tipos de IDS, juntamente com uma visão geral do que é um sistema de detecção de intrusões.

2 FUNDAMENTAÇÃO TEÓRICA E REVISÃO DE BIBLIOGRAFIA

A conseqüência destes estudos levou ao conhecimento de um amplo conjunto de ferramentas de segurança existente atualmente, muitas das quais são baseadas em técnicas utilizadas inicialmente em ataques. Infelizmente, as ferramentas tradicionais não tem sido suficientes para conter o surgimento crescente de ataques sofisticados, desencadeados por indivíduos que são verdadeiros estudiosos do assunto (pessoas que a cada dia exploram uma nova vulnerabilidade em uma determinada aplicação e/ou Sistema Operacional). Isto tem levado a uma pesquisa mais ousada das atividades intrusivas para que esta ameaça seja dominada, ou pelo menos amenizada. A descoberta de vulnerabilidades em sistemas computacionais tem ocorrido a uma velocidade alarmante, sendo que estas podem pertencer a classes variadas. Em resposta a esses novos tipos de ataques, foi estabelecido um novo paradigma para o desenvolvimento de ferramentas de segurança:

- Os sistemas de detecção de intrusão (IDS's). Esta é uma classe de ferramentas defensivas mais flexíveis, com o potencial necessário para identificar um amplo conjunto de ataques, e tomar as medidas cabíveis, de acordo com o que for pré-definido.

Foram testados alguns sistemas de NIDS, para se ter um comparativo, tanto de utilização / funcionamento, quanto de consumo de recursos. Os sistemas utilizados foram o Snort [SNOR] e o Firestorm. [FIRE]

2.1 HISTÓRICO TECNOLÓGICO

Este tópico apresenta alguns acontecimentos envolvendo a descrição e o relato de assinaturas de intrusão mais relevantes no decorrer das últimas décadas.

O CERT [CERT] é o maior órgão que reporta incidentes de segurança da *Internet*. Em dezembro de 1988, foi divulgado o primeiro boletim do CERT (CA-1988-01). Este tipo boletim, conhecido como *CERT Avisory*, descreve, identifica e explica como corrigir o problema.

Em fevereiro de 1998, Max Vision criou o Whitehats.com [WHIT], que tem a função de prover informações sobre vulnerabilidades de segurança. Em 1999 o foco principal do site

passou a ser um banco de dados de assinaturas de intrusão para uso em sistemas detectores de intrusão (arachNIDS). Entretanto essas assinaturas não têm um formato padronizado e bem especificado.

Outro fato importante na evolução das assinaturas de ataque foi o lançamento do Snort [SNOR] e, conseqüentemente, das suas regras em dezembro de 1998. Essas regras podem ser consideradas assinaturas, mas possuem informações limitadas sobre o ato intrusivo.

Por fim, em 1999 iniciou-se o funcionamento do CVE (*Common Vulnerabilities and Exposure*) [CVEX]. Este projeto tenta padronizar uma identificação única para cada vulnerabilidade.

Vale ainda lembrar que em 2001, foi divulgado o IDMEF (*Intrusion Detection Message Exchange Format*) [IETF]. Esta é uma proposta que visa padronizar o modo como as informações são trocadas entre partes de um sistema detector de intrusão.

2.2 ATACANTES DIGITAIS E SEUS TIPOS

Abaixo segue algumas descrições adotadas para pessoas que atuam na área de Intrusão (Invasor), assim como algumas possíveis motivações que as levam a invadir.

2.2.1 TIPOS DE ATACANTES

2.2.1.1 HACKER

Alguém interessado em sistemas operacionais, *softwares*, segurança e *internet* em geral. Também pode designar um programador, um indivíduo que desenvolve programas como forma de trabalho. Geralmente, suas intenções não são más, só querem mesmo superar desafios. Os grandes Hackers são anônimos, por isso não pense que aquele seu amigo que diz saber tudo de computadores é hacker. Um bom exemplo real foi quando o cracker Kevin Mitnick invadiu o computador do analista de sistemas Shimomura. Mitnick destruiu dados e roubou informações vitais. Shimomura é chamado de hacker, pois usa sua inteligência para o bem e possui muitos mais conhecimentos que seu inimigo digital.

2.2.1.2 CRACKER

Com más intenções, quebra ilegalmente a segurança de sistemas de computação, além de quebrar esquemas de registro de *software* comercial. Com um alto grau de conhecimento e nenhum respeito, invadem sistemas e podem apenas deixar a sua "marca" ou destruí-los completamente. Geralmente são hackers que querem se vingar de algum operador, adolescentes que querem ser aceitos por grupos de crackers (ou script kiddies) e saem apagando tudo que vêem, ou mestres da programação que são pagos por empresas para fazerem espionagem industrial. Geralmente, não têm princípios éticos como os hackers e só querem causar estragos aos ícones de seu maior inimigo: o Capitalismo. Hackers e crackers costumam entrar muito em conflito. Guerras entre grupos é comum, e isso pode ser visto em muitos fóruns de discussão e em grandes empresas, as quais contratam hackers para proteger seus sistemas.

2.2.1.3 LAMER OU LAMMER

Indivíduo que não tem domínio dos conhecimentos de um hacker, pouco experiente, com poucas noções de informática, porém tenta fazer-se passar por um hacker a fim de obter fama, o que acaba gerando antipatia por parte dos hackers verdadeiros. Um exemplo de Lammer é o Script Kiddie (Lammer que se utiliza de programas feitos por terceiros para tentar invadir sistemas, sem nem mesmo saber como funcionam estes programas). [WIKI]

2.2.1.4 WANNABE

Este é um estágio mais evoluído do Lammer. Ele aprendeu a usar alguns truques patenteados pelos hackers e acha que já pode invadir os computadores do Pentágono. Tudo bem, ele já invadiu o computador de um amigo ou um provedor de fundo de quintal, mas daí a dizer que ele vai entrar nos computadores do FBI...

2.2.1.5 WAREZ

É o fanático de *software* pirata. Distingue-se do pirata comum porque não ganha dinheiro com o seu trabalho. O termo tem principalmente a ver com cópia não autorizada de jogos de computador. É um cara fascinado pelos vírus de computador (talvez porque tenha tanto warez com vírus). E claro, costuma ter sempre a última versão oficial de vários anti-

vírus no micro (sem pagar, óbvio). A descoberta de warez não é passatempo. É obsessão. Tem que se ficar na frente do micro para se descobrir:

- Onde os *softwares* estão;
- Fazer o *download*;
- Divulgar para os amigos.

2.2.1.6 BANKERS

Indivíduos que fazem fraudes com contas bancárias via *Internet* são chamados de Bankers.

2.2.1.7 CARDERS

Indivíduos que fazem fraudes com cartões via *Internet* são chamados Carders.

2.2.1.8 NEWBIE

Expressão pejorativa referente aos novatos, principiantes. Começou sendo usada pelos hackers, mas atualmente, é usada por todos os segmentos da *Internet*. Aplica-se a todos aqueles que estão começando a aprender algo.

2.2.1.9 SCRIPT KIDDIE

O script kiddie nada mais é do que alguém procurando por um alvo fácil. Este alguém não procura por informações ou companhias específicas. O seu objetivo é obter acesso à conta do administrador de uma máquina (root) da maneira mais fácil possível. Assim, a técnica utilizada consiste em focalizar as ações em um pequeno número de falhas (exploits) e procurar pela *Internet* inteira, até que se consegue encontrar uma máquina que seja vulnerável, o que acontece mais cedo ou mais tarde. Alguns deles são usuários avançados, que desenvolvem suas próprias ferramentas e deixam para trás backdoors sofisticadas. Outros sabem apenas superficialmente o que estão fazendo e limitam-se a digitar "go" na linha de comando. Embora o nível técnico deles possa diferir, todos usam uma estratégia comum: procurar, alternadamente, por falhas específicas, para que, mais a frente, elas possam ser exploradas.

2.2.2 MOTIVAÇÃO

Como qualquer forma de crime, o invasor está atendendo as suas próprias necessidades, ou qualquer que seja a razão, e a motivação varia. Não há nenhum sentimento de dúvida associado com invadir e ganhar acesso completo a sistemas computacionais de terceiros. Segue abaixo algumas possíveis motivações:

- Desafio de invadir um sistema que parece impossível;
- Adquirir o máximo de conhecimento;
- Notoriedade ou o fator "elite";
- Malícia ou destruição;
- Diversão;
- Espionagem;
- Vingança;
- Investigação legal;
- Empregados descontentes;
- Ganho financeiro e furto.

2.3 INCIDENTES DE SEGURANÇA

Esquema de classificação que divide um bloco de conhecimento e define as relações entre as partes, também conhecido como taxonomia, é usado para classificar e entender as partes de um processo de ataque por exemplo. A relação desse esquema é importante para o entendimento geral de eventos que ocorrem num ambiente de rede.

Eventos, ataques e incidentes constituem conhecimentos e ações presentes na natureza de sistemas interligados em rede. Para o IDS proposto, a taxonomia básica, segue com detalhamento respectivo abaixo.

2.3.1 EVENTOS

Evento é uma ação direcionada a um destino com a finalidade de se alterar o status do mesmo. Alguns pontos compõem um evento, dentre eles como pode ser observado na *Figura 01* [CAM2001] estão a AÇÃO e o OBJETIVO.

Figura 01 – Esquema de Eventos em Redes e Computadores

AÇÃO – Pode ser caracterizada como um passo feito por um usuário ou um processo para alcançar um resultado. Como um probe, varredura, flood, autenticação, desvio, spoof, leitura, cópia, roubo, modificação e destruição.

OBJETIVO – Pode ser caracterizado como um computador ou uma entidade lógica de rede (conta, processo ou dado) ou entidade física (componente, computador, rede ou redes dentro de redes – Inter redes).

2.3.2 ATAQUES

Ataque é uma série de passos seguidos por um atacante para obter um acesso não autorizado. Esse atacante pode parametrizar passos, como mostra a *Figura 02* [CAM2001] e realizar um ataque.

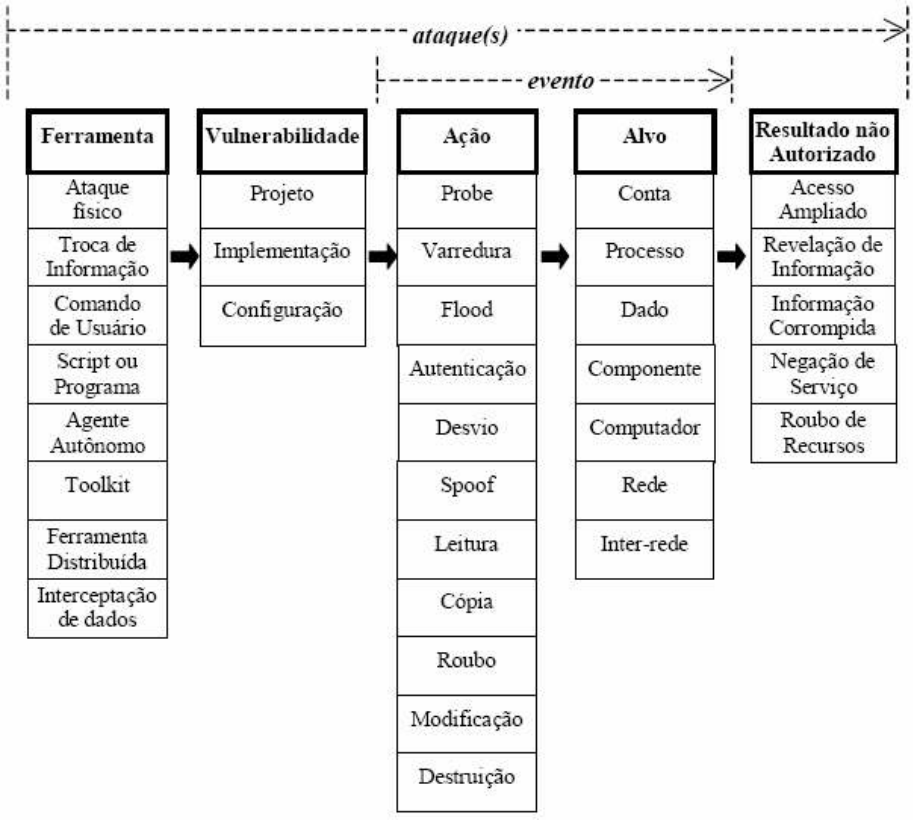

Figura 02 – Esquema de Ataques em Redes e Computadores

O domínio dessas técnicas, aliado ao conhecimento e experiência que esse atacante possui, pode garantir o acesso a um sistema que este não tem permissão para usar. A união de outras formas de ataque ou até mesmo as mais comuns podem traçar um perfil do comportamento e preferências de um ou mais atacantes. Será tratado aqui o perfil dos ataques mais comuns que os atacantes usam para localizar, identificar e invadir um sistema. Com tais informações, o administrador de rede poderá ser capaz de configurar um Firewall e/ ou IDS com regras mais elaboradas, evitando assim possíveis Falsos Negativos, que ocorrem quando o IDS considera uma tentativa de ataque como sendo um tráfego normal de rede e por esse motivo deixa de alertar o administrador do sistema quanto a ocorrência do evento ou Falsos Positivos, que ocorrem quando um tráfego normal de rede é identificado como um ataque, quando o IDS está configurado com regras muito rigorosas de maneira que pacotes normais que trafegam pela rede são detectados como ataques. Está sendo descrito primeiramente os ataques que envolvem a criação de conexões permitidas entre um cliente e SERVIDOR.

2.3.2.1 ATAQUE DE CANAL DE COMANDO

Um ataque de canal de comando é aquele que ataca diretamente o SERVIDOR de um determinado serviço enviando a ele comandos da mesma maneira que ele os recebe regularmente (sob seu canal de comando). Existem dois tipos básicos de ataques de canal de comando; os ataques que exploram comandos válidos para realizar ações indesejáveis, e ataques que enviam comandos inválidos e exploram bugs do SERVIDOR para lidar com entrada inválida. Se for possível usar comandos válidos para realizar ações indesejáveis, isso será culpa da pessoa que decidiu quais comandos devem existir. Se for possível usar comandos inválidos para realizar ações indesejáveis, isso será culpa dos programadores que implementaram o protocolo. Essas são duas questões separadas e precisam ser avaliadas separadamente, mas certamente haverá insegurança em ambos os casos. [ZWI2001]

2.3.2.2 ATAQUES DIRIGIDOS AOS DADOS

Um ataque dirigido aos dados é aquele que envolve os dados transferidos por um protocolo, em vez do SERVIDOR que os implementa. Mais uma vez, existem dois tipos de ataques dirigidos aos dados; os ataques que envolvem dados maliciosos e os ataques que comprometem dados de boa qualidade. Vírus transmitidos em mensagens de correio eletrônico são ataques dirigidos aos dados que envolvem dados maliciosos. Os ataques que roubam números de cartões de crédito em transito são ataques dirigidos aos dados que comprometem dados de boa qualidade. [ZWI2001]

2.3.2.3 ATAQUES DE TERCEIROS

Um ataque de terceiros é aquele que não envolve o serviço planejado admitir, mas utiliza as provisões criadas para oferecer suporte a um serviço com a finalidade de atacar um serviço completamente diferente. Por exemplo, se permitir conexões TCP de entrada em qualquer porta acima de 1024 para dar suporte a algum protocolo, estará sendo aberto um grande número de oportunidades para ataques de terceiro, à medida que as pessoas criarem conexões de entrada para servidores totalmente distintos. [ZWI2001]

2.3.2.4 FALSA AUTENTICAÇÃO DOS CLIENTES

Um risco importante nas conexões de entrada é a falsa autenticação: a subversão da autenticação que se exige de seus usuários, de forma que um atacante possa se fazer passar com sucesso por um dos seus usuários. Esse risco é aumentado por algumas propriedades especiais de senhas. Na maioria dos casos, se tiver um segredo e desejar repassá-lo dessa forma. Isso não ajuda se as informações não têm de ser compreendidas para serem usadas. Por exemplo, criptografar senhas não funcionará, pois um atacante que esteja usando sniffing de pacotes pode simplesmente interceptar e reenviar a senha codificada sem ter que decifrá-la (isso se chama ataque de reprodução porque o atacante grava uma interação e a reproduz mais tarde). Então, lidar com a autenticação através da *Internet* exige algo mais complexo que codificar senhas. É necessário um método de autenticação no qual os dados que passam através da rede não sejam reutilizáveis e assim um atacante não pode capturá-los e reproduzi-los. Também não é suficiente apenas se proteger contra ataques de reprodução. Um atacante que possa descobrir ou adivinhar a senha, não precisará usar um ataque de produção, e os sistemas que impedem reproduções não impedem necessariamente a adivinhação. Por exemplo, o sistema de desafio/resposta do WindowsNT é razoavelmente seguro contra ataques de reprodução, mas a senha realmente introduzida pelo usuário é a mesma todo o tempo. Além disso, se um atacante puder convencer um usuário de que ele é o seu SERVIDOR, o usuário entregará alegremente ao atacante seu nome de usuário e os dados de senha, que o atacante poderá então usar de imediato ou quando quiser. Para evitar isso, o cliente precisa se autenticar para o SERVIDOR, usando algum fragmento de informação que não passe pela conexão (por exemplo, codificando a conexão), ou o SERVIDOR precisa se autenticar para o cliente. Abaixo estão sendo apresentados os ataques que contornam a necessidade de se criar as conexões. [ZWI2001]

2.3.2.5 SEQÜESTRO

Os ataques de seqüestro permitem a um atacante assumir o controle de um terminal aberto ou uma sessão de *login* de um usuário que foi autenticado e autorizado pelo sistema. Em geral, os ataques de seqüestro ocorrem em um computador remoto, embora as vezes seja possível seqüestrar uma conexão de um computador na rota entre o computador remoto e seu computador local. Uma forma de proteger-se contra ataques de seqüestro no computador remoto é permitir conexões apenas a partir de computadores remotos que sejam de segurança;

no caso ideal, esses computadores devem ser pelo menos tão seguros quanto seu próprio computador. A esse tipo de restrição podem ser aplicados filtros usando filtros de pacotes ou servidores modificados. Os filtros de pacotes são mais fáceis de aplicar a uma coleção de sistemas, mas SERVIDORES modificados em sistemas individuais lhe oferecem maior flexibilidade. Por exemplo, um SERVIDOR FTP modificado poderia permitir FTP anônimo a partir de qualquer *host* da rede, mas FTP autenticado apenas a partir de *hosts* específicos. Este tipo de controle não pode obtido usando a filtragem de pacotes. No UNIX, o controle de conexão no nível de *hosts* está disponível a partir do TCP Wrapper da Wietse Venema ou de envoltórias no TIS FWTK (o programa netacl). Esses programas podem ser mais fáceis de configurar que filtros de pacotes, mas fornecem o mesmo nível de discriminação – somente por *hosts*. O seqüestro por sites intermediários pode ser evitado usando-se a proteção de integridade de um extremo a outro. Quando usada a proteção de integridade de um extremo a outro, os sites intermediários não poderão inserir pacotes autênticos no fluxo de dados porque eles não conhecem a chave apropriada e os pacotes serão rejeitados e então não poderão seqüestrar sessões através deles. O padrão IPsec da IETF (*Internet Enginnering Task Force*) fornece esse tipo de proteção na camada IP sob o nome "Authentication Headers" (cabeçalhos de autenticação), ou protocolo AH [RFC2402]. A proteção contra seqüestros na camada de aplicação, juntamente com a proteção de privacidade, pode ser obtida adicionando-se um protocolo de segurança ao aplicativo; as escolhas mais comuns para isso são a TLS (*Transport Layer Security*) ou a SSL (*Secure Socket Layer*), mas também existem aplicativos que usam a GSSAPI (*Generic Security Services Application Programming Interface*). No caso do acesso remoto a sistemas UNIX, o uso do SSH (*Secure Shell*) pode eliminar o risco de seqüestro de sessões baseado na rede. O seqüestro no computador remoto é bastante direto, e o risco é grande se as pessoas deixam as conexões abandonadas. O seqüestro a partir de sites intermediários é um ataque bastante técnico e só tem probabilidade de ocorrer se existe alguma razão para seu site ser o alvo específico de alguma pessoa. O seqüestro pode ser considerado um risco aceitável para a sua própria empresa, particularmente se for capaz de minimizar o número de contas que têm acesso total e o tempo que elas despendem conectadas remotamente. Porém, é provável que não deseje permitir que centenas de pessoas efetuem logon a partir de qualquer lugar na *Internet*. De modo semelhante, pode ser permitido que os usuários se conectem de forma consistente, a partir de sites remotos específicos sem tomar precauções especiais, nem deseja que os usuários efetuem logon em contas ou máquinas particularmente seguras e a partir da *Internet*. O risco de seqüestro pode ser reduzido por meio de uma política de sessão ociosa com imposição de tempos limites rígidos. Além disso, é útil

ter controle de auditoria sobre o acesso remoto, de forma que tenha alguma esperança de notar se uma conexão está sendo seqüestrada. [ZWI2001]

2.3.2.6 SNIFFING DE PACOTES

Talvez os atacantes não precisem seqüestrar uma conexão a fim de obterem informações que se deseja manter secretas. Simplesmente observando a passagem de pacotes – em qualquer lugar entre o site remoto e o seu site – eles podem ver quaisquer informações não codificadas que estejam sendo transferidas. Os programas de sniffing de pacotes automatizam essa observação de pacotes. Os sniffers podem seguir senhas ou dados. Há diferentes riscos associados com cada tipo de ataque. Proteger seus dados contra sniffers é mais difícil, pois os dados precisam ser codificados antes de passarem pela rede. Há dois meios utilizados para este tipo de criptografia:

- Codificar arquivos que serão transferidos;
- Codificar links de comunicação.

A codificação de arquivos é interessante quando se está utilizando protocolos que transferem arquivos inteiros; quando há um modo seguro de inserir as informações que serão usadas para codificá-los; e quando se tem um modo seguro de fazer chegar ao destinatário as informações necessárias para decodificá-las. Ela é particularmente útil quando o arquivo deve cruzar vários links de comunicação e é certo de que todos eles estarão protegidos, ou se o arquivo despenderá tempo em *hosts* nos quais não sejam de confiança. Por exemplo, se estiver escrevendo mensagens confidenciais em um laptop e usando um sistema de criptografia de chave pública, poderá ser feita toda codificação da máquina que controla e encaminhar o arquivo codificado inteiro em segurança, ainda que ele passe através de vários servidores de correio e links de comunicação desconhecidos. Em muitas situações, em vez de codificar os dados com antecedência, é mais prático codificar a conversação inteira. Poderá se optar por codificar no nível do IP, através de uma solução de rede privada virtual (*VPN – Virtual Private Network*), ou então escolher um protocolo criptografado (por exemplo, o SSH para acesso remoto ao shell). Atualmente, a espionagem e a criptografia estão ambas difundidas. A codificação deve ser deixada em serviços de entrada, a menos que tenha algum modo para se certificar de que não haverá dados confidenciais passando através deles. Talvez também queira codificar as conexões de saída, em particular, se tem qualquer razão para acreditar que as informações que elas contêm são confidenciais. [ZWI2001]

2.3.2.7 INJEÇÃO E MODIFICAÇÃO DE DADOS

Um atacante que não possa assumir o controle bem sucedido de uma conexão poderá ser capaz de alterar os dados no interior da conexão. Um atacante que controla um roteador entre um cliente e um SERVIDOR pode interceptar um pacote e modificá-lo, em vez de simplesmente ler o pacote. Em raros casos, até mesmo um atacante que não controla um roteador pode conseguir isso enviando o pacote modificado de tal forma que ele chegue antes do pacote original. A criptografia de dados não o protegerá contra esse tipo de ataque. Um atacante ainda poderá modificar os dados codificados. A criptografia impedirá o atacante de transformar intencionalmente um pedido de 200 pingüins de geladeira em um pedido de 2000 pingüins de geladeira, mas não impedirá que o atacante transforme o pedido em lixo que destruirá seu sistema de entrada de pedidos. Não podendo sequer dar a certeza de que o atacante não transformará por acaso em algo com outro significado. A proteção total de serviços contra modificação exige alguma forma de proteção de integridade de mensagens, na qual o pacote inclui o valor de um total de verificação que é calculado a partir dos dados e não pode ser recalculado por um atacante. [ZWI2001]

2.3.2.8 RETRANSMISSÃO

Um atacante que possa assumir o controle de uma conexão ou alterar a conexão ainda poderá ser capaz de provocar danos simplesmente gravando informações que passaram e enviando-as novamente. Existem dois tipos de transmissão, as capazes de identificar certos fragmentos de informação (por exemplo, o ataque de senhas), e aqueles que simplesmente enviam novamente o pacote inteiro. Muitas formas de codificação o protegerão contra ataques em que o atacante está obtendo informações para retransmissão, mas elas não o ajudarão se for possível apenas reutilizar um pacote sem saber o que ele contém. A retransmissão de pacotes não funciona com o TPC por causa dos números de seqüestro, mas não há nenhuma razão para ela falhar com protocolos baseados no UDP. A única proteção contra ela é ter um protocolo que rejeite o pacote retransmitido (por exemplo, utilizando timbres de hora ou números de seqüência incorporados por algum tipo). O protocolo também deve realizar alguma forma de verificação de integridade de mensagens, a fim de impedir que o atacante atualize o pacote interceptado. [ZWI2001]

2.3.2.9 NEGAÇÃO DE SERVIÇO

Um ataque de negação de serviço é aquele em que o atacante não esta tentando conseguir acesso a informações, mas apenas tentando impedir que outras pessoas tenham acesso a elas. Os ataques de negação de serviço podem assumir várias formas, e é impossível impedir todos eles. [ZWI2001]

Embora o ataque de negação de serviço não possa ser evitado, é possível torná-lo muito mais difícil de implementar. Primeiro, os SERVIDORES não devem ficar indisponíveis quando forem emitidos comandos inválidos. SERVIDORES mal implementados podem cair ou entrar em loop em resposta a entrada hostil, o que simplifica bastante a tarefa do atacante. Em segundo lugar, os servidores devem limitar os recursos alocados a qualquer entidade isolada. Isso inclui:

- Números de conexões abertas ou solicitações pendentes;
- Tempo decorrido desde que uma conexão foi estabelecida ou enquanto uma solicitação esta sendo processada;
- Tempo do processador gasto em uma conexão ou uma solicitação;
- Quantidade de memória alocada a uma conexão ou uma solicitação;
- Quantidade de espaço em disco alocado a uma conexão ou uma solicitação.

2.3.3 INCIDENTE

Incidente é um grupo de ataques que pode ser distinguido de outros ataques por conta das peculiaridades dos atacantes, como formas de ataque, objetivos, sites, tempos. Isso pode ser observado na *Figura 03* [CAM2001], onde estão relacionados grupos humanos com ideologias e comportamentos específicos, entretanto, desejam alcançar objetivos diferentes utilizando-se de meios comuns.

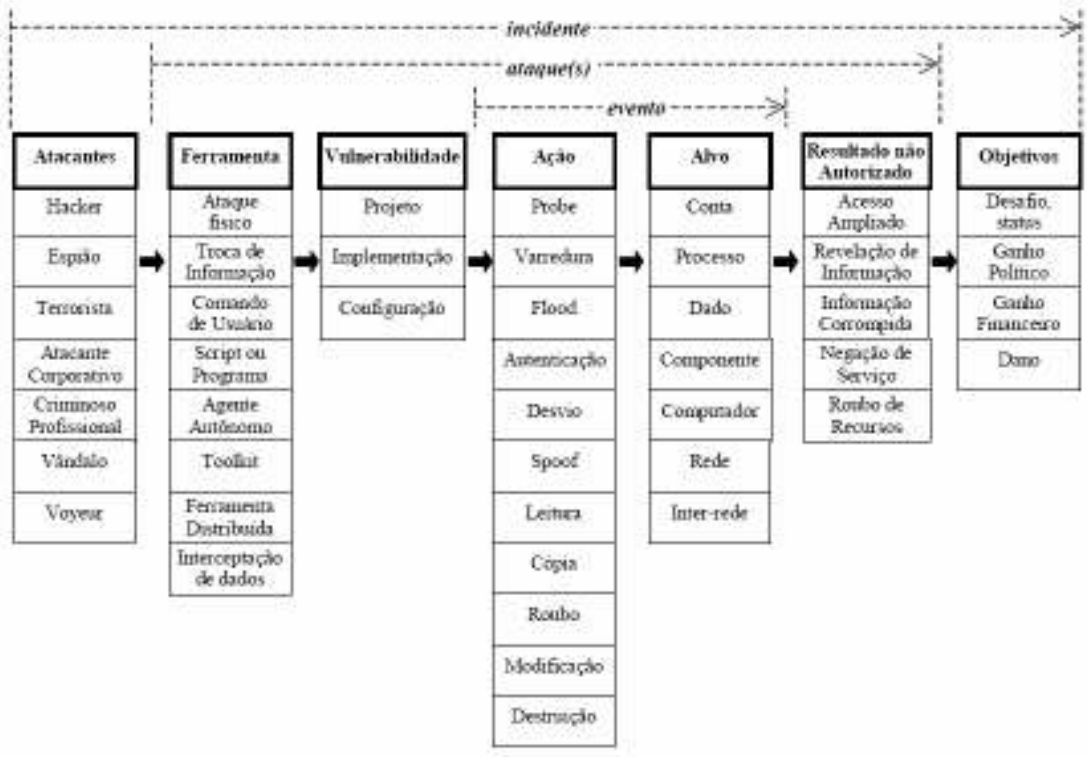

Figura 03 – Esquema de Incidentes de Taxonomia em Redes e Computadores

2.4 PERFIL DE ATAQUES

Nota-se que para um ataque bem sucedido, se faz necessário uma série de pré-requisitos, como por exemplo, levantar informações da vítima, possíveis vulnerabilidades, possíveis proteções, qual a melhor ferramenta e/ou exploit a ser utilizada, etc. Baseado nisso, está descrito abaixo alguns dos perfis que podem ser utilizados para se ter um ataque com sucesso.

2.4.1 RECONHECIMENTO

Técnicas utilizadas para conhecer melhor a vítima, podendo assim ter conseqüentemente uma maior possibilidade de sucesso na tentativa de invasão.

2.4.1.1 ENGENHARIA SOCIAL

Este mecanismo de recolhimento de informações é uma das formas mais perigosas e eficientes utilizada pelos Invasores.

Um bom exemplo de ataque de engenharia social é o de ligar para um setor de informática de uma corporação, dizendo ser um novo funcionário de um determinado setor e dizer que precisa de um *login* e senha para acesso ao sistema. Muitas vezes o Invasor consegue através deste telefonema o *login* e a senha necessários para o início de seu ataque.

Uma forma mais fácil ainda é de ligar para o setor de informática dizendo ser o fulano de tal que esqueceu a senha e gostaria que a senha fosse trocada. Claro que desta forma o Invasor tem que conhecer um nome que possua conta no sistema e esteja muito tempo sem utilizá-la.

Variando muito de organização para organização, a obtenção de informações através de engenharia social ainda é utilizada com muito sucesso em diversas organizações e seu sucesso depende exclusivamente do conhecimento do pessoal em assuntos de redes e computadores. A melhor defesa contra este ataque é o treinamento dos funcionários e usuários de redes e computadores.

2.4.1.2 VARREDURAS DE PORTAS E SOFTWARES DE ANÁLISE DE REDES

A primeira coisa que um atacante normalmente faz em uma rede é a chamada Varredura de Portas. Esta varredura consiste em enviar pacotes para todas as portas de uma máquina, ou de várias máquinas de uma rede, de modo a descobrir quais são os serviços oferecidos por cada uma delas.

Com o uso desta técnica, é possível se determinar exatamente quais serviços TCP e UDP estão sendo oferecidos.

Este tipo de ataque ficou muito popular quando em 1995 foi lançado, com grande publicidade, o *software SATAN*, desenvolvido por Dan Farmer e Wietse Wenema. Feito em Perl, C e HTML (*HyperText Markup Language*), com uma interface amigável, o SATAN faz uma varredura completa em uma máquina ou rede denominada "alvo", relatando todos os serviços oferecidos e pontos vulneráveis. Ele não só descobre quais serviços estão disponíveis como também utiliza estes serviços para descobrir mais informações sobre o alvo e mais pontos vulneráveis. Existem atualmente inúmeros programas com funcionalidade similar ao SATAN, muitos deles em domínio público.

Uma ferramenta mais atual e que permite explorar muitas falhas e vulnerabilidade em sistemas é o NESSUS (*Security Scanner*).

Para redes que estejam protegidas por filtro de pacotes a técnica usada para se descobrir todos os serviços TCP que estão rodando em uma rede é a chamada varredura

invisível (*Stealth Scanning*), logicamente não será possível se conectar aos serviços utilizando esta técnica, mas ela já é um primeiro passo para um ataque.

O ataque de varredura invisível consiste em fazer uma varredura de portas, como já citado, porém com a diferença de não se enviar pacotes de abertura de conexão e sim pacotes simulando uma conexão existente. Como os filtros de pacotes normalmente permitem o trânsito dos pacotes que não são abertura de conexão, todos estes pacotes atingiriam a máquina destino. A máquina destino ao receber os pacotes reagiria de maneira diferenciada caso existisse um serviço rodando na porta especificada ou não.

O protocolo TCP utiliza um esquema de três pacotes para estabelecer uma conexão:

- A máquina cliente envia um pacote para a máquina servidora com um flag especial, chamado de flag de SYN. Este flag indica que a máquina cliente deseja estabelecer uma conexão.
- A máquina servidora responde com um pacote contendo os flags de SYN e ACK. Isto significa que ela aceitou o pedido de conexão e está aguardando uma confirmação da máquina cliente para marcar a conexão como estabelecida.
- A máquina cliente, ao receber o pacote com SYN e ACK, responde com um pacote contendo apenas o flag de ACK. Isto indica para a máquina servidora que a conexão foi estabelecida com sucesso.

Todos os pedidos de estabelecimento de conexões recebidas por um SERVIDOR ficam armazenados em uma fila especial, que tem um tamanho pré-determinado e dependente do sistema operacional, até que o SERVIDOR receba a comunicação da máquina cliente de que a conexão está estabelecida. Caso o SERVIDOR receba um pedido de conexão e a fila de conexões em andamento estiver cheia, este novo pacote de pedido de abertura de conexão é descartado.

O ataque consiste basicamente em se enviar um grande número de pacotes de abertura de conexão, com um endereço de origem forjado, para um determinado SERVIDOR. Este endereço de origem é forjado para o de uma máquina inexistente (muitas vezes se usa um dos endereços reservados da *Internet*). O SERVIDOR, ao receber estes pacotes, coloca uma entrada na fila de conexões em andamento, envia um pacote de resposta e fica aguardando uma confirmação da máquina cliente. Como o endereço de origem dos pacotes é falso, esta confirmação nunca chega ao SERVIDOR. O que acontece é que em um determinado momento a fila de conexões em andamento do SERVIDOR fica lotada. A partir daí, todos os pedidos de abertura de conexão são descartados e o serviço inutilizado. Esta inutilização persiste durante alguns segundos, pois o SERVIDOR ao descobrir que a confirmação está

demorando demais remove a conexão em andamento da lista. Entretanto, se o atacante persistir em mandar pacotes seguidamente, o serviço ficará inutilizado enquanto ele assim o fizer. O Firewall Aker possui um módulo especial de proteção contra ataques de SYN flood, oferecendo total proteção contra ataques deste tipo.

A varredura de endereços é um primeiro sinal comum de ataque. Os Invasores usam freqüentemente a varredura de endereços e as técnicas relacionadas para descobrir *hosts*. Felizmente, a varredura de endereços é fácil de filtrar.

2.4.1.3 VARREDURA LENTA

Os detectores de varreduras de portas podem detectar uma taxa alta de conexões similares em diversas portas. Uma possível solução para os Invasores seria diminuir a taxa de varredura.

Este ataque é uma modificação da técnica de varredura de portas. Sua utilidade é duvidosa, considerando o tempo que leva. Os métodos de varredura lenta contam com o fato de que os Firewalls e os detectores de varreduras esperam encontrar altas taxas de conexões oriundas de um único endereço para determinar se está ou não ocorrendo uma varredura. Os detectores de varredura detectam essas taxas altas acompanhando o número de conexões que um *host* específico solicita em certo tempo (10 por segundo, por exemplo). Ao fazer varreduras mais lentas do que essa taxa de detecção, uma tentativa por segundo por exemplo, os Invasores evitam a detecção.

Infelizmente para os Invasores a varredura lenta demora muito, portanto esses ataques são usados somente quando sistemas específicos são atacados com um objetivo. Quanto a se defender desse tipo de ataque, as varreduras lentas são muito difíceis de detectar.

2.4.1.4 SONDAS DE ARQUITETURA

Os atacantes transmitem pacotes alterados para os *hosts* com o intuito de obter alguma resposta, caso a máquina alvo esteja operacional. Examinando as respostas, eles são capazes de determinar o sistema operacional que está sendo executado na máquina-alvo, pois para cada tipo de Sistema Operacional a resposta é diferenciada.

É examinada a resposta às transmissões de pacotes inválidos oriundas de um *host-alvo* usando uma ferramenta automatizada que contém um banco de dados de tipos de respostas conhecidos. Como não existe uma definição de resposta típica, cada sistema

operacional responde de maneira própria. Comparando essas respostas com respostas conhecidas de um banco de dados, os Invasores podem determinar com freqüência que Sistema Operacional está sendo executado no *host-alvo*.

Quanto a se defender desse tipo de ataque, considere que os Invasores conseguem determinar os sistemas operacionais dos *hosts* públicos. Com isso em mente, planeje suas defesas de modo que elas não dependam dessa informação. Por exemplo, não se deve considerar que um Invasor não possa saber que o Windows NT Server está sendo executado em uma máquina porque as portas identificadoras foram bloqueadas. Todas as medidas de segurança possíveis devem ser tomadas para proteger um sistema operacional mesmo que os Invasores não tenham como saber qual é o sistema operacional.

2.4.1.5 E-MAIL FORJADO

Os atacantes podem criar mensagens de e-mail que pareçam ter vindo de qualquer um e que peçam uma resposta. Em uma variação do ataque, eles também podem falsificar o endereço de resposta, tornando a falsificação não detectável.

Usando técnicas tão simples quanto configurar um cliente de e-mail com informações incorretas, os atacantes podem forjar e-mails para clientes internos de uma rede. Alegando ser de alguém que o cliente conhece e confia, esse e-mail usa um ataque psicológico para induzir o leitor a retornar informações úteis ou então inclui um cavalo de Tróia instalável ou ainda um link para um site Web malicioso.

O SMTP (*Simple Mail Transfer Protocol*) não autentica a identidade de um remetente de e-mail, e muitas versões de programas de e-mail não registram informações suficientes para rastrear adequadamente a origem de uma mensagem de e-mail. Os servidores de e-mail normalmente incluem uma seqüência de cabeçalhos de transmissão em suas mensagens de e-mail, mas diversos servidores de e-mail são conhecidos por retirarem esses cabeçalhos. Como os atacantes sabem que servidores retiram os cabeçalhos dos e-mails, eles os usam para tornar anônimos os seus ataques. A “lavagem” de endereços IP por meio de um SERVIDOR proxy mal configurado também pode fazer com que seja impossível rastrear os e-mails.

Quanto a se defender desse tipo de exposição, até pouco tempo atrás a única defesa real contra a falsificação de e-mails era saber que ela existia, mas atualmente pode ser implementado o "POP before SMTP". [4] [SMTP]

Certifique-se de que os usuários da rede entendam que a falsificação de e-mail é possível e que ela constitui um mecanismo de ataque plausível em redes bem defendidas. Também podem ser usados clientes de e-mail com S/MIME *(Secure/Multipurpose Internet Mail Extensions)* habilitado e instalar certificados pessoais de criptografia para assinar e-mails oriundos de todos os usuários internos. Assim, qualquer e-mail não assinado pode ser considerado um suspeito em potencial. Filtrando os anexos executáveis de todos os e-mails no Firewall.

2.4.1.6 PASSIVE SNIFFING

Este tipo de ataque vem se tornando freqüente na *Internet*. Este ataque é geralmente o primeiro passo para ataques como hijacking attack e IP spoofing juntamente com SYN flooding.

Para iniciar um ataque de sniffing, o atacante primeiramente necessita obter um username e senha de um usuário válido no sistema, para que ele possa se logar na rede. Depois de entrar na rede, o atacante instala um sniffer e começa a ter acesso ao conteúdo dos pacotes transmitidos na rede, e quando houver interesse copia estes pacotes, assim ele vai obtendo informações valiosas da rede e de seus usuários. [STR1999]

Quando duas máquinas quaisquer estão se comunicando, todo o tráfego entre elas passa em claro de máquina em máquina, da origem até o destino. Na quase totalidade das vezes, a administração destas máquinas intermediárias é feita por terceiros e nada se pode afirmar quanto a sua honestidade, na maioria das vezes, não é nem possível se saber de antemão por quais máquinas os pacotes passarão até atingir o destino.

O problema que isso causa, é que todo o tráfego entre as duas máquinas que estão se comunicando pode ser visualizado ou alterado por qualquer uma das máquinas intermediárias. Este problema se torna ainda mais crítico quando se pretende transmitir dados confidenciais ou críticos para a operação das duas entidades se comunicando.

A finalidade original dos sniffers era localizar problemas na rede e nos protocolos, mas está sendo mais utilizado para capturar senhas e *logins*.

[4] POP before SMTP, ou seja, autententicar antes de enviar novas mensagens.

Entre os serviços, o telnet e o FTP são os mais atingidos por este ataque, pois tornam-se serviços extremamente vulneráveis a este ataque.

Telnet (porta 23)

Os programas telnet e telnetd provêm serviço de terminal virtual remoto, ou seja, através deste serviço o usuário entra em uma máquina remota, através da rede, e trabalha nela como se estivesse sentado num terminal, diretamente conectado a ela. O telnet realmente imita um terminal, não uma estação gráfica, ele provê acesso a somente aplicações baseadas em texto. Através do telnet qualquer máquina conectada a *Internet* pode ter acesso a um sistema, desde que devidamente identificado. O grande problema deste serviço é que para a identificação do usuário na máquina remota, trafegam pela rede o username e a senha em claro, sem qualquer método de criptografia. Ficando assim muito suscetível a um monitoramento. Por este motivo, o telnet é atualmente considerado um dos serviços mais perigosos quando acessa remotamente um sistema, podendo comprometê-lo. O telnet só é seguro quando a máquina remota e todas as redes entre ela e a máquina local são seguras, ou seja, não é seguro um telnet na *Internet*, quando não se sabe por que máquinas passam os pacotes e muito menos se pode confiar nelas. Por outro lado, o telnet é um serviço extremamente útil, e pode ser usado mesmo com conexões de baixa velocidade.

O telnet por si só não é inseguro, se o protocolo IP implementasse mecanismos de privacidade, onde os dados trafegassem cifrados através da rede, o telnet seria um serviço seguro.

FTP (File Transfer Protocol) - portas 20 e 21

O serviço FTP permite que usuários transfiram arquivos facilmente de um sistema para outro através da rede. Juntamente com o e-mail e WWW ele é um dos serviços mais utilizado na *Internet*.

O FTP apresenta o mesmo problema de segurança do telnet, trafegar username e senha pela rede sem nenhuma proteção. [STR1999]

2.4.1.7 ACTIVE SNIFFING

O Sniffing ativo é muito semelhante ao passivo, diferindo somente que nesta técnica o invasor, além de "olhar" os pacotes, pode também alterá-los. Com esta característica, esta técnica torna-se muito perigosa e através dela é possível partirem diversos ataques. Para a utilização desta técnica é necessário um sniffer e um programa gerador de pacotes. Se este

conjunto de *softwares* estiver bem localizado na rede, um Invasor torna todas as conexões TCP vulneráveis. [STR1999]

2.4.1.8 PHISHING SCAN

Pescaria, ou "fishing" em inglês, é o ato de enviar um e-mail a alguém, alegando falsamente ser uma entidade como uma empresa ou organização, tentando fazer com que a vítima entregue informações pessoais objetivando o roubo de identidade, senhas, números de cartões de crédito, de contas de banco...

Nesse caso, freqüentemente o golpista envia mensagem com o mesmo visual que é usado por uma empresa como isca (bancos, lojas de e-commerce...), e tenta fazer com que o internauta revele informações financeiras e pessoais.

Aparentemente, em resposta a isso, os internautas estão modificando seus hábitos na web. Assim, considerando os ataques online, é importante excluir mensagens suspeitas (e-mails de pessoas desconhecidas), sem sequer abrí-las.

2.4.2 SPOOFING

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

2.4.2.1 TRANSFERÊNCIA DE ZONA DNS

Os atacantes podem transferir informações sobre nomes de seu SERVIDOR de DNS (*Domain Name System*) para identificar os *hosts* internos. O protocolo DNS não faz autenticação de transferências ou atualizações informativas. Isso torna o protocolo explorável de diversas maneiras. Os atacantes podem fazer uma transferência de zona para obter os nomes e os endereços IP internos de todos os *hosts* em uma única operação se houver um SERVIDOR DNS público. Quanto a se defender desse tipo de ataque, é recomendado que organizações menores não devem executar servidores de DNS próprios. Use Firewalls que suportem a divisão de DNS para garantir que os nomes e os endereços internos permaneçam

privados. Filtre as solicitações de transferência de zona no próprio Bind (Daemon SERVIDOR de DNS).

2.4.2.2 POLUIÇÃO DO CACHE DE DNS

Os atacantes podem fornecer atualizações falsas a servidores de DNS com endereços IP incorretos. Como os servidores de DNS não fazem autenticação quando trocam informações com outros servidores de nomes, podem inserir informações erradas com a intenção de desviar os usuários para os *hosts* do próprio Invasor.

Quanto a se defender desse tipo de exposição, é recomendado filtrar as atualizações de DNS dirigidas para dentro do sistema no Firewall. Nenhum SERVIDOR de nomes externo deve atualizar as informações sobre as máquinas internas no SERVIDOR interno.

2.4.2.3 ARP POISON OU ARP SPOOF OU ARP CACHE

Uma das operações básicas do protocolo Ethernet refere-se a geração de requisições (request) e respostas (reply) ARP (Address Resolution Protocol). Em geral, quando um nó A deseja se comunicar com um nó C, ele envia uma requisição ARP. O nó C irá enviar uma resposta ARP que irá incluir o endereço MAC. Mesmo em um ambiente chaveado como na *Figura 04*, a requisição ARP inicial é enviada via *broadcast*.

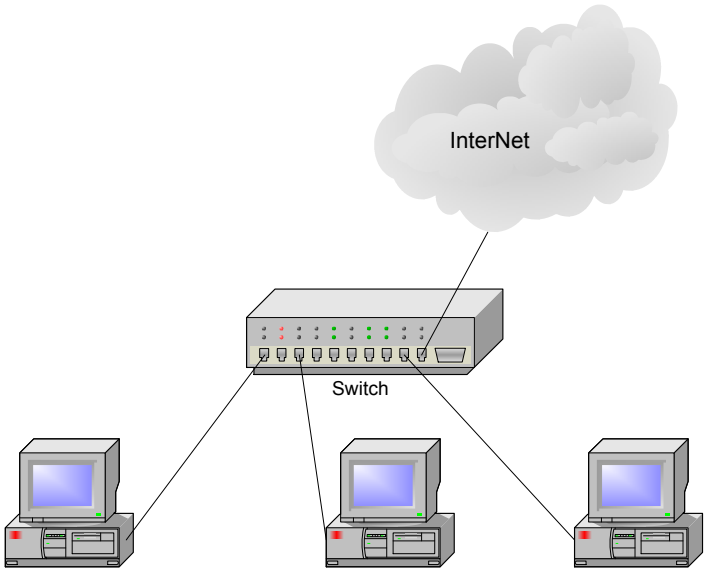

Figura 04 – Esquema de Topologia de Rede Segmentada utilizando Switch

É possível para um nó B compor e enviar uma resposta ARP não solicitada ao nó A. Esta falsa resposta ARP irá especificar que o nó B tem o endereço MAC do nó C. Com isso, o nó A irá inconscientemente enviar o tráfego ao nó B, já que este declara que possui o endereço MAC desejado. Algumas ferramentas disponíveis são especializadas no envio de falsos pacotes ARP a classes de máquinas (por exemplo, servidores NFS, HTTP, etc). Um exemplo deste tipo de ferramenta é o dsniff, recomendada para capturar tipos de tráfego específicos. Outras ferramentas concentram-se nos pedidos ARP em geral e enviam respostas falsas logo em seguida. Para que esse tipo de ataque funcione, é necessária a habilidade de enviar (forward) frames recebidos aos seus *hosts* origem. Isso é possível através de algum tipo de IP forwarding em nível de kernel ou aplicação. Esta técnica consiste em enviar pacotes ARP forjados para as máquinas-alvo, de forma que todo o tráfego entre elas passe pela máquina do invasor, conforme *Figura 05*.

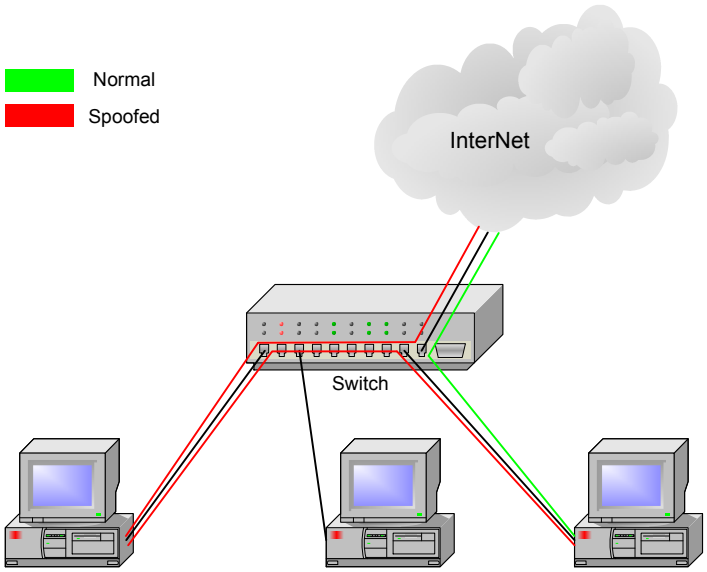

Figura 05 – Esquema de Arp Poison para captura de pacote

2.4.2.4 IP SPOOFING

Este ataque consiste em "mentir" o número IP da máquina, geralmente trocando-o por um número IP qualquer, isto pode ser feito através de manipulação direta dos campos do cabeçalho. Quando um *host* A quer se conectar ao B, a identificação é feita através do número IP que vai no cabeçalho, por isto, se o IP do cabeçalho enviado pelo *host* A for falso (IP de um *host* C), o *host* B, por falta de outra forma de identificação acredita estar se comunicando com o *host* C. O IP Spoofing não é exatamente uma forma de ataque, mas sim uma técnica

que é utilizada na grande maioria dos ataques, pois ele ajuda a esconder a identidade do atacante. Através da técnica de IP falso, o Invasor consegue atingir os seguintes objetivos:

- Obtém acesso a máquinas que confiam no IP que foi falsificado.
- Capturar conexões já existentes (para isto é necessário utilizar-se também de outras técnicas como predizer o número de seqüência).
- Burlar os filtros de pacotes dos firewalls que bloqueiam o tráfego baseado nos endereços de origem e destino.

A técnica de IP Spoofing é também utilizada juntamente com um ataque de negação de serviço, o qual consiste em disparar algum processo que sobrecarregue a máquina ou algo que ela não consiga finalizar. Um bom exemplo de ataque, que junta IP Spoofing com ataque de negação de serviço é o SYN Flood, um dos mais populares ataques DoS. Esse tipo de ataque tenta desabilitar um equipamento ou até mesmo a rede inteira. As chances deste tipo de ataque podem ser reduzidas com arquiteturas de rede segura, com filtragem de pacotes e utilização de classes privadas na rede interna. O perigo maior deste ataque é para serviços baseados no protocolo UDP. Muitos serviços baseados neste protocolo só permitem acesso para determinadas máquinas, explicitamente configurado pelo administrador. Este é o caso do serviço NFS, que permite o compartilhamento de discos remotos. O serviço NFS é só um exemplo, pois este ataque é válido para qualquer serviço similar. Um atacante pode enviar um pacote UDP para a máquina servidora de NFS como se este viesse de uma máquina cliente, autorizada a gravar em algum diretório. Este pacote pode conter solicitação de criação, remoção e escrita de arquivos. Como a máquina servidora acha que o pacote vem da máquina cliente, por causa do IP de origem falsificado, todas as solicitações serão prontamente atendidas. Este tipo de ataque não pode ser bloqueado sem a existência de um filtro de pacotes. Logicamente, por ter o IP falso, a máquina do atacante nunca recebe os pacotes de retorno, pois o *host* atacado retorna os pacotes para o verdadeiro *host*. Em alguns ataques, isto pode complicar um pouco a vida do atacante, como no caso do sequence number, mas em suma para muitos ataques isto não chega a ser uma limitação, pois em muitos deles o atacante não precisa receber os pacotes de retorno. Um bom exemplo disso seria o ataque explorando o rlogin para deixar um backdoor no sistema a fim de facilitar acessos futuros. [STR1999]

2.4.3 NEGAÇÃO DE SERVIÇO

2.4.3.1 DENIAL OF SERVICE

Os ataques do tipo "negação de serviço" consistem em impedir o funcionamento de uma máquina ou de um serviço específico. No caso de ataques a redes, geralmente ocorre que os usuários legítimos de uma rede não consigam mais acessá-la. No caso do SYN Flood, consegue-se inutilizar quaisquer serviços baseados no protocolo TCP. DoS não é um ataque propriamente dito, é um tipo de ataque, o qual inclui ataques como sobrecarga da rede, excessivos pedidos de abertura de conexão (*SYN Flooding*) etc.

Os antigos mainframes tinham defesas contra estes ataques. Os sistemas atuais são excessivamente pobres em relação a estes ataques. A situação vem piorando com as novas linguagens e ambientes de programação, nas quais é possível alocar recursos sem maiores limitações. Java e Javascript são dois bons exemplos, elas podem manipular com bastante liberdade diversos recursos do sistema, possibilitando assim diversos Denial of Service. Define-se também como sobrecarga de serviço quando uma enchente de requisições de rede é feita a um daemon num único computador. Estas requisições podem ser iniciadas de diversas formas, a maioria delas intencionais. Esta enchente causa a interrupção do serviço. Os ataques de negação de serviço vêm crescendo muito atualmente, e muitas vezes eles são usados em conjunto em outros ataques. [STR1999]

2.4.3.2 SYN FLOOD

SYN Flood é um dos mais populares ataques de negação de serviço (DoS). Esses ataques visam impedir o funcionamento de uma máquina ou de um serviço específico. No caso do SYN Flood, consegue-se inutilizar quaisquer serviços baseados no protocolo TCP. Para se entender este ataque, é necessário primeiro se entender o funcionamento do protocolo TCP, no que diz respeito ao estabelecimento de conexões, o esquema tree way handshake (ou handshake de três vias). O ataque consiste basicamente em se enviar um grande número de pacotes de abertura de conexão, com um endereço de origem forjado, para um determinado SERVIDOR. Este endereço de origem é forjado para o de uma máquina inexistente (muitas vezes se usa um dos endereços reservados). O SERVIDOR, ao receber estes pacotes, coloca uma entrada na fila de conexões em andamento, envia um pacote de resposta e fica

aguardando uma confirmação da máquina cliente. Como o endereço de origem dos pacotes é falso, esta confirmação nunca chega ao SERVIDOR. O que acontece é que em um determinado momento, a fila de conexões em andamento do SERVIDOR fica lotada. A partir daí, todos os pedidos de abertura de conexão são descartados e o serviço inutilizado. Esta inutilização persiste durante alguns segundos, pois o SERVIDOR ao descobrir que a confirmação está demorando demais, remove a conexão em andamento da lista. Entretanto, se o atacante persistir em mandar pacotes seguidamente, o serviço ficará inutilizado enquanto ele assim o fizer. O SYN Flooding é não apenas uma forma de ataque que pode ser utilizada separadamente, mas também uma técnica utilizada em ataques mais elaborados, onde o atacante precisa "parar" uma máquina para que ela não atrapalhe o seu ataque. [STR1999]

2.4.3.3 LAND

O ataque Land é uma variação da inundação SYN que pode fazer com que as implementações de TCP/IP não – protegidas “corram atraz de seu próprio rabo“ em um loop de estabelecimento de conexão que nunca termina. Este consiste em mandar para um *host* um pacote IP com endereço de origem e destino iguais, o que ocasiona um loop. Para executar um ataque como este, basta que o invasor tenha um *software* que permita a manipulação dos campos dos pacotes IP e configurá-lo. No ataque Land, um pacote SYN especialmente montado é transmitido para um *host* SERVIDOR. Tanto, o endereço de origem quanto o destino do pacote SYN são definidos como endereço IP do SERVIDOR. Isso faz com que o SERVIDOR receptor responda com uma mensagem SYN – ACK para seu próprio endereço, o que é prontamente respondido com um ACK e faz com seja estabelecida uma conexão vazia. Cada conexão ficará assim até que o sistema operacional do SERVIDOR cancele a conexão por inatividade. Diversos sistemas operacionais respondem ao ataque Land de maneira diferente – o Windows NT fica extremamente lento por cerca de cinco minutos. Muitas implementações UNIX travam. A maioria dos fornecedores já supriu uma correção para proteção contra o ataque Land.

Quanto a se defender desse tipo de ataque, é sugerido aplicar as correções contra o ataque Land, os últimos Service Packs ou atualizações. Configure os Firewalls para recusar quaisquer pacotes na interface externa que tenham o endereço de origem interno. Isso sempre indica o endereço falso, e recusar esses pacotes deveria ser a política padrão de qualquer Firewall.

Os endereços IP a seguir são inválidos na *Internet* e sempre devem ser filtrados:

Domínio 10

Domínio 127

Domínios no intervalo de 172.16 a 172.31

Domínio 192.168

Além disso, o endereço IP atribuído deve ser filtrado.

Alguns sistemas operacionais, como Windows NT a partir do Service Pack 4, já estão atualizados contra este ataque e são imunes. No final de 1997 foi lançada uma variação do Land, o LA TIERRA. O La Tierra envia um pacote para um endereço IP de origem contendo como endereço IP destino o mesmo endereço IP de origem e para diversas portas (podendo variar os flags setados - SYN, ACK, FIN, etc).

2.4.3.4 TEARDROP

Este ataque consiste em explorar uma falha de implementação na montagem dos fragmentos de um pacote IP. O atacante coloca o endereço de início de um fragmento, um valor posterior ao endereço de fim do fragmento. Na hora da montagem dos fragmentos muitos sistemas operacionais se perdem e com isto desestabilizam assim a máquina. Os sistemas operacionais Windows 95 e Windows NT Workstation, inclusive já com o Service Pack 4, ainda não estão imunes a estes ataques, pois no teste de execução deste ataque sobre máquinas destes sistemas operacionais elas ficaram travadas. Geralmente nestes ataques o endereço IP de origem é modificado, para esconder a identidade do atacante. Em janeiro de 1998 foi lançada uma variação do Teardrop, então chamado de New Teardrop, onde a parte de dados do pacote é diminuída e o tamanho total do pacote UDP é falsificado. O New Teardrop afeta os sistemas operacionais Windows 95 e Windows NT Workstation, mesmo com o Service Pack 4 instalado, travando estas máquinas. Existe ainda o ataque SYNDROP, o qual mistura o ataque Teardrop com um SYN Flooding. A máquina atacante envia pacotes TCP com o flag SYN setado e com o mesmo problema de offsets do Teardrop, causando uma "negação de serviço". Este ataque atinge os seguintes sistemas operacionais Windows 95 e Windows NT Workstation com Service Pack 3, travando as máquinas. Quanto a se defender do TearDrop é recomendado adotar uma rotina de atualização do Sistema Operacional, podendo essa ser em forma de Service Pack e/ou Patchs, verificando a forma correta de acordo com o fornecedor do Sistema utilizado.

2.4.3.5 PING O'DEATH

Este ataque foi descoberto a um certo tempo e é bastante explorado na *Internet*. Os ataques de ping da morte são propagados criando-se um pacote de solicitação de echo ICMP malformado no qual o tamanho alegado do pacote excede o máximo possível. Como o indicador dos dados úteis tem 16 bits permitindo um tamanho máximo de pacote de 65.535 bytes (o limite real é cerca de 65.500 bytes devido aos dados de controle do cabeçalho do pacote), os pacotes que alegam serem maiores que 65.500 bytes podem causar erros de TCP/IP no sistema receptor. Em uma implementação TCP/IP típica, quando um cabeçalho de pacote é lido, conta-se com as informações contidas no cabeçalho para criar um buffer para os dados úteis. Quando o tamanho alegado do cabeçalho do pacote mais o tamanho dos dados úteis ultrapassa o limite máximo de 64 KB definido pela especificação TCP/IP, a implementação TCP/IP pode travar devido a erros na alocação de memória. Praticamente todas as plataformas eram afetadas por este ataque, e todas as que não tiveram correções de segurança instaladas, ainda o são. Este ataque recebeu o nome de Ping O' Death porque as primeiras ocorrências deste ataque foram a partir do programa ping, entretanto, qualquer pacote IP com mais de 65535 (pacote inválido) provoca o mesmo efeito. Quanto a se defender do Ping O'Death, todas as implementações TCP/I padrão foram corrigidas para lidar com pacotes de tamanho excessivo e a maioria dos Firewalls filtra automaticamente esses ataques. As versões do Windows desde o Windows 98, Windows NT com o Service pack 3, do Linux, do Solaris e do Mac OS, todas são imunes às variações padrão (conhecidas) do ping da morte. Configurar os Firewalls para bloquear ICMP e qualquer protocolo desconhecido evita esse tipo de ataque. Por esse motivo, esse ataque em sua forma atual já está obsoleto e não tem mais futuro.

2.4.3.6 INUNDAÇÃO UDP

Vários ataques de falsificação (*spoof*) exploram serviços TCP/IP simples como Chargen e Echo para transmitir dados inúteis que só ocupam largura de banda dos canais de transmissão. As inundações UDP são extremamente simples: forjando uma conexão UDP com o serviço Chargen executando em um *host* que tenha o endereço de resposta de um *host* executando o serviço Echo, um Invasor pode criar um fluxo de dados inúteis entre os dois *hosts*. A criação de um número suficiente desses fluxos causa uma recusa de serviço por falta de largura de banda. Quanto a se defender desse tipo de ataque, é sugerido configurar os *hosts*

para desativar serviços TCP/IP simples que não são necessários, assim como configurar os roteadores para bloquear solicitações UDP oriundas da *Internet* para esses serviços.

2.4.3.7 INUNDAÇÃO SYN

As inundações SYN são realizadas por meio de ataques simples que exploram o mecanismo de conexão do TCP. Como uma inundação SYN afeta o computador atacado depende de sua implementação de TCP/IP. Algumas implementações de pilha TCP/IP são capazes apenas de esperar pelas mensagens ACK de um número limitado de computadores, porque têm um buffer de memória limitado para estabelecer as conexões. Se esse buffer for preenchido com inicializações de conexões artificiais, o SERVIDOR irá parar de responder às tentativas de conexões subseqüentes até que as tentativas no buffer esgotem o tempo – limite (*timeout*). Em implementações que não limitam o estabelecimento de conexões, os ataques de inundação SYN têm um efeito similar. Como o SERVIDOR não sabe distinguir uma mensagem SYN legítima de uma falsa, ele reserva recursos de computação e de memória para estabelecer uma conexão. Sobrecarregando o SERVIDOR com um grande volume de solicitações, a capacidade máxima do SERVIDOR pode ser usada por essas tentativas de conexões artificiais e inúteis. Quanto a se defender desse tipo de ataque, a forma mais eficiente e recomendada é um bom Firewall que possa reconhecer as características inerentes a uma inundação SYN – várias tentativas idênticas de conexões vindas do mesmo endereço IP. Esses Firewalls podem filtrar conexões subseqüentes oriundas do mesmo *host*, eliminando assim os ataques de inundação SYN. Esse tipo de inundação SYN não seria diferente de um volume alto de tráfego e poderia passar pelos filtros contra sobrecarga SYN.

2.4.3.8 SMURF

O ataque Smurf é um ataque de recusa de serviço extremamente eficaz com base no recurso IP de endereçamento por broadcast (difusão) direto que possibilita a um *host* transmitir dados para todos os *hosts* de sua sub-rede. Um ataque Smurf simples ocorre inundando-se um *host-alvo* com pacotes de solicitações de eco ICMP (ping) contendo o endereço de resposta igual ao endereço de broadcast da rede da vítima. Isso faz com que todos os *hosts* da rede respondam à solicitação de eco ICMP, gerando assim ainda mais tráfego - normalmente um tráfego de uma a duas ordens de magnitude maior que a inundação ping inicial poderia gerar. Um ataque Smurf mais complexo ocorre como dito anteriormente, mas a

origem da solicitação de eco é definida como uma terceira vítima, que irá receber todas as solicitações de eco geradas pela sub-rede de *hosts-alvo*. Este ataque é inútil para os Invasores porque eles podem usar um enlace relativamente lento como um modem, para causar uma avalanche de tráfego ping enviada para qualquer ponto da *Internet*. Dessa maneira, um Invasor com um enlace mais lento que o de sua vítima pode ainda assim inundar o sistema dela aplicando o ataque Smurf a uma outra rede de velocidade mais alta que a da vítima. Quanto a se defender desse tipo de ataque, deve-se desativar o recurso de endereçamento por broadcast do roteador externo ou do Firewall. Para evitar ser a vítima final de um ataque Smurf, deve-se configurar o Firewall para recusar mensagens ping ICMP. Se um provedor de alta velocidade (como seu provedor de acesso a *Internet*) tiver sido usado com sucesso para atacar sua rede, não haverá nada que possa ser feito para aliviar o congestionamento que o tráfego ICMP irá causar mesmo se for filtrado do seu lado. Nesse caso, entre em contato com o seu provedor sobre a política de segurança de seu Firewall caso isso seja um problema.

2.4.3.9 FRAGGLE

O ataque Fraggle é uma variante simples do ataque Smurf, que usa mensagens de eco UDP em vez de ICMP. Essa característica possibilita ao ataque passar através de Firewalls que somente filtram ICMP. Quanto a se defender desse tipo de ataque é sugerido a filtragem das mensagens de eco UDP no Firewall.

2.4.3.10 BOMBAS DE E-MAIL

Os invasores podem sobrecarregar um SERVIDOR de e-mail enviando a ele repetidamente um mesmo arquivo de e-mail grande. As bombas de e-mail são um dos ataques mais antigos e chatos existentes. Configurando uma máquina para transmitir constantemente e-mail para o mesmo endereço, um Invasor pode ocupar a largura de banda da rede do receptor. Este ataque não é tão sério, considerando-se o fato de que ele requer quase tanta largura de banda do lado do transmissor quanto do lado do receptor que se deseja prejudicar, tornando difícil uma recusa de serviço realmente completa. Bombas de e-mails são fáceis de rastrear, a menos que estejam sendo transmitidas por meio de um *host* de correio eletrônico que remova os cabeçalhos.

2.4.4 EXPLORAÇÃO DE FALHAS E TENTATIVAS DE ACESSO

2.4.4.1 VULNERABILIDADE

É uma falha no sistema operacional, protocolo, serviços ou quaisquer outros componentes no sistema que permitem acesso ou intervenção de pessoas não autorizadas. A vulnerabilidade existe independente do ataque, ela não depende do tempo de observação.

2.4.4.2 BASEADOS EM SENHAS

O ataque ao arquivo de senhas do sistema mais conhecido é chamado de Ataque do Dicionário. Este ataque foi criado por Robert Morris, coincidência ou não, filho de Robert Morris da NSA (*National Security Agency*) que foi um dos pesquisadores que desenvolveu o crypt( ), função a qual cifra as senhas nos sistemas Unix. O ataque consiste na cifragem das palavras de um dicionário através da função crypt( ), e posterior comparação com os arquivos de senhas de usuários. Desta forma, quando uma palavra do dicionário cifrada coincidisse com a senha cifrada de um usuário, o atacante teria obtido uma senha. Para dificultar este ataque foi criado o chamado SALT *(Society for Applied Learning Technology*®). O SALT é composto de dois números randômicos gerado na hora em que o usuário está inserindo ou alterando a sua senha. O número gerado pode estar entre 0 e 4095 e é cifrado juntamente com a senha, o que impede a utilização de um dicionário genérico para todos os usuários. O atacante agora tem que cifrar cada palavra do dicionário com o SALT de cada usuário. Depois de obter acesso a rede alvo, o Invasor já pode ter acesso a arquivos do usuário o qual quebrou a senha, ler e-mails, manter o usuário válido sem poder acessar a rede, dentre outras opções. Como defesa para tal tipo de ataque, utilize sempre senhas que sejam difíceis de adivinhar, com combinações de letras e pontuação. Certifique-se de que serviços que possam ser explorados como o NFS, NetBIOS e telnet não estejam expostos ao público. Estabeleça políticas de bloqueio caso o serviço o permita.

2.4.4.3 EXPLORAÇÃO DE SISTEMAS CONFIÁVEIS

São comuns sistemas operacionais de rede que incorporem mecanismos de acessos baseado em confiança. Tanto servidores da família Microsoft como o Windows NT, Windows 2000, Windows 2003, assim como os da família Unix possui esta característica, mas ela é

particularmente insegura em sistemas Unix. No Unix, os usuários podem criar arquivos com os *hosts* confiáveis. A partir do momento que um *host* confia em um outro, os usuários que tenham o mesmo *login* em ambos os *hosts* podem passar de um ao outro sem ter que digitar a senha novamente, ou seja, o usuário uma vez autenticado, ele passa ter acesso a todos os outros *hosts* que confiem no que ele está logado. Existe também o conceito de usuários confiáveis e funciona da mesma forma que os *hosts* confiáveis. Se um usuário for identificado como um usuário confiável em uma conta qualquer, ele pode logar-se nesta conta sem ter que digitar a senha. *Hosts* e usuários confiáveis têm muitas vantagens, principalmente em ambientes fechados, pois fica muito mais rápido passar de uma máquina para a outra. Praticamente, os *hosts* confiáveis fazem com que uma vez o usuário identificado frente a uma máquina, está identificado frente a toda a rede. Os *hosts* e usuários confiáveis são configurados no UNIX através do arquivo /etc/hosts.equiv e ~/.rhosts. Existe também o arquivo /etc/hosts.lpd. Este arquivo lista os *hosts* que possam utilizar a impressora local. Assim, pode ser permitido a alguns computadores imprimir sem torná-los *hosts* confiáveis. *Hosts* e usuários confiáveis têm sido responsáveis por diversos problemas de segurança, uma vez que, uma máquina seja invadida, todas as que confiam naquela também estão comprometidas. Além disso, os arquivos .rhosts estão sendo utilizados como backdoors. "*Backdoors*" são programas que instalam um ambiente de serviço em um computador, tornando-o acessível à distância, permitindo o controle remoto da máquina sem que o usuário saiba. O atacante adiciona o seu *login* no arquivo para facilitar a sua entrada no sistema no futuro, deixando assim um backdoor. Por estes riscos é que muitos administradores de sistema desabilitam a utilização do arquivo .rhosts. Isto é possível de duas formas, ou se consegue o código fonte dos programas rlogin e rsh, e remove esta característica, ou então percorrer os diretórios dos usuários periodicamente verificando quem está usando este arquivo. [STR1999]

Os comandos "r" do Unix são os responsáveis por utilizar estes arquivos, entre eles o melhor exemplo é o rlogin, o qual provê um serviço de terminal remoto muito semelhante ao telnet, diferenciando por duas diferenças básicas:

- no rlogin o username é automaticamente transmitido no início da conexão, não sendo necessário ao usuário digitar o seu *login*.
- o rlogin não exige senha caso a conexão venha de um *host* confiável ou de um *login* confiável.

2.4.4.4 ESTOURO DE BUFFER

O estouro de buffer é uma falha de programação que ocorre quando a quantidade de bytes que é enviado para uma determinada variável ultrapassa o tamanho destinado para esta variável fazendo com que ela tome a área de memória seguinte. Os estouros de buffer se aproveitam do fato de que a maior parte dos programas reserva blocos de memória de tamanho fixo para criar uma área de armazenamento temporária chamada buffer, na qual os programas processam as informações oriundas da rede. O nome buffer overflow, que significa o excesso do buffer, indica que o limite de memória disponível foi ultrapassado. Isso ocorre, porque muitos programadores não têm a preocupação de verificar o espaço destinado a entrada de dados.

É uma classe de ataques que exploram uma fraqueza comum a todo *software*. Ataques de estouro de buffer novos são descobertos a toda hora. Os estouros de buffer ocorrem quando uma mensagem "mente" sobre seu tamanho ou foi criada deliberadamente maior que o tamanho máximo permitido. Por exemplo, se uma mensagem informa que tem 240 bytes, mas na verdade tem 256, o serviço que a recebe e processa poderá alocar um buffer de somente 240 bytes, mas depois tentar copiar 256 nesse buffer. Os 16 bytes da memória além do final do buffer serão sobrescritos com o que a mensagem contiver. Os invasores se aproveitam desses problemas incluindo o código em linguagem de máquina na seção da mensagem que ultrapassa o final do buffer. Ainda mais perturbador é o fato de que o *software* é escrito freqüentemente de tal modo que a execução do código começa exatamente depois do final do buffer. Permitindo assim que os invasores executem código no contexto de segurança do serviço sendo executado, escrevendo um pequeno programa para abrir uma brecha na segurança adicional e colocando esse código na parte citada do buffer, eles conseguem o controle do sistema. Quanto a se defender desse tipo de ataque, a única defesa é estar atualizado com os boletins relativos à segurança mais recente do Sistema Operacional a aplicativos instalados no Sistema.

2.4.4.5 HIJACKING ATTACK

Este ataque consiste em um seqüestro de uma conexão já existente entre dois *hosts*, A e B. O invasor depois de ter acesso ao *host* A com privilégios de root (citando como exemplo o Unix), assume todas as conexões existentes com o *host* A, inclusive executando comandos como se fosse o proprietário da conexão.

2.4.4.6 ATAQUE DE DESSINCRONIZAÇÃO

Este ataque consiste em quebrar a conexão nos seus primeiros estágios e criando uma nova conexão com número de seqüência diferente.

- Quando o atacante ouve que o SERVIDOR mandou um SYN/ACK para o cliente, ele manda ao SERVIDOR um pacote de RST e um pacote de abertura de conexão SYN com os mesmos parâmetros (porta TCP), mas número de seqüência diferente (referenciado como ATAC-ACK0).
- O SERVIDOR fecha a primeira conexão e reabre outra na mesma porta, mas com outro número de seqüência (SRV-SEQ0) e envia o SYN/ACK ao cliente.
- O atacante, detectando o SYN/ACK do SERVIDOR, envia um ACK para ele.
- O SERVIDOR passa para o estado ESTABLISHED e o cliente que já havia recebido um SYN/ACK do SERVIDOR também se encontra no estado ESTABLISHED.

Neste momento as duas pontas estão no estado ESTABLISHED numa conexão dessincronizada. [STR1999]

2.4.4.7 FALHAS EM IMPLEMENTAÇÕES DE SERVIÇOS INTERNET

O protocolo TCP/IP foi projetado para utilização através de meios de comunicação seguros, ou seja, não foi prevista uma utilização mundial com informações cruzando o mundo nem tampouco para comunicações particulares através de meios de comunicação compartilhados como os backbones *Internet*. Sendo assim, em sua concepção original não se preocupa com mecanismos que possam garantir a privacidade e a autenticidade das comunicações. O conjunto de protocolos TCP/IP é intrinsecamente inseguro, pois os dados que trafegam nos pacotes IP não passam por nenhum mecanismo que garanta integridade, privacidade, autenticidade, não-repúdio entre outros. Por sua vez, os serviços que funcionam em cima destes protocolos também são inseguros. A seguir serão citados alguns dos problemas dos serviços *Internet* que estão diretamente ligados ao protocolo. [STR1999]

FINGER - PORTA 79

Um erro de implementação no finger possibilitou que o verme da *Internet* de Robert Morris, em 1988, aproveitando-se desta falha, invadisse diversos sistemas. A implementação

do finger permitia a um programa mal intencionado ultrapassar os 512 bytes e causar um buffer overflow, interrompendo a execução do fingerd e com isso abrindo um shell, dando o programa acesso irrestrito ao sistema. Em seguida, as implementações do finger corrigiram este problema. [STR1999]

MAIL

Através do protocolo SMTP (*Simple Mail Transfer Protocol*) qualquer usuário envia uma mensagem de uma máquina para outra através da rede. O mail é o serviço mais difundido da *Internet*. O SMTP em si não possui problemas de segurança e sim as implementações de seus servidores, como o *sendmail*. O *sendmail* é uma das implementações mais difundidas do SMTP e atua tanto como SERVIDOR como cliente. O *sendmail* já é conhecido como fonte de problemas de segurança. Ele possui tanto falhas de implementação que são corrigidas periodicamente, como possui características que podem ser exploradas por Invasores. Uma das vulnerabilidades do *sendmail* envolve em fazer o SERVIDOR executar o corpo de uma mensagem com um Shell script. O script pode fazer qualquer coisa que um usuário comum do sistema possa fazer incluindo enviar o arquivo de senhas do sistema para um atacante. O *sendmail* praticamente não possui validação dos dados e permite também que seja enviado um mail para um arquivo, inclusive para o arquivo /etc/passwd, fazendo com que um usuário possa adicionar o seu *login* num arquivo de senhas. O *sendmail* possui alguns comandos que devem ser desabilitados pelo administrador do sistema, como, “kill”, “debug”, para evitar que o sistema fique muito vulnerável a ataques. O arquivo de configuração é encontrado em local variado de acordo com a distribuição e versão do Sistema Operacional utilizado. O arquivo de alias também merece cuidado, deve ser evitado que através de um alias seja enviado um mail diretamente para um programa, como por exemplo:

uudecode: “| /usr/bin/uudecode”

Outro grande problema do correio eletrônico é a facilidade em forjar um e-mail através de um telnet na porta 25 de um SERVIDOR de mail, desta vez não comprometendo o sistema, mas sim a reputação dos usuários. O verme da *Internet* de 1988 utilizou-se do modo de depuração do *sendmail* para obter acesso irrestrito ao sistema invadido. O *sendmail* possui este modo de depuração com o objetivo de permitir acesso irrestrito para configuração do *sendmail*. Depois de instalado o *sendmail*, este deve ser recompilado com o modo de depuração desligado, para que não haja riscos de invasão. Como o *sendmail* roda como superusuário, a sua invasão compromete a segurança de todo o sistema. Como visto, o

sendmail é um dos serviços mais fracos em nível de segurança e um dos mais utilizados. Como a maioria dos problemas do *sendmail* está na sua configuração e também na utilização da última versão, um bom administrador de sistema pode garantir uma boa segurança ao *sendmail*. É importante lembrar que as últimas versões estão sempre corrigindo "furos" das versões antigas, por isto é sempre aconselhável à utilização da última versão. [STR1999]

WWW

Como toda a estrutura da *Internet*, no WWW o programa cliente (o browser) que faz as requisições de documentos e o SERVIDOR (SERVIDOR Web) que deixam as informações disponíveis na Web. Uma forma encontrada de expandir mais ainda a utilização da Web é a colocação de programas por de trás das páginas Web. Os programas podem ser scripts, Perl, Java, Javascript, ActiveX, entre outros. O Web, como todos os serviços da *Internet*, também possui os seus problemas de segurança. Entre eles estão os furos que podem haver nos servidores Web ou inclusive nos programas que podem ser explorados por um atacante a fim de obter acesso não autorizado ao sistema. Os servidores Web são projetados para receber requisições anônimas de computadores não autenticados e para responderem de maneira rápida e eficiente. Com isso estão muito suscetíveis a atacantes buscando furos no código fonte para conseguir acesso ao sistema. Por sua vez os programas de servidores Web são complicados e geralmente tem o seu fonte disponível, possibilitando assim ao atacante achar vulnerabilidades e atacar o sistema. A habilidade de adicionar funções nos Web browser com programas complica ainda mais a segurança. Assim como um programa pode adicionar novas características a um SERVIDOR Web, pode também inserir novos riscos a segurança do site. Por exemplo, de nada adianta um SERVIDOR Web ser cuidadosamente configurado para não acessar outros diretórios e lá ser instalado um CGI que permite que alguém de fora leia qualquer arquivo do sistema, comprometendo toda a segurança do sistema. Uma sugestão para resolver este problema é a utilização de um computador exclusivamente com o SERVIDOR Web, não permitindo nenhum outro serviço como rlogin, rsh, etc. A utilização de uma máquina exclusivamente para o SERVIDOR Web dificulta ao atacante quebrar o sistema, e se por acaso conseguir, isto reduzirá os danos causados por uma invasão. Outros problemas podem ser os bugs dos Web browser, um bug de um browser pode ser explorado por um SERVIDOR Web não confiável. Muitos browser´s podem ser configurados para executar automaticamente alguns tipos de arquivos quando baixados da rede. Isto é

desaconselhável, pois proporciona uma maneira de atacantes rodarem programas sem pedir permissão ao usuário, um programa pode estar incluído em um documento HTML.

2.4.4.8 PROTOCOLO NETBIOS

Praticamente todas as redes locais existentes hoje em dia possuem PCs rodando plataformas Windows 3.11, Windows 95, Windows NT, Windows 2000, Windows XP, Windows 2003 ou OS/2. Estas plataformas possuem a característica de disponibilizar todos os seus serviços de rede através do protocolo NETBIOS. Ocorre também que praticamente todas estas máquinas rodam o protocolo TCP/IP, possibilitando que o NETBIOS seja encapsulado sobre o TCP/IP. Os serviços da rede Microsoft foram projetados originalmente para uma rede local. O problema é que ao conectar esta rede local via TCP/IP a *Internet* as portas serão abertas para que pessoas de outras redes tenham acesso aos seus recursos. Se as máquinas estão conectadas à *Internet*, é possível para um atacante verificar quais são os diretórios e impressoras compartilhadas por cada uma delas. Além disso, devido ao fato de muitas vezes os compartilhamentos serem feitos indiscriminadamente, normalmente pelos próprios usuários, é muito comum se conseguir acesso de leitura ou de gravação em muitos diretórios. Desta forma um atacante pode ter acesso à informações confidenciais, apagar informações importantes ou mesmo implantar arquivos contaminados nas máquinas da rede interna. Além disso, até pouco tempo atrás, as máquinas Windows NT eram vulneráveis a diversos tipos de ataques realizados contra os serviços NETBIOS, que terminavam por travar o sistema ou inutilizar completamente todos os serviços (os ataques podiam ir de um simples telnet para a porta 139 até ataques mais sofisticados). Mesmo hoje em dia ainda existem muitas máquinas vulneráveis, devido ao fato de ser necessário a aplicação de correções específicas para estes problemas, coisa que muitos administradores não fazem. De fato a enorme maioria das invasões e outros incidentes de segurança em NTs ligados a *Internet* utilizaram NetBIOS trafegado sobre TCP/IP como seu canal. Há um modo simples de evitar que isto aconteça: impedindo a entrada de NetBIOS na rede interna. Com este impedimento, Invasores e outros não poderão acessar os recursos da rede interna, roubar senhas ou tentar derrubar os servidores. Para impedir a entrada de NetBIOS, o filtro de pacotes deve impedir a entrada de pacotes da rede externa destinados as seguintes portas: 135 (utilizado pelo RPC), 137 e 138 (UDP) e 139 (TCP) utilizadas pelo NetBIOS. Um ataque específico a plataformas Windows é o WIN NUKE, o qual envia um pacote TCP para a porta 139 da máquina alvo com o flag de dados “Out of Band” setado, causando um erro na máquina alvo (que não esta preparada para

receber esse tipo de dados). Este ataque ocasiona a paralisação de uma máquina com Windows 95. [STR1999]

2.4.4.9 X-WINDOWS

X-Windows é o padrão de interface gráfica utilizado em máquinas Unix. Por ser um padrão que foi criado para uso em redes locais. É possível se executar um programa em uma máquina e mostrar sua saída em outra máquina, localizada em qualquer lugar da *Internet*. É também possível um usuário ter programas sendo executados em diversas máquinas e todos eles sendo visualizados como se fossem aplicações locais, cada um na sua janela. Toda esta flexibilidade, entretanto, tem problemas de segurança: caso disponha de autorização do SERVIDOR X-Windows (a máquina onde está a tela gráfica), é possível para um atacante exercer um controle completo sobre a tela, capturando imagens e caracteres ou mesmo enviando caracteres para outras aplicações. É possível até mesmo simular o movimento do mouse ou encerrar outras aplicações. Esta autorização muitas vezes não é difícil de ser conseguida: o X-Windows possui dois métodos de autenticação, um a nível de usuário e outro a nível de máquina. Quando se está usando autenticação a nível de máquina, um usuário pode através de IP spoofing se conectar ao SERVIDOR e passar a ter o controle total da tela. Mesmo a autenticação ao nível de usuário não é muito segura, uma vez que as informações para acesso trafegam em texto pleno pela rede (de acordo com os serviços utilizados). Além disso, existem vulnerabilidades em alguns servidores X-Windows de modo que é possível travá-los com um simples telnet para a porta do SERVIDOR. Este travamento pode durar até a conexão ser interrompida. [COND]

2.5 PRAGAS DIGITAIS

Vírus, Worms e Cavalos de Tróia são programas mal-intencionados que podem causar danos ao seu computador e às informações armazenadas nele. Também podem deixar a *Internet* mais lenta e usar o seu computador para espalharem-se entre os seus amigos, familiares, colegas de trabalho e o restante da Web. A boa notícia é que, com prevenção e algum bom senso, se terá menos probabilidade de ser vítima dessas ameaças.

2.5.1 VÍRUS

Um vírus é um código de computador que se anexa a um programa ou arquivo para poder se espalhar entre os computadores, infectando-os à medida que se desloca. Os vírus podem danificar seu *software*, *hardware* e arquivos. Um programa ou um conjunto de instruções executáveis com habilidade para danificar arquivos ou para exibir mensagens estranhas.

Assim como os vírus humanos possuem níveis de gravidade diferentes, como o vírus Ebola e o vírus da gripe, os vírus de computador variam entre levemente perturbador e totalmente destrutivo. A boa notícia é que um verdadeiro vírus não se dissemina sem ação humana. É necessário que alguém envie um arquivo ou envie um e-mail para que ele se alastre.

Hoje, em um mundo conectado, com uma população de internautas de aproximadamente 500 milhões, trocando diariamente arquivos e mensagens eletrônicas, a *internet* é o mecanismo adequado para a proliferação dos vírus e a contaminação dos computadores e sistemas operacionais. [SECN]

2.5.2 WORMS

Um worm, assim como um vírus, cria cópias de si mesmo de um computador para outro, mas faz isso automaticamente. Primeiro, ele controla recursos no computador que permitem o transporte de arquivos ou informações. Depois que o worm contamina o sistema, ele se desloca sozinho. O grande perigo dos worms é a sua capacidade de se replicar em grande volume. Por exemplo, um worm pode enviar cópias de si mesmo a todas as pessoas que constam no seu catálogo de endereços de e-mail, e os computadores dessas pessoas passam a fazer o mesmo, causando um efeito dominó de alto tráfego de rede que pode tornar mais lenta as redes corporativas e a *Internet* como um todo. Quando novos worms são lançados, eles se alastram muito rapidamente. Eles obstruem redes e provavelmente fazem com que os clientes tenham de esperar um tempo maior para abrir páginas na *Internet*.

Um worm pode consumir memória e largura de banda de rede, o que pode travar o seu computador.

Como os worms não precisam viajar através de um programa ou arquivo "hospedeiro", eles também podem se infiltrar no seu sistema e permitir que outra pessoa

controle o seu computador remotamente. Exemplos recentes de worms incluem o worm Sasser e o worm Blaster.

2.5.3 CAVALO DE TRÓIA

Assim como o mitológico cavalo de Tróia parecia ser um presente, mas na verdade escondia soldados gregos em seu interior que tomaram a cidade de Tróia, os cavalos de Tróia da atualidade são programas de computador que parecem ser úteis, mas na verdade comprometem a sua segurança e causam muitos danos. Um cavalo de Tróia apresentava-se como um e-mail com anexos de supostas atualizações de segurança da Microsoft, mas na verdade era um vírus que tentava desativar programas antivírus e firewalls.

Os cavalos de Tróia se alastram quando as pessoas que são seduzidas a abrir o programa o executam por pensar que vem de uma fonte legítima. Eles também podem ser incluídos em *software* que são distribuídos gratuitamente.

Nunca se deve baixar *software* de uma fonte não confiável.

2.5.4 ANTI-VÍRUS

O Anti-vírus é uma ferramenta criada para impedir a contaminação de um computador por um vírus. A cada dia novos vírus e suas variações surgem como pragas. Por esta razão, ele deve ser atualizado constantemente, para evitar que o seu computador seja contaminado.

Quanto mais atualizado, mais seguro estará o sistema, pois dezenas de novos vírus aparecem a cada dia. Muitas vezes os fabricantes liberam uma ou duas atualizações por semana mas, em semanas de intensa atividade de vírus, worms e outros, a freqüência destas atualizações aumenta consideravelmente. Além disso, os anti-vírus devem ser configurados para verificar automaticamente anexos, mídias removíveis e arquivos e executar varreduras regularmente.

2.6 POLÍTICAS DE SEGURANÇA

Política de Segurança é uma série de normas internas padronizadas pela empresa que devem ser seguidas à risca para que todas as possíveis ameaças sejam minimizadas e combatidas eficientemente pela equipe de segurança. Ela define o que é permitido e o que é

proibido em um sistema. Existem basicamente duas filosofias por trás de qualquer política de segurança:

- Proibitiva – tudo que não é expressamente permitido é proibido;
- Permissiva – tudo que não é expressamente proibido é permitido;

Geralmente as instituições mais preocupadas com a segurança adotam a primeira abordagem. Uma política deve descrever exatamente quais operações são permitidas em um sistema. Qualquer operação que não esteja descrita de forma detalhada na política de segurança deve ser considerada ilegal ao sistema. [REV1999]

Muitos usuários respeitam o conjunto de regras sociais. Estas regras encorajam que uns respeitem aos outros tanto a privacidade quanto o ambiente de trabalho. Então se torna muito fácil subverter a confiança em um ambiente onde existe um acordo de confiança, ou seja, é relativamente fácil invadir um sistema onde os usuários confiam uns nos outros.

Diante disso torna-se necessário a documentação e divulgação das regras que primam pela manutenção da privacidade e integridade.

2.6.1 ELEMENTOS QUE COMPÕEM UMA POLÍTICA DE SEGURANÇA

Um sistema de computadores pode ser considerado como um conjunto de recursos que é disponibilizado para ser utilizado pelos usuários autorizados.

Existe um documento escrito por Donn Park [PAR1994], que descreve seis elementos que devem ser contemplados em uma política de segurança [REV1999]:

- Disponibilidade - o sistema deve estar disponível para uso quando o usuário precisar. Dados críticos devem estar disponíveis de forma ininterrupta;
- Utilização – O sistema e os dados devem ser utilizados para as devidas finalidades;
- Integridade – O sistema e os dados devem estar completamente íntegros e em condições de serem utilizados;
- Autenticidade – O sistema dever ter condições de verificar a identidade do usuário e o usuário deve ter condições de verificar a identidade do sistema;
- Confidencialidade – Os dados privados devem ser apresentados somente para os donos dos dados ou para o grupo de usuários para o qual o dono dos dados permitir;
- Posse – O dono do sistema deve ter condições de controlá-lo.

Capítulo 3

Descreve os principais tipos de IDS, algumas formas físicas para exploração e captura

3 IDS

3.1 TIPOS DE IDS

Os IDS são caracterizados segundo algumas formas que utilizam para detectar uma invasão na rede à qual estão analisando. A seguir temos os principais tipos de IDS.

3.1.1 IDS BASEADO EM HOST

Os primeiros sistemas baseados em detecção de intrusão foram construídos baseados na técnica de *Host Based*, com um agente inteligente rodando no SERVIDOR sendo monitorado.

Esse agente avalia diferentes aspectos da segurança do SERVIDOR, como os arquivos de logs do sistema operacional, logs de acesso, logs da aplicação, etc. Essas formas de IDS dependem da trilha de auditoria criada pelos sistemas operacionais ou pelas aplicações. [BRI2002]

A *Figura 06* mostra os IDS baseados em *host*. Esses foram desenhados para monitorar somente o SERVIDOR onde o agente está instalado, o que restringe as habilidades destes sistemas a monitorar apenas as atividades geradas pelas trilhas de auditoria do Sistema Operacional ou da Aplicação.

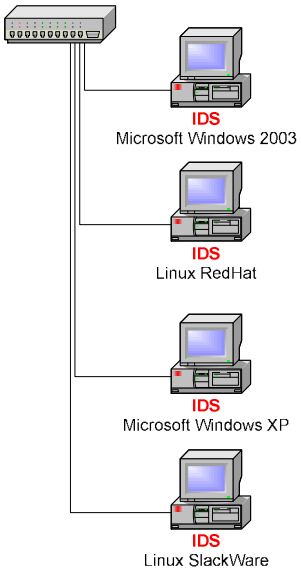

Figura 06 – Esquema de IDS Baseado em Hosts

Esse modelo pode ainda detectar atividades que ocorrem localmente no SERVIDOR sendo monitorado, como modificação de permissão de arquivos, criação ou modificação de atributos de usuários, etc.

3.1.2 IDS BASEADO EM REDE (NIDS)

A evolução das empresas para a utilização de grandes redes requer que a monitoração de dados seja feita em todos os níveis.

Devido a esta requisição, foram desenvolvidos os Sistemas de Detecção de Intrusão baseados em rede (*Network-Based*). [BRI2002]

A *Figura 07* mostra esquema dos IDS's baseados em Rede. Esses realizam a sua detecção de um ataque mediante a comparação de parâmetros das sessões e comandos do usuário com uma base de regras e técnicas normalmente utilizadas pelos invasores para ganhar acesso a um sistema. Essas técnicas, chamadas de assinaturas de ataques, são o que estes sistemas procuram diante o comportamento de um usuário. Este tipo de abordagem, que procura identificar padrões conhecidos de problemas de segurança, é chamado de modelo de detecção por uso inadequado.

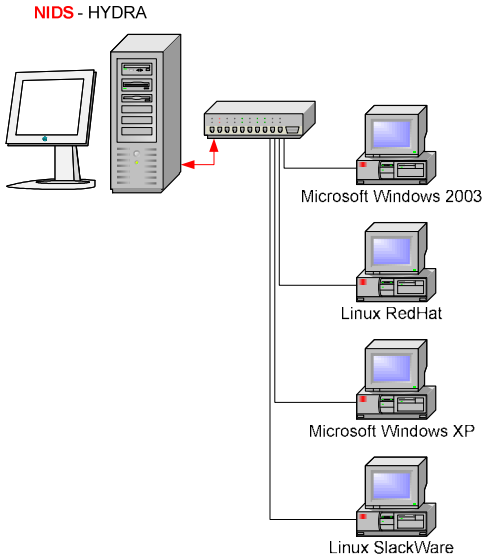

Figura 07 – Esquema de IDS Baseado em Redes

Os NIDS possuem geralmente dois componentes:

Sensores: São colocados em seguimentos de rede distintos, nos quais se deseja monitorar.

Estação de gerenciamento: Possui uma interface gráfica e recebe os alarmes dos sensores informando ao operador.

3.1.3 IDS HÍBRIDO

O IDS híbrido consiste em combinar os aspectos positivos do HIDS e do NIDS, para que a detecção de intrusão se torne mais eficaz. Esses sistemas operam como se fossem NIDS, capturando e processando pacotes do tráfego da rede e detectando e reagindo a ataques. Porém, efetuam esse processo como HIDS, ou seja, processando os pacotes endereçados ao próprio sistema. Dessa maneira, é possível resolver o problema de desempenho dos IDS's baseados em rede. Entretanto, o problema da escalabilidade em sistemas baseados em *host* continua, sendo que um IDS híbrido deve ser instalado em cada equipamento monitorado. [FER2003]

3.2 FORMAS FÍSICAS PARA EXPLORAÇÃO E CAPTURA

Será descrito nesse tópico as formas de conexão físicas e modificações lógicas que podem auxiliar ou prejudicar o invasor na captura dos dados.

3.2.1 IDS E SWITCHES

A implementação de ferramentas IDS baseadas em redes comutadas tem sido estudada com bastante afinco pelos pesquisadores. Os switches permitem a comunicação direta, não compartilhada entre dois dispositivos pertencentes a uma rede. Embora esta característica permita um ganho alto na performance, ela introduz algumas limitações à implementação do IDS.

3.2.2 IDS E HUB

Esta configuração não é recomendada, pois a chance de causar um problema maior de rede é muito grande. O Hub é colocado entre as conexões a serem monitoradas. Isso é comum entre dois switches, um roteador e um switch, ou um SERVIDOR e um switch, pois permite que o tráfego ainda flua entre o roteador e o switch, enquanto as propriedades do hub causam cópia do tráfego pra fora do IDS.

3.2.3 PORT SPAN

Esta parece ser uma das soluções mais utilizadas (ou a mais desejada) em switches, onde os dispositivos de rede (servidores, roteadores, etc.) estão conectados, por exemplo, a portas de 10 Mbps, e um IDS é conectado à porta SPAN 11 de 100 Mbps, recebendo uma cópia de todo o tráfego do switch. Desta maneira, obtém-se uma monitoração completa do tráfego, simulando um ambiente compartilhado. Alguns switches possuem características avançadas de monitoração, por exemplo, de VLANs específicas, portas em trunking, distribuição da monitoração para várias portas de destino ou até de portas de outros switches da rede. Neste caso, é possível monitorar com IDS's o tráfego tratado por um switch inteiro ou apenas o tráfego pertencente a alguns subconjuntos específicos de sistemas, como as VLAN's.

3.3 HONEYPOT

É um sistema que possui falhas de segurança reais ou virtuais, colocadas de maneira proposital, a fim de que seja invadido e que o fruto desta invasão possa ser estudado. Um honeypot (Pote de Mel) pode ser uma máquina executando serviços falsos, que respondem como seus originais, mas na realidade estão fazendo outras operações totalmente diferentes.

Estes programas que executam este tipo de atividade são chamados de fake servers. Um atacante executa um telnet para uma máquina deste tipo. O fake server emulando o telnet, responde, mas neste momento, já capturou várias informações do atacante úteis para seu rastreio. Esta definição é muito simplista, mas nos ajuda a começar. A história dos honeypots começa em 1991 com a publicação dos papers "The Cuckoo´s Egg" de Cliford Stools e "An Evening with Berferd" de Bill Cheswicks, onde foram lançadas a bases do projeto e a priemira análise de um ataque. Em 1997 Fred Cohen´s lança o DTK, o primeiro honeypot, que era aberto e gratuito. O grande ponto importante foi que em 1999 surgiu o Honeynet Project, onde foi criada por Lance Spitzner uma entidade formada por cerca de 50 especialistas de segurança para num esforço, enumerar e lançar ferramentas de defesa contra ataques. Um dos grandes resultados foi o lançamento do honeyd, uma execelente solução em *software* livre de *honeyPot*, mantido por Niels Provos.[5]

Os *honeyPots* também podem ser ainda máquinas propositadamente compromissadas, com serviços vulneráveis, mas com um aparato de segurança invisível para o invasor. Obviamente que este tipo de *honeypot* é mais perigoso e, principalmente para o iniciante. [HONE]

3.3.1 TIPOS E NÍVEIS DE HONEYPOTS

3.3.1.1 HONEYPOTS DE PESQUISA

Os *HoneyPots* de pesquisa têm como características acumular o máximo de informações dos atacantes e suas ferramentas, tem um alto grau de comprometimento e normalmente são utilizados em redes externas ou sem ligação com rede principal.

3.3.1.2 HONEYPOTS DE PRODUÇÃO

Os *HoneyPots* de produção têm como características diminuir risco, tornando-se agora um elemento de distração ou dispersão.

[5] Niels Provos, um dos desenvolvedores responsáveis pelo projeto Honeyd - http://www.citi.umich.edu/u/provos/

3.3.1.3 BAIXA INTERATIVIDADE

Os *HoneyPots* de Baixa Interatividade têm como características:

- Serviços Falsos;
- Geralmente ficam de listener TCP/UDP;
- Geram sempre respostas falsas.

3.3.1.4 MÉDIA INTERATIVIDADE

Os *HoneyPots* de Média Interatividade têm como características:

- A representação de um ambiente falso;
- Criação de uma ilusão de domínio da máquina;
- Estudo melhor das técnicas utilizadas;
- Invadir o sistema.

3.3.1.5 ALTA INTERATIVIDADE

Os *HoneyPots* de Alta Interatividade têm como características:

- SO com serviços comprometidos;
- Não perceptível ao atacante;
- Estudo melhor das técnicas utilizadas;
- Vários riscos;
- Utilização como trampolim;
- Repositório de informações roubadas.

3.3.2 PROJETO HONEYNET

O Projeto de *Honeynet* é um projeto sem fins lucrativos, uma organização de pesquisa de profissionais de segurança dedicados a segurança de informação. Sem nenhum produto, serviços ou empregados, todas as pesquisa são feitas por pessoas voluntárias. A meta é aprender as ferramentas, táticas, e motivos da comunidade *blackhat* "chapéus negros" e

compartilhar estas lições aprendidas. Espera-se que tal pesquisa beneficiará a todos os membros da rede e a comunidade de segurança.

3.3.3 PROJETO HONEYPOTBR

Surgiu através de um grupo de especialistas em segurança e pesquisadores independentes, sendo esses inspirados no Projeto *HoneyNet* de Lance Spitzner.

3.3.4 FERRAMENTAS

- Fake Echo – Daniel B. Cid
- Fake Ftp – Fabio Henrique
- Fake Http – Adriano Carvalho
- Fake Pop3 – Humberto Sartini
- Fake Smtp – Daniel B. Cid
- Fake Squid – Antonio Marcelo
- Fake Telnet – Daniel B. Cid

3.4 PADRONIZAÇÕES DOS IDS

3.4.1 CIDF

Trata-se de um modelo conceitual para ferramentas de detecção de intrusão que propõe o agrupamento de um conjunto de componentes que definem uma ferramenta de IDS. [SECN]

3.4.1.1 COMPONENTES DO MODELO CIDF

O modelo CIDF, é dividido em quatro componentes que devem estar operando simultaneamente, como segue:

GERADOR DE EVENTOS (E-BOX)

A função deste componente é obter os eventos a partir do meio externo ao CIDF, ou seja, ele "produz" os eventos, mas não os processa. Isso fica a cargo do componente especializado na função de processamento, que por sua vez após analisar os eventos (violação de política, anomalias, intrusão) envia os resultados para outros componentes. [REV1999]

ANALISADOR DE EVENTOS (A-BOX)

Este componente basicamente recebe as informações de outros componentes, analisa estas informações e as envia de forma resumida para outros componentes, ou seja, recebe os dados de forma bruta, faz um refinamento e envia para outros. [REV1999]

DATABASE DE EVENTOS (D-BOX)

A função deste componente é armazenar os eventos e/ou resultados para uma necessidade futura. [REV1999]

UNIDADE DE RESPOSTA (R-BOX)

Este componente é responsável pelas ações, ou seja, matar o processo, resetar a conexão, alterar a permissão de arquivos, notificar as estações de gerência, etc. [REV1999]

3.4.2 INTEROPERABILIDADE DE IDS

A padronização proposta pelo modelo CIDF, permite que vários sensores IDS possam trabalhar em conjunto, interoperando de forma simbiótica. São 6 os cenários nos quais os IDS podem interoperar:

- Análise;
- Complemento;
- Reforço;
- Verificação;
- Ajuste de monitoração;
- Resposta.

3.4.3 CISL

A CISL (*Common Intrusion Specification Language*) é a tentativa de padronizar transferência de informações e interoperabilidade entre IDS, faz parte do CIDF e pode ser utilizada na comunicação dos sensores com a estação de gerenciamento, comunicação entre os sensores ou comunicação com firewall ou roteadores. [BRI2002]

3.4.4 IAP

O IAP (*Intrusion Alert Protocol*) foi proposto por um grupo do IETF e é um protocolo a nível de aplicação para troca de dados entre agentes IDS. O IAP utiliza o TCP como protocolo a nível de transporte é primariamente destinado a transmissão de dados do sensor/analisador para a estação de gerenciamento que informa a ocorrência, grava o evento e toma as determinadas contramedidas. [BRI2002]

Capítulo 4

Descreve os tipos de protocolo utilizados para implementação da ferramenta NIDS

4 PROTOCOLOS DE REDE

Será exibido neste tópico a descrição detalhada dos protocolos básicos que compõem a alma deste trabalho, sendo esses IP, TCP e UDP, para que possa ser melhor entendido e discutido posteriormente.

4.1 PROTOCOLO IP

A arquitetura *internet* especifica um conjunto de protocolos para executar a transferência de dados: IP, TCP, UDP e ICMP.

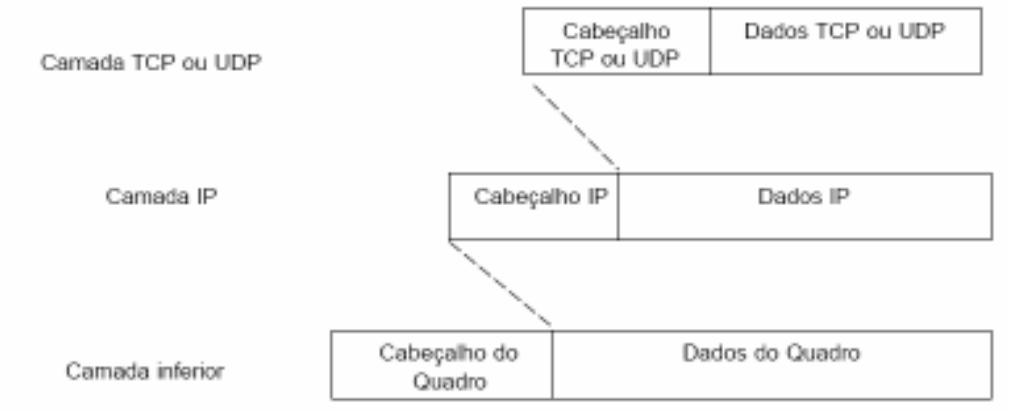

Figura 08 - Os cabeçalhos IP e TCP dentro do pacote

O protocolo IP atua a nível de camada 3 (rede), ele recebe dados da camada de transporte (TCP ou UDP) conforme *Figura 08* e é o responsável pela transferência dos datagramas da sub-rede de origem a sub-rede destino. Entre suas principais funções estão:

Fragmentação

Durante o percurso de um datagrama, ele trafega por diversos tipos de rede e como cada tecnologia tem um tamanho de bloco diferente, ou seja, pode passar por redes em que o tamanho máximo que um datagrama (medido em bytes) pode ser menor que o da rede anterior. Para resolver este problema, o datagrama será fragmentado em datagramas menores e independentes que serão enviados ao destino.

Ao receber o primeiro fragmento, a estação inicia uma temporização para aguardar o conjunto completo de fragmentos; se algum faltar, o datagrama é descartado. Assim, o processo de fragmentação provoca uma perda de eficiência devido a preservação dos fragmentos até a estação destinatária e devido ao aumento de retransmissões nos casos de perda de fragmentos, quando então, o datagrama completo é descartado.

Além destes problemas, o mecanismo de fragmentação possibilita a existência de ataques.

Mapeamento de endereços físicos em endereço *internet* e vice-versa

O endereço IP permite o roteamento das mensagens entre sub-redes, sendo as mensagens encaminhadas de *gateway* a *gateway* até seu destino. Porém, entre os *gateways*, as mensagens são transportadas pelas sub-redes físicas locais. Por este motivo, é necessário se obter o endereço físico na sub-rede local correspondente ao endereço IP. Este problema é chamado de resolução de endereços. O TCP/IP possui dois protocolos específicos para a questão da resolução de endereços: os protocolos ARP e RARP (*Reverse Address Resolution Protocol*).

Roteamento

Roteamento é o processo de escolha do caminho ao qual a mensagem (que pode ter sido dividida em vários pacotes) é enviado ao seu destino. Basicamente existem 2 tipos de roteamento: o direto (onde as estações origem e destino estão na mesma sub-rede) e o indireto (onde o destinatário não está conectado fisicamente a mesma sub-rede da estação origem, necessitando assim, que o pacote seja enviado a um *gateway*).

Todas estações e *gateways* (máquinas que interligam duas ou mais sub-redes) possuem uma tabela de roteamento, constituída basicamente dos seguintes dados: endereço IP (destino) e endereço IP do próximo *gateway* (no caso de roteamento indireto).

A tabela de roteamento do IP contém apenas endereços dos *gateways* que podem ser atingidos diretamente, ou seja, todos os *gateways* listados na tabela de roteamento estão conectados diretamente a sub-rede deste *gateway*.

Todo o roteamento em uma rede *internet* é baseado em endereços IP e não em endereços físicos das sub-redes conectadas. Além disso, as tabelas de roteamento mapeiam endereços IP das estações de destino em endereços IP de *gateways* que fazem parte da rota para estas estações.

A utilização de endereços IP cria uma carga de mapeamento de endereços na rede, visto que, para enviar um datagrama ao próximo *gateway*, a camada IP deve traduzir o endereço IP contido na tabela de roteamento no endereço físico do *gateway*. Esta é a filosofia do protocolo IP, criar uma abstração, de forma a esconder dos usuários detalhes das redes físicas que compõem a *internet*.

A tabela de roteamento armazenada em uma estação contém uma configuração inicial e permanece estática por um longo período de tempo. Esta tabela é atualizada sempre que ela recebe uma mensagem ICMP do tipo redireção vinda de um *gateway*. Isto causa uma fragilidade de segurança, pois esta mensagem ICMP não é validada em momento algum.

O RIP (*Routing Information Protocol*) é o protocolo usado na *Internet* para o roteamento dos datagramas e funciona da seguinte forma: as máquinas são divididas em ativas e passivas, as primeiras sendo as responsáveis por divulgar as informações de roteamento para as outras, enquanto as passivas recebem as informações e atualizam suas rotas, sem divulgá-las. Geralmente os *gateways* executam o RIP no modo ativo e as estações no modo passivo.

Um *gateway* difunde mensagens a cada 30 segundos ou então quando recebe uma solicitação de outro *gateway*. Cada mensagem difundida pelo *gateway* consiste em pares de informações, composto de endereço IP da sub-rede e da distância do *gateway* da outra sub-rede. A distância é medida em número de hops (número de *gateways*) na melhor rota. Para compensar diferentes tecnologias de redes, algumas implementações do RIP informam uma distância maior quando a rota atravessa uma rede lenta.

Os participantes ativos e passivos do RIP, quando recebem uma mensagem, atualizam suas rotas quando a nova mensagem trouxe a informação de uma nova rota onde a distância é menor. Para casos em que a distância é a mesma, a rota atual não é substituída e nem a nova é colocada na tabela de roteamento, isto para evitar que uma rota oscile entre dois ou mais caminhos. O RIP determina um valor de distância máximo possível, geralmente este é um valor relativamente baixo. O valor de distância máximo é utilizado para indicar que uma rede está inacessível.

Formato dos datagramas

O formato do datagrama IP é apresentado na *Figura 09* e a seguir estão detalhadas as funções dos vários campos, assim como os valores que eles podem assumir.

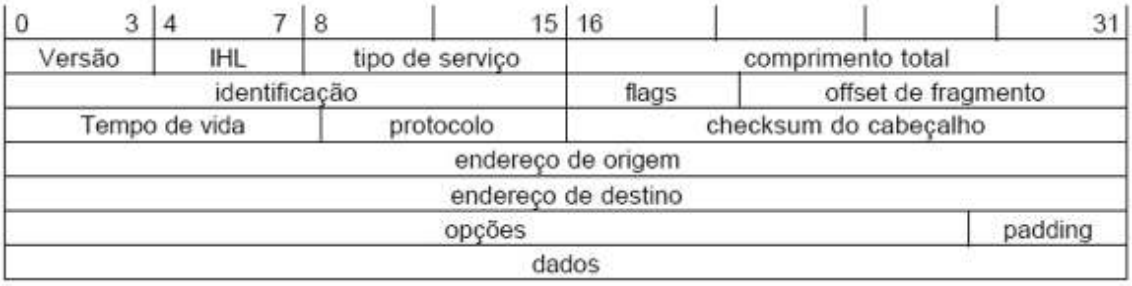

Figura 09 - Formato do Cabeçalho IP

- Versão (4 bits) – indica a versão do protocolo IP que está sendo usada (valor 4 para o IPv4), o que determina o formato do cabeçalho IP.
- IHL (*Internet Header Length*) – comprimento do cabeçalho (4 bits) – fornece o comprimento do cabeçalho em número de palavras de 32 bits, indicando assim o início do campo de dados, o valor mínimo válido para o comprimento do cabeçalho é de 5 palavras.
- Tipo de serviço (8 bits) – fornece uma indicação dos parâmetros da qualidade de serviço desejada.
- Comprimento total (16 bits) – fornece o comprimento do datagrama, medido em bytes, incluindo o cabeçalho e a parte de dados, o comprimento máximo é de 65.535 bytes.
- Identificação (16 bits) – é usado na montagem dos fragmentos de um datagrama.
- *Flags* (3 bits) – serve para controle de fragmentação, indicando se um datagrama pode ou não ser fragmentado e se houve fragmentação.
- *Offset* de fragmento (13 bits) – indica o posicionamento do fragmento dentro do datagrama original. Este posicionamento é medido em unidades de 8 bytes, e vale zero para datagramas não fragmentados e no primeiro fragmento de um datagrama.
- Tempo de vida (8 bits) – indica o tempo máximo que um datagrama pode trafegar na rede *Internet*, sendo este campo decrementado em cada *gateway*, e sendo descartado quando o seu conteúdo chega a zero.
- Protocolo (8 bits) – indica o protocolo usuário do IP, cujos dados são transportados no campo de dados, por exemplo o TCP.
- *Checksum* do cabeçalho (16 bits) – serve para identificar os erros ocorridos durante a transmissão ou na atualização do cabeçalho; desta forma o checksum é recalculado e verificado em cada ponto onde o cabeçalho é processado.

Os próximos campos são os endereços IP do emissor (32 bits) e do destinatário (32 bits).

- Opções – possui tamanho variável, podendo conter nenhuma ou diversas opções. O campo é divido em duas partes, uma indicando a classe da opção e outra o número da opção. As classes podem ser de controle, de indicação de erros e de medição ou testes, dentro de cada classe há os números de opção que identificam as funções auxiliares disponíveis.
- *Padding* – possui tamanho variável, é usado para garantir que o comprimento do cabeçalho do datagrama seja sempre um múltiplo de 32 bits.

4.2 PROTOCOLO TCP

O protocolo TCP reside na camada imediatamente acima do protocolo IP, ou seja, aos dados enviados no segmento TCP é adicionado o cabeçalho IP e enviado a sub-rede de comunicação.

O protocolo TCP que atua a nível de camada 4 (transporte), é responsável por oferecer um serviço confiável de transferência de dados, para isto são implementados mecanismos de recuperação de dados perdidos, danificados ou recebidos fora de seqüência e mecanismos para minimizar o atraso na transmissão de dados. Entre suas principais funções estão:

Multiplexação

O TCP provê um conjunto de portas (endereços) dentro de cada estação para permitir que vários processos dentro de uma mesma estação utilizem simultaneamente os serviços de comunicação. A associação de portas aos processos é tratada independentemente em cada estação. Estas portas concatenadas com o endereços IP da estação constituem um *socket*, cujo número é único em uma rede *internet*.

Os serviços TCP são fornecidos através de uma conexão lógica entre um par de *sockets*.

Gerenciamento da Conexão

É responsabilidade do TCP gerenciar toda a conexão, desde o estabelecimento desta (sincronização dos dois lados da conexão), transferência dos dados e encerramento da

conexão com liberação dos recursos alocados, para permitir que esses recursos venham a ser utilizados por outros usuários.

Uma conexão entre dois usuários pode ser estabelecida se:

- Não existir, ainda nenhuma conexão entre os *sockets* associados a esses usuários. Um determinado *socket* pode pertencer simultaneamente a mais de uma conexão, mas não pode existir mais de uma conexão envolvendo o mesmo par de *sockets*;
- Os recursos internos do TCP forem suficientes;
- Ambos os usuários concordarem com a conexão.

Transferência de dados

O TCP é capaz de transferir um stream contínuo de bytes em cada direção entre seus usuários, agrupando um certo número de bytes para formar os segmentos (modo stream). O TCP também permite que os usuários utilizem serviço orientado a registros (modo letter).

No modo stream, o próprio TCP decide quando segmentar e enviar os dados, de acordo com sua conveniência.

O serviço de transporte de dados oferece facilidades referentes a:

- Transferência full-duplex – os usuários podem transmitir dados, simultaneamente em ambas as direções de uma conexão.
- Temporização na entrega – o usuário pode especificar um prazo máximo (timeout) para a entrega de dados. Caso o TCP não consiga, ele notifica o usuário da falha do serviço e termina abruptamente a conexão.
- Transferência ordenada – O TCP garante que o stream de dados fornecidos pelo usuário de origem seja entregue na mesma ordem ao usuário de destino, ainda que tenha ocorrido segmentação durante a transmissão.
- Rotulação – o TCP estabelece uma conexão se os níveis de segurança fornecidos pelos dois usuários coincidirem. Quanto aos níveis de precedência, se eles não coincidirem, o nível mais alto solicitado é associado a conexão.
- Controle de fluxo – O TCP regula o fluxo de dados para prevenir congestionamento interno que poderia levar a degradação e falha do serviço de transporte de dados.
- Verificação de erros – O TCP utiliza o algoritmo de soma verificadora (checksum) para garantir dados livres de erro, dentro das possibilidades deste algoritmo.

Capacidades Especiais

O TCP suporta duas capacidades especiais: Data Stream Push, que possibilita ao usuário obrigar o TCP a enviar todos os dados acumulados para envio até então, sem a necessidade de completar o segmento com o tamanho determinado pela TCP e sinalização de dados urgentes, que possibilita informar ao usuário destino que chegaram dados urgentes.

Relatório de Erros

O TCP relata falhas de serviço originadas em situações de funcionamento anormal no ambiente da rede, falhas que ele não conseguiu recuperar como, por exemplo, a queda de enlaces físicos.

Segmento TCP

A unidade de transferência de dados trocada entre estações usando o protocolo TCP é chamada de segmento. Estes segmentos são trocados para estabelecimento de conexão, para transferência de dados, para enviar confirmações de recebimento (ACK), para alterar o tamanho de uma janela e para encerrar uma conexão. O segmento é dividido em duas partes: cabeçalho TCP e bloco de dados como mostra a *Figura 10*. O cabeçalho carrega a sua identificação e informações de controle.

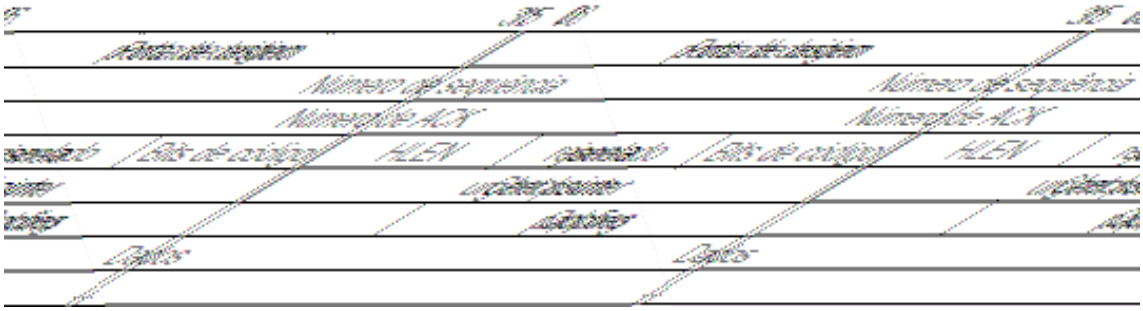

Figura 10 - Formato do Segmento TCP

- Porta de origem e porta de destino – contêm os números da portas TCP associadas aos programas de aplicação em cada ponto da conexão.
- Número de seqüência – identifica o número de seqüência deste segmento.
- Número de ACK (acknowledgement number) – identifica o número do próximo segmento que a origem espera receber. Importante observar que o número de seqüência faz o controle do fluxo de dados na transmissão enquanto o número de ACK faz o controle na direção oposta, ou seja, de recepção.

- HLEN (Header Length) – contém um inteiro que especifica o tamanho do cabeçalho do segmento medido em múltiplos de 32 bits, isto é necessário porque o campo opções tem tamanho variável, dependendo de quais opções estão incluídas. Desse modo, o tamanho do cabeçalho TCP varia conforme as opções selecionadas.
- Reservado de 6 bits – é deixado para uso futuro.
- Bits de código (code bits) – para determinar o propósito e o conteúdo do segmento, por exemplo, iniciar ou encerrar uma conexão, etc. Os 6 bits mostram como interpretar os outros campos no cabeçalho de acordo com a tabela abaixo:

Bit (da esquerda para a direita)	Significado (se o bit estiver ligado)
URG	Campo urgent pointer é válido.
ACK	Campo número de ACK é válido
PSH	Este segmento solicita um push.
RST	Ocorre uma reinicialização da conexão
SYN	Sincroniza os números da sequência
FIN	O transmissor deseja terminar a conexão

Figura 11 – Tabela de possíveis valores para o campo bits de código do TCP

Através do campo janela o TCP especifica a quantidade de dados que ele está apto a aceitar.

- O *checksum* é baseado na soma em complemento de um pseudo-cabeçalho, o cabeçalho TCP e dos dados. Este pseudo-cabeçalho é constituído de IP de origem e destino, a identificação do protocolo (no caso, TCP) e do comprimento do segmento TCP. Este pseudo-cabeçalho não é transmitido com o segmento e é somente usado para efeito de cálculos do checksum. A sua utilização tem como objetivo detectar erros no roteamento de segmentos.
- *Urgent pointer* – é usado para identificar dados urgentes.
- Opções – utilizado pelo TCP para negociar com a outra ponta da conexão, o tamanho máximo do segmento (MSS - Maximum Size Segment).
- *Padding* – é usado para garantir que o cabeçalho TCP seja múltiplo de 32 bits.

Início de uma Conexão TCP

O protocolo TCP utiliza um esquema de três pacotes para estabelecer uma conexão, este esquema é chamado de *Tree Way Handshake*, a seguir a descrição do esquema:

1. A máquina cliente envia um pacote para a máquina servidora com um flag de SYN para uma porta válida. Este flag indica que a máquina cliente deseja estabelecer uma conexão.

2. Na máquina servidora, o TCP aloca uma parte da memória para armazenar as estruturas de dados relacionadas a conexão. Como existe um limite de memória que pode ser alocado, consequentemente existe também um número limitado de conexões que podem ser estabelecidas.

O *backlog,* nome dado a pilha onde são armazenadas as conexões que ainda não foram completadas, quando ficar cheio, faz com que o TCP simplesmente comece a descartar as novas solicitações de abertura de conexão, sem emitir mensagens de erro. O TCP só volta a aceitar novas conexões quando alguma das conexões pendentes for finalizada.

O tamanho da pilha de *backlog* varia conforme as implementações, mas geralmente varia entre 5 a 32 conexões, conforme o sistema operacional. Este é um número razoavelmente pequeno, mas que se justifica, pois o tempo de abertura de uma conexão geralmente é muito curto.

A máquina servidora responde com um pacote contendo os flags de SYN e ACK. Isto significa que ela aceitou o pedido de conexão e está aguardando uma confirmação da máquina cliente para marcar a conexão como estabelecida.

3. A máquina cliente, ao receber o pacote com SYN e ACK, responde com um pacote contendo apenas o flag de ACK. Isto indica para a máquina servidora que a conexão foi estabelecida com sucesso.

Todos os pedidos de abertura de conexões recebidas por um servidor ficam armazenados em uma fila especial, que tem um tamanho pré-determinado e dependente do sistema operacional, até que o servidor receba a comunicação da máquina cliente de que a conexão está estabelecida. Caso o servidor receba um pacote de pedido de conexão e a fila de conexões em andamento estiver cheia, este pacote é descartado.

4.3 PROTOCOLO UDP

O protocolo UDP também atua na camada 4, difere, portanto do TCP, no fato de prover um serviço não orientado a conexão, não utilizando mecanismos de reconhecimento para assegurar que as mensagens transmitidas cheguem ao seu destino, não ordena as mensagens que chegam e não provê meios para controlar a taxa com que as informações fluem entre as máquinas.

Desta forma, pode-se perder, duplicar ou receber dados fora de ordem. Com isto, a aplicação é quem deve resolver este problema da falta de confiabilidade na transferência de dados.

O protocolo UDP também trabalha com o conceito de portas, a fim de possibilitar multiplexação. Cada mensagem UDP contém uma porta destino e uma porta origem, tornando possível ao UDP no destino entregar a mensagem ao processo correto e a este processo receptor encaminhar uma resposta ao processo emissor da mensagem.

A *Figura 12* mostra o datagrama UDP, o qual, da mesma forma que o *stream* TCP, é composto de duas partes: cabeçalho e campo de dados.

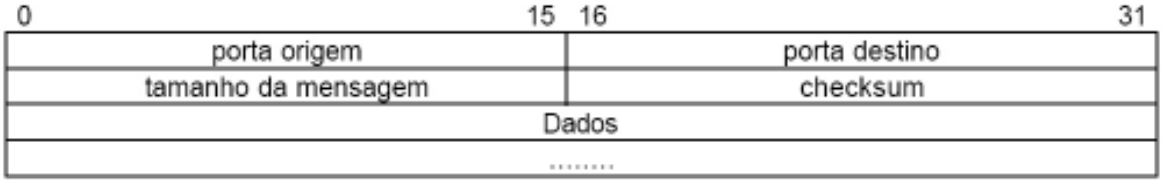

Figura 12 - Formato do datagrama UDP

- Porta de Origem e Porta Destino – Identificam as portas de serviço UDP utilizadas na comunicação. A porta de origem é opcional, quando usado, especifica a porta para onde devem ser enviadas as respostas, senão deve ficar zerada.
- Tamanho da Mensagem – Contém o número total de bytes do datagrama, incluindo cabeçalho e dados. Desta forma, o valor mínimo para este campo é de 8 bytes (tamanho do cabeçalho).
- *Checksum* – Campo opcional. Conterá valor zero se não estiver sendo usado. Importante observar que, no IP, o campo checksum é calculado apenas sobre o cabeçalho, então se o UDP não utilizá-lo, nenhuma verificação estará sendo feita em cima dos dados do usuário. Para calcular o checksum, o UDP adiciona um pseudo-cabeçalho, o qual não é transmitido junto com o cabeçalho.

A camada IP somente é responsável por transferir dados entre um par de estações, enquanto a camada UDP é somente responsável por diferenciar entre múltiplas fontes ou destinatários dentro de uma estação. Resumindo, o cabeçalho IP é responsável por identificar as estações de origem e destino e o UDP é responsável por identificar as portas de origem e destino dentro de uma estação. [STR1999]

Problema do protocolo UDP

Para usar um serviço UDP, a máquina cliente inicialmente escolhe um número de porta, que é variável cada vez que o serviço for utilizado e envia um pacote para a porta da máquina servidora correspondente ao serviço (esta porta na máquina servidora é fixa). A máquina servidora, ao receber a requisição, responde com um ou mais pacotes para a porta da máquina cliente. O problema é que o protocolo UDP é um protocolo não orientado à conexão. Isto significa que se um determinado pacote for observado isoladamente, fora de um contexto, não se pode saber se ele é uma requisição ou uma resposta de um serviço.

Nos filtros de pacotes tradicionais, como o administrador não pode saber de antemão qual porta será escolhida pela máquina cliente para acessar um determinado serviço, ele pode ou bloquear todo o tráfego UDP ou permitir a passagem de pacotes para todas as possíveis portas.

4.4 PROTOCOLO ICMP

O protocolo ICMP é usado no transporte de mensagens de erro e de controle entre *gateways* e estações, utiliza o IP para o transporte das mensagens, não oferecendo portanto garantia de entrega. O ICMP é considerado integrante da camada de rede apesar de, na verdade, utilizar os serviços do protocolo IP.

Na *Figura 13* temos o formato do cabeçalho ICMP dentro de um pacote.

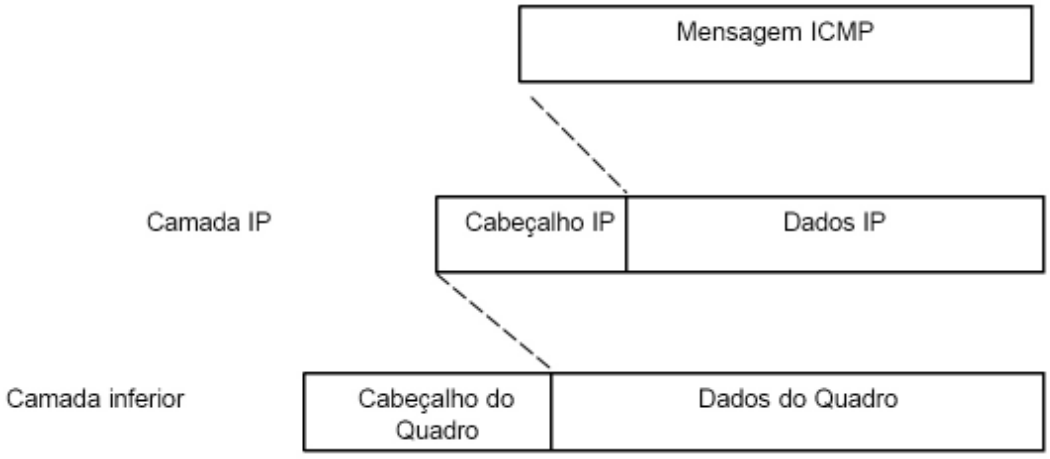

Figura 13 - Cabeçalho ICMP dentro do pacote

Na *Figura 14* temos o formato básico de uma mensagem ICMP.

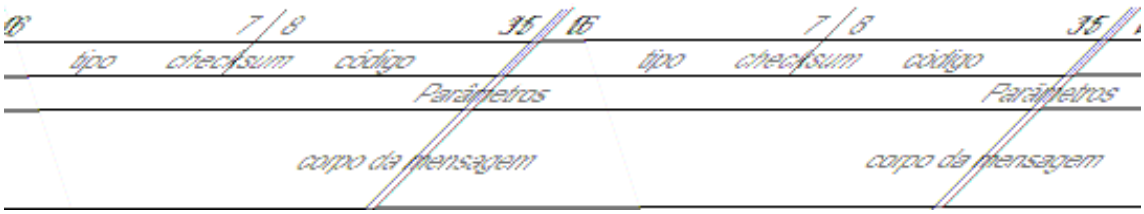

Figura 14 - Formato da mensagem ICMP

- Tipo – Identifica uma mensagem ICMP particular, conforme a tabela a seguir.
- Código – Usado na especificação dos parâmetros da mensagem.
- Checksum – Corresponde ao verificador de erro, calculado a partir da mensagem ICMP completa.
- Parâmetros – Usado na especificação de parâmetros mais longos que os 8 bits do campo código e seu formato e tamanho variam de acordo com o tipo de mensagem.

Nos casos em que a mensagem ICMP se referir a um datagrama anteriormente enviado, ela transporta também o cabeçalho completo daquele datagrama IP, para facilitar ao emissor a identificação do datagrama problemático.

Na tabela abaixo temos as mensagens de controle do ICMP. [STR1999]

Mensagem	Tipo	Descrição
Destinatário inacessível	3	Utilizada quando o datagrama não pode ser entregue ao destinatário especificado. A causa pode ir desde um *gateway* sem informação necessária para dar continuidade ao encaminhamento do datagrama até uma porta na estação destino sem possibilidade de acesso.
Ajuste de fonte	4	Mensagem gerada por *gateways* ou estações quando eles precisam reduzir a taxa de envio de datagramas do emissor.
Redireção	5	Mensagem gerada pelo *gateway* para notificar uma estação sobre uma rota mais adequada ao destinatário do datagrama por ele enviado. Desta forma as estações atualizam a suas tabelas de roteamento.
Resposta ao eco	0	Mensagens usadas para testar se a comunicação entre duas entidades é possível. O destinatário é obrigado a responder.
Eco	8	
Tempo excedido	11	Retornada por um *gateway* quando o tempo de um datagrama expira. Retornada por uma estação quando o tempo para remontagem de uma mensagem segmentada expira.
Problema de Parâmetro	12	Relata a ocorrência de erros de sintaxe semântica no cabeçalho do datagrama IP.

Marca de tempo	13	São usadas para medir as características de atraso no transporte de datagramas.
Resposta a marca de tempo	14	
Solicitação de informações	15	Usada por estações para recuperar o endereço da sub-rede a qual está conectada.
Resposta de informações	16	
Solicitação de máscara de endereço	17	Usada pelas estações para recuperar a máscara de endereços. O *gateway* é responsável por responder com a descrição da máscara.
Resposta de máscara de endereço	18	
IPv6 Where-Are-You	33	Para comunicação de hosts/*gateways* IPv6
IPv6 I-Am-Here	34	
Solicitação de Nome do domínio	37	Usada por estações para recuperar o nome do domínio ao qual está conectada.
Resposta de Nome de domínio	38	

Figura 15 –Tabela de Mensagens de controle do ICMP

Capítulo 5

Descreve de forma detalhada como foi o desenvolvimento do projeto, as dificuldades encontradas, as descobertas obtidas e caminhos percorridos no decorrer do desenvolvimento, em suma, aspectos de implementação em geral.

---

5 DESENVOLVIMENTO DO NIDS HYDRA

O NIDS HYDRA foi desenvolvido em uma plataforma O*penSource*, com o objetivo de verificar os pacotes que trafegam na rede (a princípio TCP e UDP) e baseado em algumas assinaturas (regras que definem o padrão de algumas formas de ataque) já conhecidas tomar uma decisão caso o mesmo seja detectado, assim como estar efetuando logs dos mesmos para uma análise posterior. Na versão inicial, o mesmo grava os dados de tentativa de ataque em Banco de Dados MySQL (SGDB atualmente free), onde pode ser consultado via WEB por uma interface amigável desenvolvida em PHP (linguagem interpretada que funciona como extensão do protocolo HTML e roda no próprio Web Server).

Quanto a plataforma de desenvolvimento, foi utilizado um ambiente Linux (RedHat Fedora Core 2), sendo utilizado como editor o vi e como compilador o gcc.

5.1 AMBIENTE DE DESENVOLVIMENTO

Como citado na descrição, foi utilizado como plataforma de desenvolvimento o editor de textos vi (padrão em ambiente *NIX), o compilador gcc, o *debugger* gdb (GNU *Debugger*) e o *front end* ddd (para utilização visual do gdb) e, quando não desenvolvido diretamente no ambiente Linux, foi utilizado o utilitário Putty (terminal *free* utilizado em plataforma MS Windows®) conforme a *Figura 16*.

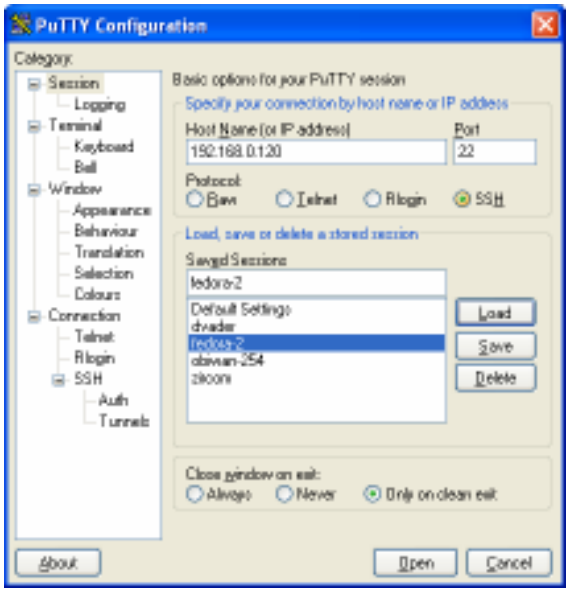

Figura 16 – Tela Principal do PuTTY

Conforme a *Figura 17*, é apresentado a ferramenta de *debbuger* ddd (*front end* para utilização do gdb), a fim de prover uma minimização em relação ao tempo de teste da aplicação.

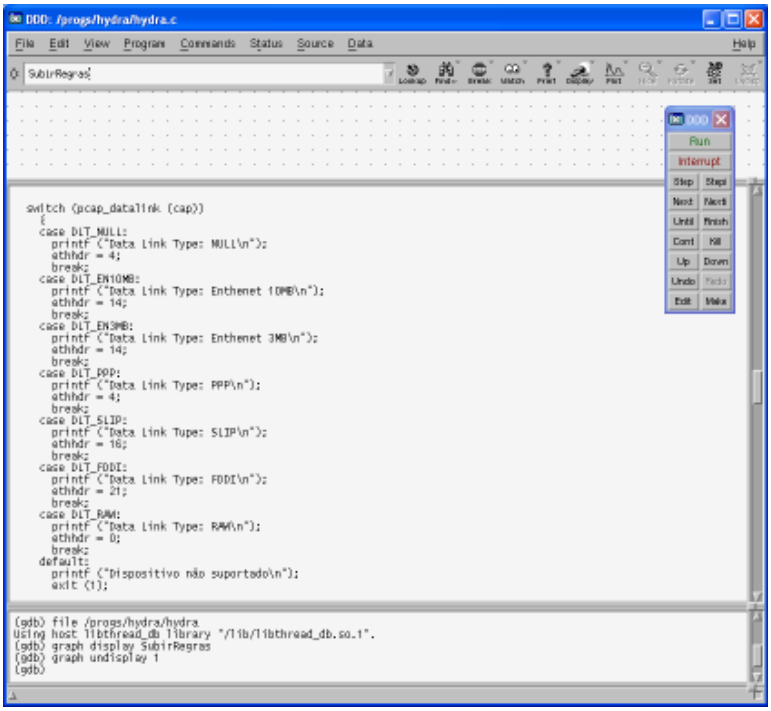

Figura 17 – Tela Principal do DDD

Foi testado também a ferramenta ECLIPSE, com objetivo de se ganhar tempo no desenvolvimento (ferramenta colaborativa para produtividade) onde foi possível notar uma grande necessidade de *Hardware* (memória e processador) para utilização do mesmo, uma

vez que a ferramenta trabalha sobre plataforma Java. A mesma foi desconsiderada devido a questão de desempenho computacional, assim como a quantidade excessiva de “lixo” que tal ferramenta adiciona ao conteúdo do código.

Capítulo 6

Descreve o funcionamento do Sistema de Detecção HYDRA.

6 ESQUEMA DE FUNCIONAMENTO

É descrito neste item todo o processo seguido pelo Sistema de Detecção HYDRA, passando pelos pontos de captura de pacotes, análise das regras à ação a ser tomada.

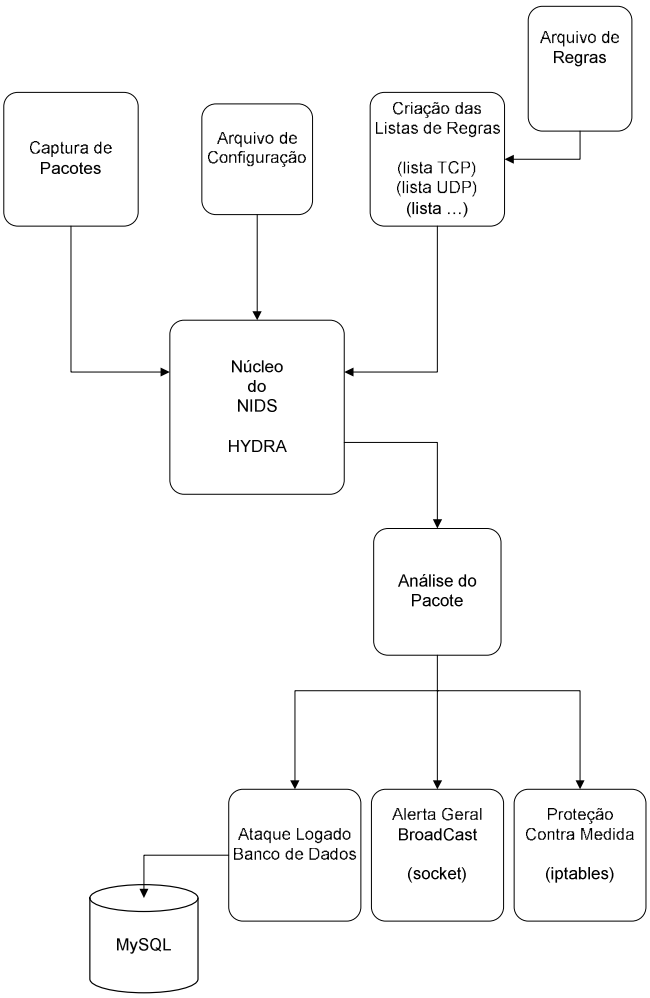

Figura 18 – Processos do NIDS HYDRA

O processo utilizado no sistema HYDRA, apresentado na *Figura 18* é explicado detalhadamente a seguir. O Sistema é carregado em memória, em tempo de carga, é verificado um arquivo de configuração (tal arquivo tem como finalidade repassar ao sistema informações como SERVIDOR de banco de dados, usuário, senha e data base). Em seguida, é lido o arquivo de regras (arquivo este que contém as regras pré-definidas de acordo com a

necessidade do ambiente, lembrando que cada regra forma uma assinatura única), sendo criado uma lista dinamicamente para cada tipo de protocolo que o sistema suporte (em nosso caso TCP e UDP). O arquivo de regras é alimentado com os dados de cada uma das regras. Após a leitura de todas as regras, o sistema define a interface do SERVIDOR NIDS para o modo promíscuo e começa a capturar TODOS os pacotes que passarem pela mesma. O sistema envia os pacotes capiturados para o módulo de Análise. Tal módulo tem como função comparar o pacote em questão com as regras pré-definidas e carregadas em memória, sendo que o mesmo faz uma busca específica. Caso o pacote seja TCP, o mesmo buscará somente na lista de pacotes TCP. Caso seja UDP, ocorrerá o mesmo, mas somente na lista UDP. Se a análise for satisfatória, ou seja, o pacote capturado e testado corresponde a uma das regras contidas na lista de regras, o sistema ativa três módulos, o módulo de Banco de Dados, (tal módulo, o módulo de Alerta Geral, e o módulo de Proteção. O módulo de Banco de Dados tem como função armazenar a tentativa de ataque em Banco de Dados com a finalidade de visualização por uma view mais amigável, assim como consultas posteriores e relatórios. O módulo de Alerta Geral tem como finalidade enviar um *broadcast* para todos os clientes da rede, ou seja, envia uma notificação informando que há um registro de tentativa de ataque e já informa uma prevenção para o mesmo "comando formado para evitar ataques vindo do IP origem". O módulo de Proteção tem como função aplicar efetivamente a proteção, ou seja, é efetuado um bloqueio para o atacante "IP originário do ataque".

Já no cliente, há um *daemon* (agente executando em *background*) que aguarda constantemente por um "alerta" vindo do SERVIDOR NIDS via *socket*, conforme *Figura 19*. Caso receba uma notificação, é implementado no sistema o comando que foi repassado pelo SERVIDOR, a fim de que a prevenção seja feita antes mesmo da tentativa de ataque ser efetivada no citado cliente.

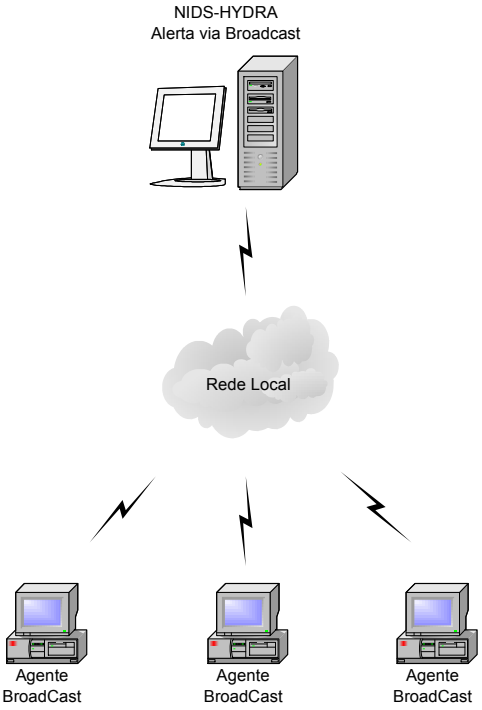

Figura 19 – Alerta Enviado do SERVIDOR NIDS Para os Clientes

6.1 CAPTURA DE PACOTES

Para um efetivo Sistema de Detecção de Intrusão de Rede (NIDS), faz necessário que todos os pacotes possam ser vistos e tratados pelo SERVIDOR. Para isso, o NIDS pode ser implementado de algumas formas básicas, sendo essas a utilização de HUB ou a utilização de Switch com suporte a *Port Mirror*. A seguir será explicado cada um dos possíveis casos.

NIDS com HUB – Conforme *Figura 20*, pode-se verificar que para qualquer pacote enviado para a rede é replicado, não somente para o SERVIDOR NIDS, mas para todos os outros equipamentos conectados a rede. Desta maneira, o Sistema funcionará a contento. Mas por outro lado, tal estrutura torna o ambiente totalmente inseguro, pois em qualquer ponto da rede é possível verificar pacotes que venham a trafegar em texto puro (*clear text*), como no serviço de FTP ou NFS por exemplo.

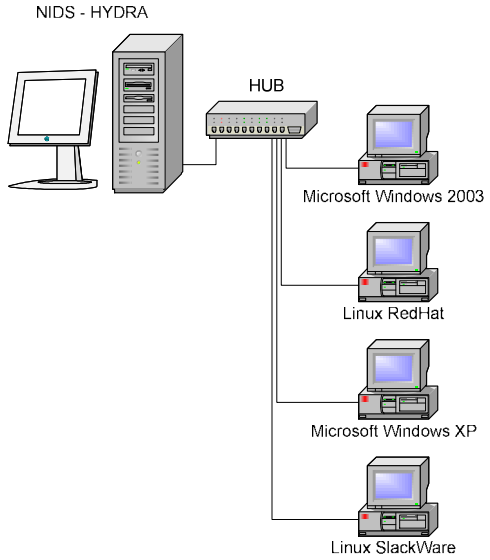

Figura 20 – NIDS Implementado em ambiente com HUB

NIDS com Switch (sem *Port Mirror*) – Conforme *Figura 21*, pode-se verificar que para qualquer pacote enviado para a rede, o mesmo não será replicado, diferente do que acontece em um ambiente com HUB, pois agora é criado uma definição de "domínio de colisão", ou seja, o Switch trabalha com uma tabela interna (em memória), onde é criado uma tabela e é definida uma referência de endereços MAC´s associadas a cada porta, com o tempo, quando um pacote é enviado, o mesmo não é mais replicado para todas as portas, mas somente para a porta onde o MAC destino consta na tabela. Sendo assim o Sistema não funcionaria a contento, pois somente os pacotes com destino ao SERVIDOR serão enxergados, ou seja, a rede permanecerá insegura.

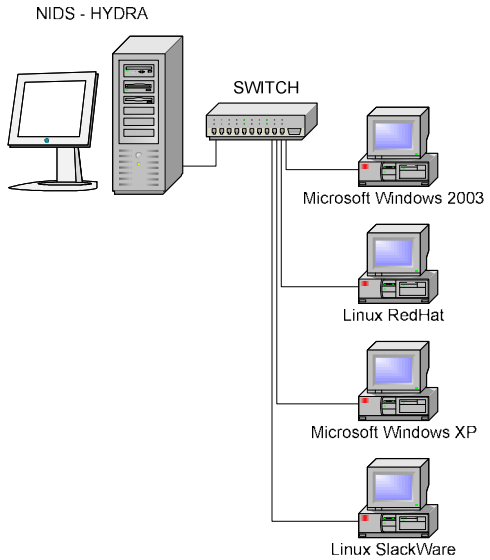

Figura 21 – NIDS Implementado em ambiente com Switch sem Port Mirror

NIDS com Switch (com *Port Mirror*) – Conforme *Figura 22*, pode-se verificar que para qualquer pacote enviado para a rede, o mesmo não será replicado, diferente do que acontece em um ambiente com o HUB, pois agora é criado uma definição de "domínio de colisão", ou seja, o Switch trabalha com uma tabela interna (memória), onde é criado uma tabela e é definida uma referencia de endereços MAC´s associadas a cada porta, com o tempo, quando um pacote é enviado, o mesmo não é mais replicado para todas as portas, mas somente para a porta onde o MAC destino consta na tabela. Vale lembrar que a definição de *Port Mirror* diz que para cada porta, ou range de portas, pode ser criado um espelho em outra porta, ou seja, todos os pacotes que passam por uma determinada porta ou grupo de portas podem ser replicados para uma porta específica, sendo que esta seria a porta de conexão com o SERVIDOR NIDS, onde o mesmo faria seu papel a contento.

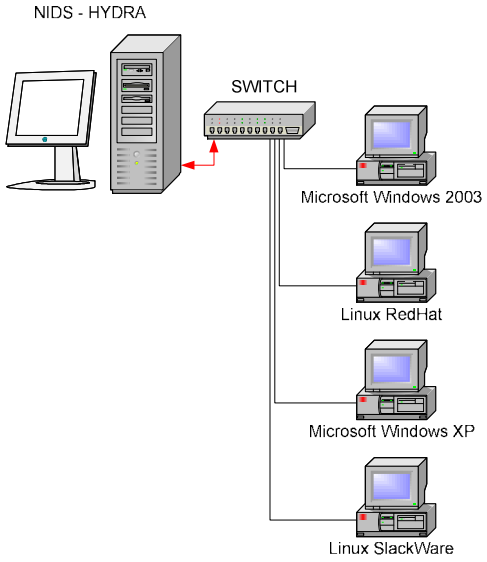

Figura 22 – NIDS Implementado em ambiente com Switch com Port Mirror

6.2 ANÁLISE DE REGRAS

Cada pacote está sendo analisado pelo NIDS "Módulo de Análise" de forma que, caso algum dos pacotes esteja de acordo com alguma das regras pré-definidas no sistema será acionado um procedimento (gravação em Banco de Dados, Proteção local e Alerta Distribuído).

Para um melhor entendimento, segue na *Figura 23 e 24* os diagrams de como ficam montados em memória as regras utilizadas pelo HYDRA tipo TCP e UDP.

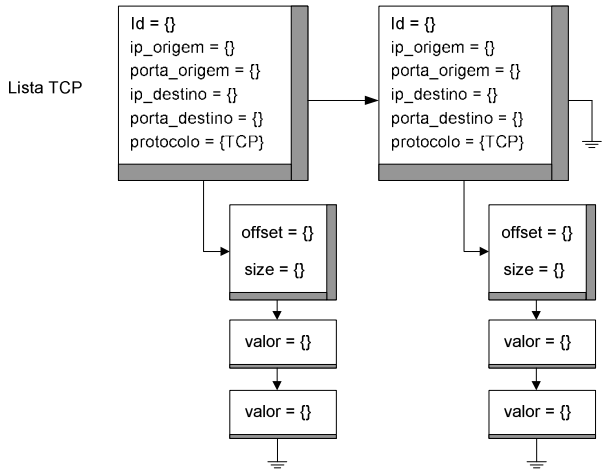

Figura 23 – Listas de Regras TCP em Memória

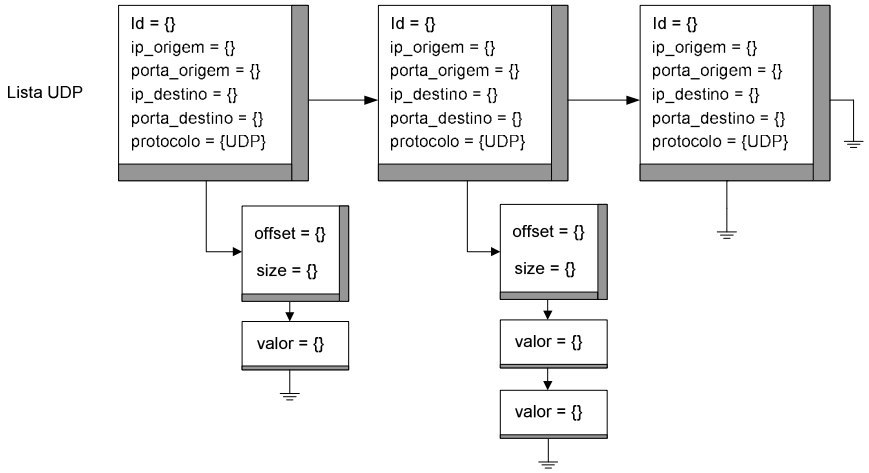

Figura 24 – Listas de Regras UDP em Memória

Como pode ser visto, para cada regra lida no arquivo de regras, é alocado uma área em memória, área esta que contém todas as definições de uma assinatura. Estas listas são utilizadas para a verificação de cada pacote a ser analisado pelo HYDRA, de acordo com o protocolo.

O arquivo de regras possui uma sintaxe específica, com os seguintes campos <id>;<ip origem>;<porta origem>;<ip destino>;<porta destino>;<tipo de protocolo>;<flags tcp>;<dados> como segue:

ID

Campo responsável por definir a identificação da regra no sistema, ou seja, uma referencia para a regra no HYDRA. Tal identificador deve ser único em todo o arquivo de regra.

Formato:

<identificador da regra>

IP origem

Identificação do endereço IP origem, caso não haja necessidade de se especificar um endereço, pode ser utilizado a palavra reservada "ALL" representando todos os IP's origem.

Formato:

<ALL> ou <endereço IP>

Porta Origem

Identificação do porta origem, caso não haja necessidade de se especificar uma porta, pode ser utilizado a palavra reservada "ALL" representando todas as portas de origem.

Formato:

<ALL> ou <porta válida>

IP destino

Identificação do endereço IP destino, caso não haja necessidade de se especificar um endereço, pode ser utilizado a palavra reservada "ALL" representando todos os IP's destino.

Formato:

<ALL> ou <endereço IP>

Porta Destino

Identificação do porta destino, caso não haja necessidade de se especificar uma porta, pode ser utilizado a palavra reservada "ALL" representando todas as portas destino.

Formato:

<ALL> ou <porta válida>

Tipo de Protocolo

Identificação do protocolo a ser verificado. O HYDRA atualmente suporta o protocolo TCP e UDP.

Formato:

<protocolo a ser testado>

Flags TCP

Identificação dos flags existentes no pacote TCP, podendo ser **F**in, **S**yn, **R**st, **P**sh, **A**ck, **U**rg, res**2**, res**1**, onde somente os caracteres em negrito devem ser informados.

Formato:

<caracter referente ao flag>

Obs.: Para regras UDP, o mesmo não precisa ser informado, basta somente fechar o campo referente a flag, com dois ponto e virgula (;;). Para regras TCP que não se deseje utilizar os flags, informar utilizando o caracter exclamação (;!;).

Dados

Identificação das palavras (hexadecimal ou string) a serem localizadas no campo de dados do pacote, informando o offset (deslocamento do ponteiro em N bytes para a correta localização do campo que se está procurando) e size (tamanho do campo que se está procurando). No campo dados, o campo a ser localizado pode ser especificado uma ou N vezes, sendo que o próximo offset a ser definido, é calculado sobre o último offset acrescido do size.

Formato:

<offset,size,{hexa}[string];>

Obs.: Caso nao seja necessário a pesquisa no campo de dados, o mesmo pode ser desabilitado usando-se o caracter exclamação (;!;).

Obs.: Todos os campos são separados por “;” assim como no campo dados, caso haja mais de um campo a ser pesquisado no campo de dados, o mesmo deve ser separado e terminado pelo caractere “;”.

Agora que a sintaxe do arquivo de regras já foi descrito, será citado alguns exemplos, onde poderá ser mostrado visualmente para melhor esclarecimento.

Como primeiro exemplo será explicado a regra referente ao RPC Portmap mountd request UDP, como segue:

Figura 25 –Regra de RPC Portmap Mountd Request UDP

Tem-se acima a regra referente ao “RPC portmap mountd request UDP”, ou seja, ao se executar o comando “**showmount –e <servidor>**” para visualização dos file systems exportados por exemplo, uma assinatura idêntica a *Figura 25* é enviada ao SERVIDOR,

baseado nisto, esta assinatura tem como objetivo verificar se um provável atacante está a levantar informação para fins de ataque.

No primeiro campo (ID) a mesma está setada para 3, no segundo campo (IP Origem) esta definido a palavra reservada ALL, ou seja, qualquer endereço IP origem, o mesmo ocorre com a Porta Origem e IP Destino. No quinto campo (Porta Destino) esta definido o valor 111, ou seja, somente será válido se o pacote for enviado ao SERVIDOR na porta destino 111, no sexto campo (Tipo de Protocolo) esta definido o protocolo UDP, ou seja, somente será válido se o pacote estiver utilizando o protocolo UDP, do sétimo campo em diante está definido a pesquisa a ser feita no campo de dados, a explicação do mesmo será dada abaixo.

Pode-se notar que existe quatro campos de pesquisa a ser feito, sendo esses (12,4,{00}{01}{86}{a0};), (4,4,{00}{00}{00}{03};), (16,4,{00}{01}{86}{a5};) e (8,4,{00}{00}{00}{00};). No primeiro campo está sendo informado que o ponteiro de byte deve ser posicionado na posição 12 (doze) a partir da base e pesquisar nos próximos quatro bytes procurando pelos valores em hexadecimal **00 01 8a a0**. Caso encontrado, a próxima pesquisa utiliza o *offset* acrescido do *size* para somar ao próximo *offset*. Por exemplo, o anterior foi 12 (doze) de *size* 4 (quatro), terminando em 16 (dezesseis). A segunda pesquisa terá como *offset* os 16 (dezesseis) do primeiro campo acrescido do 4 (quatro) da segunda pesquisa. O que significa que o ponteiro deve ser deslocado para o byte 20 (vinte) e a partir daí pesquisar nos próximos quatro bytes procurando pelos valores em hexadecimal **00 00 00 03**. Caso esta seqüência também seja encontrada, a próxima pesquisa utiliza o *offset* acrescido do *size* para somar ao próximo *offset*, ou seja, 24 (vinte e quatro) acrescido de 16 (dezesseis) da terceira pesquisa. O que significa deslocar o ponteiro agora para o byte 40 (quarenta) e a partir daí procurar pela próxima seqüência de 4 (quatro) bytes com seus valores em hexadecimal 00 01 86 a5. Caso encontrado, a próxima pesquisa utiliza o *offset* acrescido do *size* para somar ao próximo *offset*, ou seja, 44 (quarenta e quatro) acrescido de 8 (oito), deslocando o ponteiro para o byte 52 e pesquisando nos próximos quatro bytes a procura da próxima seqüência **00 00 00 00.** Caso esta seqüência também seja localizada, pode-se afirmar que o pacote está de acordo com a regra informada ao sistema.

Para essa pesquisa que parece complexa a princípio, estará sendo mostrado na *Figura 26* como a mesma fica teoricamente estruturada em memória, uma vez que para todos os pacotes lidos, assim como as regras carregadas, os valores armazenados são convertidos para decimal.

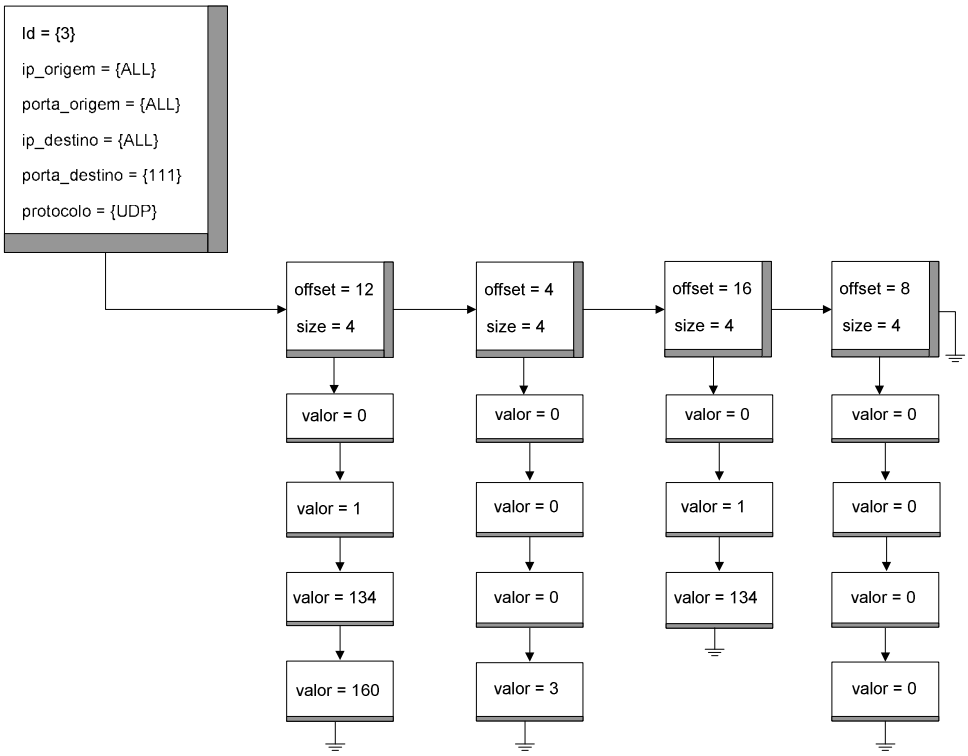

Figura 26 –Regra de RPC Portmap Mountd Request UDP em Memória Conceitualmente

Na *Figura 27* será exibido como fica realmente em memória após o carregamento da regra, utilizando-se a aplicação ddd para visualização do mesmo.

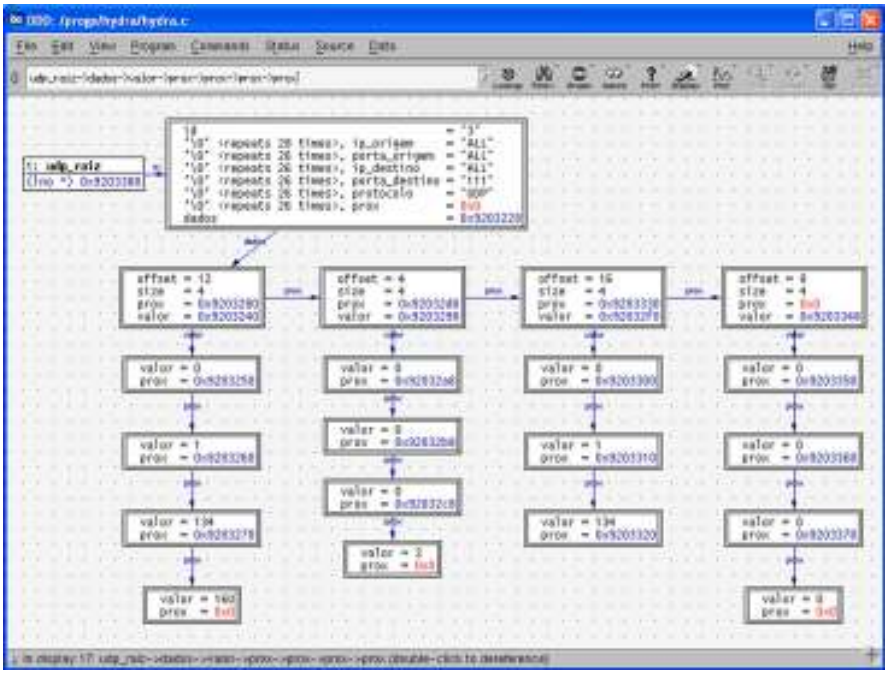

Figura 27 –Regra de RPC Portmap Mountd Request UDP em Memória Realmente

Como segundo exemplo, será explicado a regra referente ao tftp get passwd, como segue:

```
4;ALL;ALL;ALL;69;UDP;;0,2,{00}{01};2,6,[passwd];
```

Figura 28 – Regra do TFTP Get Passwd

Na *Figura 28* tem-se a regra referente ao "TFTP Get Passwd", ou seja, ao se executar o comando "**get passwd**" no prompt do *tftp client* para *download* do arquivo de senhas. Uma assinatura idêntica a *Figura 28* é enviada ao SERVIDOR, baseado nisto, esta assinatura tem como objetivo verificar se um provável tentativa de ataque.

No primeiro campo (ID) está configurado para o valor 6, no segundo campo (IP Origem) está definido a palavra reservada ALL, ou seja, qualquer endereço IP origem, o mesmo ocorre com a Porta Origem e IP Destino. No quinto campo (Porta Destino) esta definido o valor 69, ou seja, somente será válido se o pacote for enviado ao SERVIDOR na porta destino 69, no sexto campo (Tipo de Protocolo) esta definido o protocolo UDP, ou seja, somente será válido se o pacote estiver utilizando o protocolo UDP, do sétimo campo em diante está definido a pesquisa a ser feita no campo de dados, a explicação do mesmo será dada abaixo.

Pode-se notar que existem dois campos de pesquisa a ser feito, sendo esses (0,2,{00}{01};) e (2,6,[passwd];). No primeiro campo está sendo informado que o ponteiro de byte deve ser posicionado na posição 0, ou seja, a partir da base e pesquisar nos próximos dois bytes procurando pelos valores em hexadecimal **00 01**, caso encontrado, a próxima pesquisa utiliza o offset acrescido do size para somar ao próximo offset, por exemplo, o anterior foi 0 de size 2, terminando em 2, a segunda pesquisa terá como offset o 2 do primeiro campo acrescido do 2 da segunda pesquisa, o que significa que o ponteiro deve ser deslocado para o byte 4 e a partir daí pesquisar nos próximos seis bytes procurando pelos valores em string **passwd**, caso esta seqüência também seja localizada, pode-se afirmar que o pacote está de acordo com a regra informada ao sistema.

Para essa pesquisa que parece complexa a principio, envolvendo valores em hexadecimal e *strings*, estará sendo mostrado na *Figura 29* como a mesma fica teoricamente estruturada em memória, uma vez que para todos os pacotes lidos, assim como as regras carregadas, os valores armazenados são convertidos para decimal.

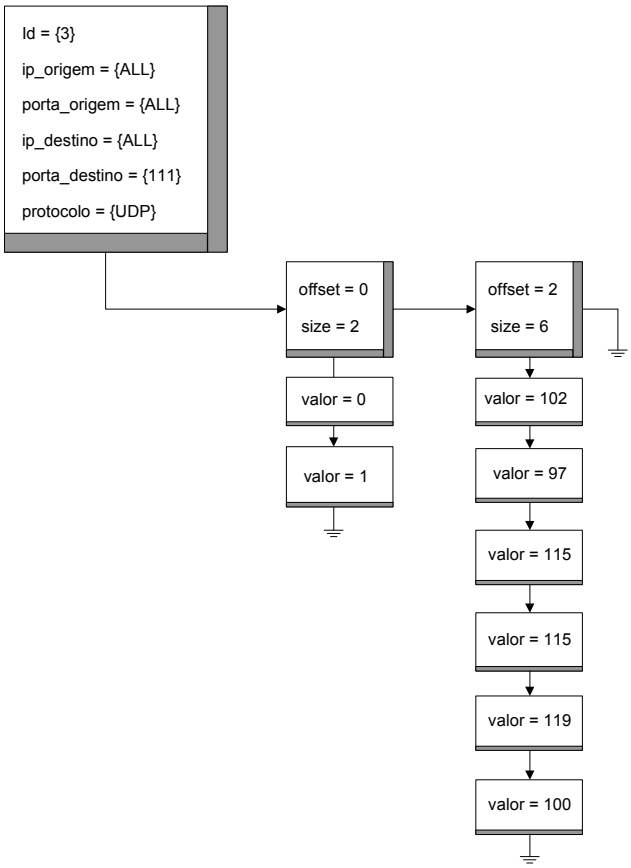

Figura 29 – Regra do TFTP Get Passwd em Memória Conceitualmente

Na *Figura 30* é exibido como fica realmente o processo em memória após o carregamento da regra, utilizando-se a aplicação ddd para visualização do mesmo.

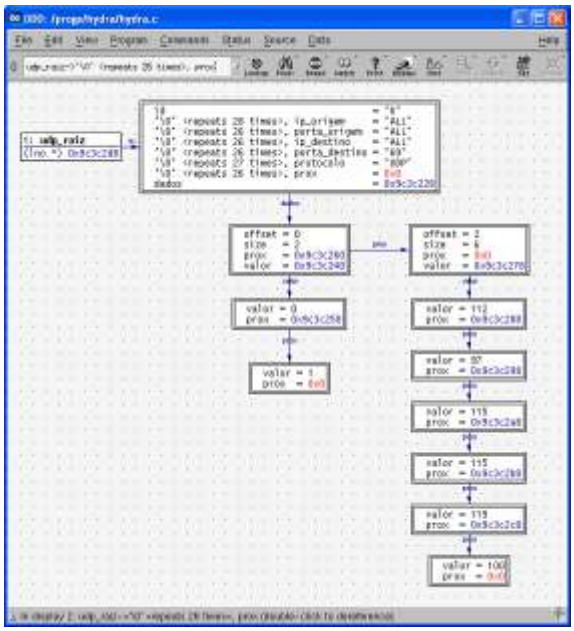

Figura 30 – Regra de TFTP Get Passwd em Memória

Conclui-se neste tópico que para a implementação do módulo de carga das regras, assim como para a implementação do módulo de análise (módulos estes vitais para o sistema NIDS HYDRA). Assim como um conhecimento mais aprofundado sobre listas e recursividade.

6.3 ARMAZENAMENTO EM BANCO DE DADOS

Quando o Sistema Hydra constata que uma assinatura foi realmente encontrada, é gerado uma rotina diferente do normal, que aciona alguns procedimentos. Um desses é o armazenamento dos dados em Banco de Dados. Foi utilizado a biblioteca do MySQL para que as chamadas ao SGDB fossem efetuadas internamente pelo sistema. Para que a configuração de SERVIDOR, Usuário, Senha e Banco de Dados não ficasse embutido no sistema. O que permite uma flexibilidade para quem for utilizá-lo. Foi implementado uma rotina para carregar tais informações a partir de um arquivo de configuração, conforme *Figura 31*, para uma área da memória (struct DADOS), que fica sempre a disposição do sistema conforme *Figura 32*. Tal medida se faz necessário uma vez que para todo novo registro a ser inserido em banco de dados, o sistema necessita de tais informações. Caso leia o arquivo, o dispositivo de I/O pode tornar-se um gargalo, diferente de quando armazenado em memória.

```
# Arquivo de Configuracao
# Defina abaixo as informacoes corretas referente
# ao banco de dados.
servidor=localhost
usuario=pf
senha=pf00
bd=PF
[root@fedora hydra]#
```

Figura 31 – Arquivo de Configuração

```
servidor = {}
usuario = {}
senha = {}
bd = {}
```

Figura 32 – Struct DADOS – Arquivo de Configuração em memória

É utilizado a função InsereBD(), onde é passado como parâmetro os dados necessários para uma identificação precisa de um ataque ou tentativa conforme *Figura 33*.

Field	Type	Null	Key	Default	Extra
cod	int(4)		PRI	NULL	auto_increment
id	varchar(10)				
ip_origem	varchar(15)	YES		NULL	
port_origem	varchar(5)	YES		NULL	
ip_destino	varchar(15)	YES		NULL	
port_destino	varchar(5)	YES		NULL	
tipo_protocol	char(3)	YES		NULL	
data	date	YES		NULL	

Figura 33 – Estrutura da Tabela Principal – db_alert

E para o armazenamento das informações adicionais da assinatura (sistemas afetados, possível impacto gerado, etc) é utilizada a tabela db_desc, conforme *Figura 34*. O relacionamento entre as tabelas se faz pelo campo id, definido no arquivo de regras, como sendo única para cada assinatura.

Field	Type	Null	Key	Default	Extra
id	int(4)			0	
sumario	varchar(70)	YES		NULL	
impacto	varchar(80)	YES		NULL	
info_detalhado	varchar(100)	YES		NULL	
cenario_ataque	varchar(100)	YES		NULL	
facilidade_ataque	varchar(80)	YES		NULL	
falso_positivo	varchar(50)	YES		NULL	
falso_negativo	varchar(50)	YES		NULL	
acao_corretiva	varchar(150)	YES		NULL	
contribuicao	varchar(50)	YES		NULL	
referencia	varchar(100)	YES		NULL	

Figura 34 – Estrutura da Tabela de Descrição – db_desc

Quanto ao suporte a tipos diferentes de tabela, foi dado preferência no sistema pelo InnoDB (mecanismo de armazenamento que suporta chaves estrangeiras, transações e lock de registros). [THO2004]

Para sua habilitação no sistema, foi alterado no arquivo de configuração do MySQL (/etc/my.cnf) uma diretriz que informa sobre a criação de uma table space (arquivo único de dados onde o mysql ira armazenar os registros) chamado de ts_pf_01 com um tamanho de 100MB, conforme *Figura 35*.

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
innodb_data_file_path = ts_pf_01:100M:autoextend

[mysql.server]
user=mysql
basedir=/var/lib

[safe_mysqld]
err-log=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

Figura 35 – Arquivo de Configuração do MySQL

6.4 VISUALIZAÇÃO VIA WEB

Atualmente o HYDRA possui uma interface via WEB, utilizada para monitorar todos os acessos "maliciosos" que ocorreram na rede administrada. Conforme *Figura 36*, pode-se verificar a tela principal do Site de monitoração, o qual pode ser acessado também via *Internet* em http://www.hydra.hpages.net

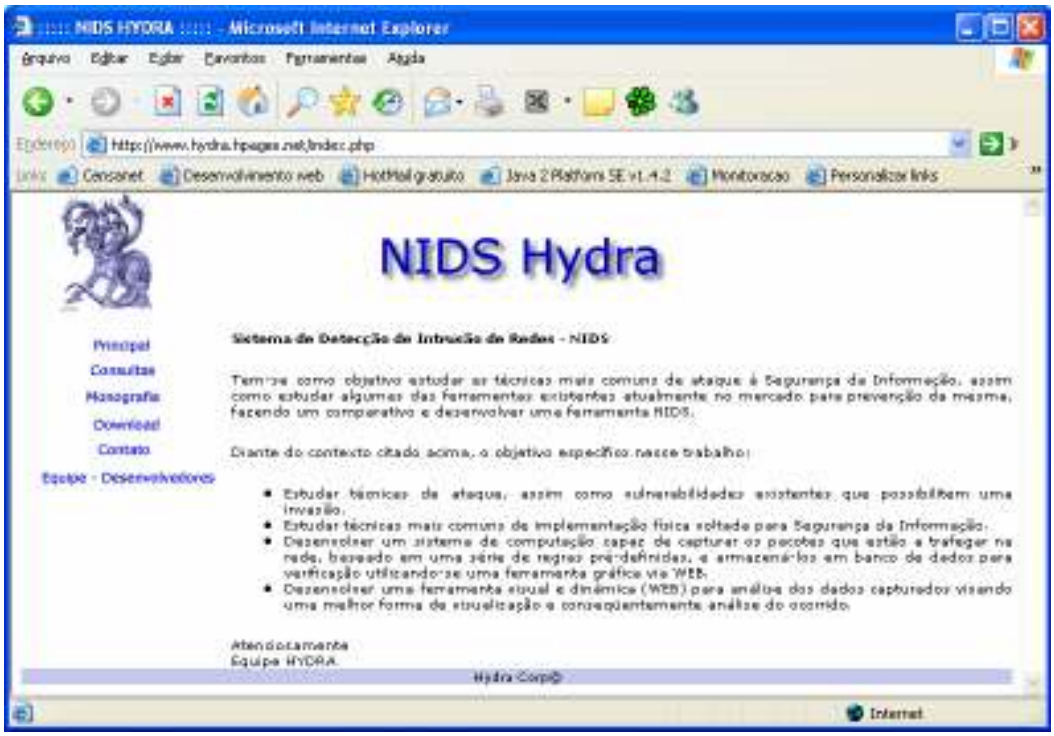

Figura 36 – Tela Principal do Site de Monitoração do HYDRA

Como existe atualmente ambientes mistos e diferentes, fica a critério de cada administrador definir qual a melhor "consulta" referente a tentativas de ataque, para que se

possa planejar e reagir de uma forma mais ágil e segura. Pensando nisso, é apresentado algumas consultas simples, como exibido na *Figura 37, 38 e 39*.

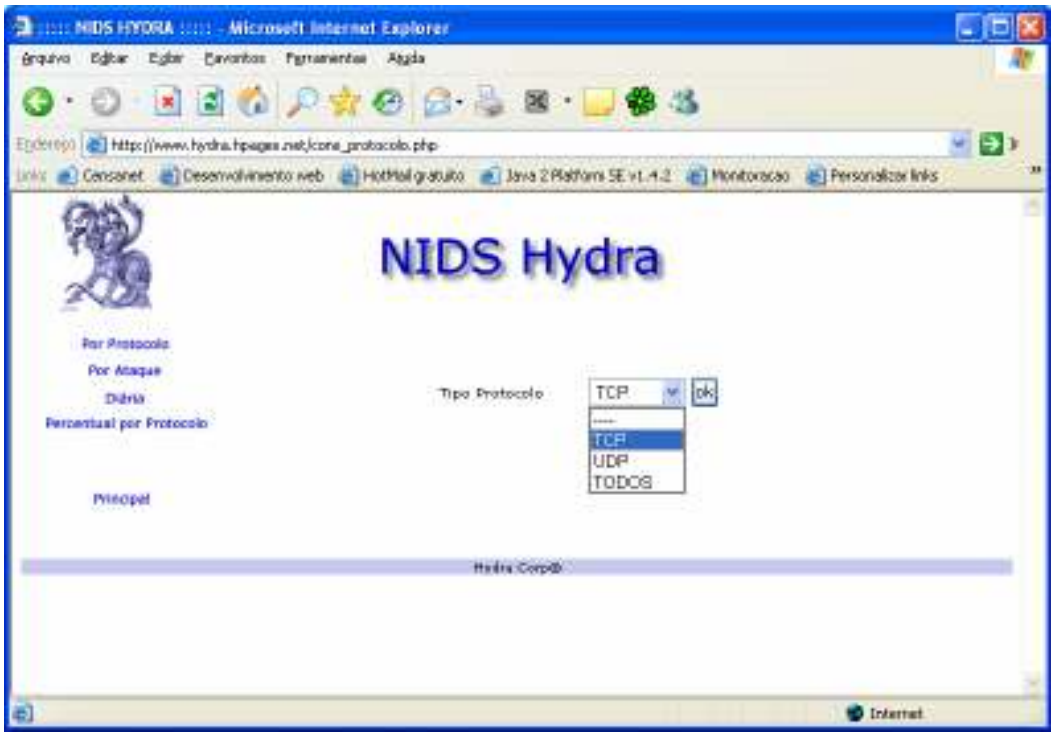

Figura 37 – Tela de Consulta por Protocolo do HYDRA

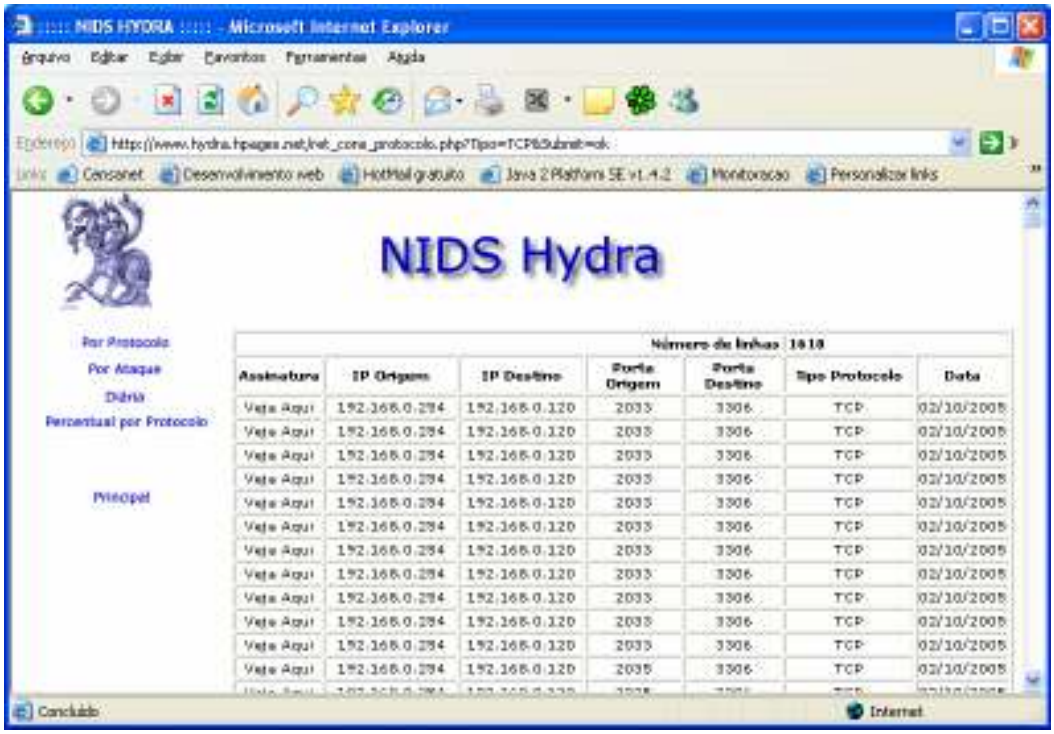

Figura 38 – Resultado da Consulta por Protocolo TCP do HYDRA

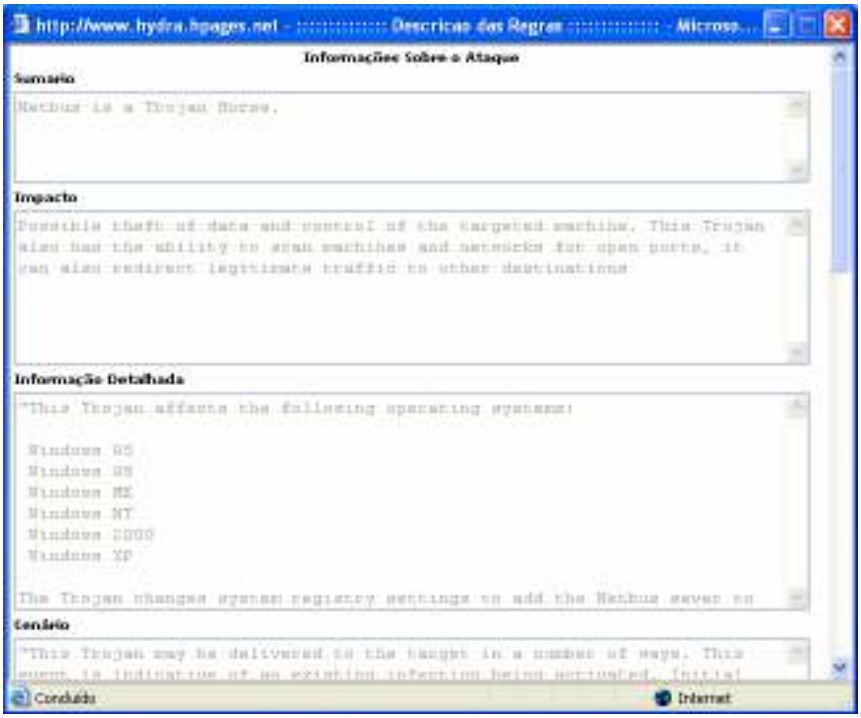

Figura 39 – Descrição da Assinatura Detectada, Informando Sumário, Impacto, etc

6.5 DETALHES DE COMPILAÇÃO

Um dos pontos fortes também desejado no projeto é a sua portabilidade. Pensando nisso, foi constatado que de uma forma geral (sistemas acadêmicos normalmente desenvolvidos) não é levado muito em conta a questão de dependência, ou seja, se faz necessário alguns arquivos de sistema, normalmente bibliotecas, para o correto funcionamento do sistema. Em primeiro instante foi verificado as dependências, assim como o tamanho, conforme a *Figura 40*. Neste caso em específico, caso não exista alguma das bibliotecas instaladas localmente, o sistema não será capaz de inicializar corretamente devido as dependências. Quanto ao tamanho, o mesmo possui aproximadamente 18KB (18943 bytes).

```
[root@fedora hydra]# ldd hydra
        linux-gate.so.1 =>  (0x00e0a000)
        libmysqlclient.so.10 => /usr/lib/mysql/libmysqlclient.so.10 (0x0011f000)
        libz.so.1 => /usr/lib/libz.so.1 (0x001ff000)
        libpcap.so.0.8.3 => /usr/lib/libpcap.so.0.8.3 (0x00361000)
        libc.so.6 => /lib/libc.so.6 (0x00254000)
        libcrypt.so.1 => /lib/libcrypt.so.1 (0x006f9000)
        libnsl.so.1 => /lib/libnsl.so.1 (0x00c6b000)
        libm.so.6 => /lib/libm.so.6 (0x00212000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x0023f000)
[root@fedora hydra]#
```

Figura 40 – Sistema Compilado Utilizando Bibliotecas Compartilhadas (shared library)

Pensando em uma melhoria de qualidade, tanto de desenvolvimento, quanto para quem irá utilizar e/ou dar suporte ao sistema, ele foi compilado de forma a estar "embutido" em seu binário todas as bibliotecas que possui como dependência (ou quase todas) como mostra a *Figura 41*, tornando assim o sistema mais portável e seguro, em contra partida, o seu tamanho aumenta consideravelmente, indo para aproximadamente 380KB (382579 bytes).

```
[root@fedora hydra]# ldd hydra
        linux-gate.so.1 =>  (0x00b23000)
        libc.so.6 => /lib/libc.so.6 (0x00254000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x0023f000)
[root@fedora hydra]#
```

Figura 41 – Sistema Compilado Utilizando Bibliotecas Estáticas (static library)

Fica claro que é relativamente mais seguro e portável se criar sistemas onde não há dependências externas (bibliotecas), sendo assim é demonstrado abaixo a mudança necessária no código para a implementação do mesmo.

Conforme *Figura 42*, é apresentado um makefile padrão, ou seja, um makefile onde não há preocupação de se utilizar bibliotecas compartilhadas. Ao se executar o comando make, o compilador verifica as flags informadas em lflags (-lmysqlclient –lz -lpcap) e pesquisa pelas bibliotecas no diretório atual e também na variável LD_LIBRARY_PATH caso esteja definida. Caso não exista ou esteja em um diretório diferente, deve ser indicado o flag "-L" seguido do caminho completo até a biblioteca.

```
# Makefile criado para o HYDRA

CC=/usr/bin/gcc
RM=/bin/rm -rf
LIBDIR=/usr/lib/mysql
lflags=-lmysqlclient -lz -lpcap

all:DEPENDENCIAS
        $(CC) -o hydra hydra.c  Geral.o LerConfig.o \
                AlertaCliente.o Regras.o \
                Inserir.o -L$(LIBDIR) $(lflags)
DEPENDENCIAS:
        $(CC) -g -c Geral.c
        $(CC) -g -c LerConfig.c
        $(CC) -g -c AlertaCliente.c
        $(CC) -g -c Regras.c
        $(CC) -g -c Inserir.c
clean:
        $(RM) *.o
        $(RM) hydra
```

Figura 42 – Makefile Gerando Binário com dependências de Sistema (bibliotecas)

Na *Figura 43*, pode-se notar que a biblioteca estática está sendo referenciada, ou seja, ao se compilar e gerar o pacote, o mesmo não terá como pendência externa algumas

bibliotecas do sistema, pois a mesma estará embutida no executável. Em nosso caso, a biblioteca do MySQL, LibZ e LibPCAP estarão embutidas.

```
# Makefile criado para o HYDRA

CC=/usr/bin/gcc
RM=/bin/rm -rf

all:DEPENDENCIAS
	$(CC) -g -o hydra hydra.c Geral.o LerConfig.o \
		AlertaCliente.o Regras.o \
		Inserir.o /usr/lib/mysql/libmysqlclient.a \
			/usr/lib/libz.a /usr/lib/libpcap.a
DEPENDENCIAS:
	$(CC) -g -c Geral.c
	$(CC) -g -c LerConfig.c
	$(CC) -g -c AlertaCliente.c
	$(CC) -g -c Regras.c
	$(CC) -g -c Inserir.c
clean:
	$(RM) *.o
	$(RM) hydra
```

Figura 43 – Makefile Gerando Binário sem dependências de Sistema (bibliotecas)

Outra observação importante é a utilização do flag –g (com tal flag, o gcc produz informações de debug em formato nativo do Sistema Operacional, como stabs, COFF, XCOFF ou DWARF), pois somente assim é possível a utilização do gdb para debug do sistema.

6.6 SOFTWARES UTILIZADOSS

Foi utilizado para a implementação do sistema os seguintes pacotes com suas respectivas versões:

- MySQL (versão utilizada - 3.23.58-9)
    - SGDB - Sistema Gerenciador de Banco de Dados.
- Apache (versão utilizada - 2.0.49-4)
    - Aplicativo responsável pelo SERVIDOR Web (Web Server).
- PHP (versão utilizada - 4.3.4-11)
    - Linguagem interpretada que funciona como extensão do protocolo HTML e roda no próprio Web Server. Responsável principalmente pela automação do Sistema via Web.

6.6.1 PASSO A PASSO – INSTALAÇÃO

Para a instalação dos pacotes abaixo, pode ser copiado diretamente da mídia de instalação do Sistema Operacional, ou estar baixando de algum site da *Internet*, ou acessar o site referente ao Projeto em questão (*www.hydra.hpages.net*).

MYSQL

Para a Instalação do MySQL, *Sistema Gerenciador de Banco de Dados*, se faz necessário a instalação de dois pacotes, o mysql e o mysql-server. Conforme *Figura 44*, vá ao diretório onde se encontra o pacote .rpm e digite o comando: rpm -ivh mysql-3.23.58-9.i386.rpm para instalar o mysql e o comando: rpm -ivh mysql-server-3.23.58-9.i386.rpm para instalar o mysql-server.

Figura 44 – Instalação dos Pacotes MySQL e MySQL Server

Em seguida, executar o comando: chkconfig mysqld on para habilitar o serviço no SERVIDOR e o comando: service mysqld start para inicializar o mesmo.

PHP

Para a instalação do PHP, termo que designa PHP: *Hypertext Preprocessor,* é uma linguagem de server-side e open-source para criação de Web pages dinâmicas e outros aplicativos da Web, se faz necessário a instalação de dois pacotes, o php e o php-mysql. Conforme ilustração abaixo, vá ao diretório onde se encontra o pacote .rpm e digite o comando: rpm -ivh php-4.3.4-11.i386.rpm para instalar o php e o comando: rpm -ivh php-mysql-4.3.4-11.i386.rpm para instalar o php-mysql.

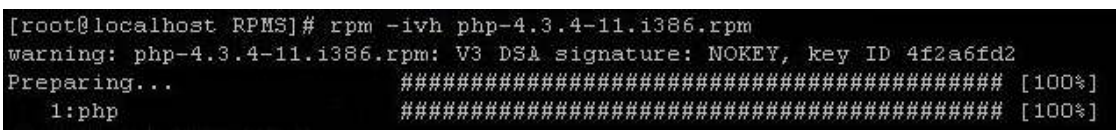

Figura 45 – Instalação do Pacote PHP

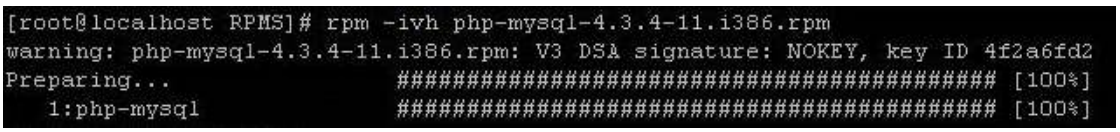

Figura 46 – Instalação do Suporte MySQL ao PHP

APACHE

Para a instalação do Apache, *WebServer ou SERVIDOR WWW*, se faz necessário a instalação de um pacote, o httpd. Conforme ilustração abaixo, vá ao diretório onde se encontra o pacote .rpm e digite o comando: rpm –ivh httpd-2.0.49-4.i386.rpm para instalar o httpd.

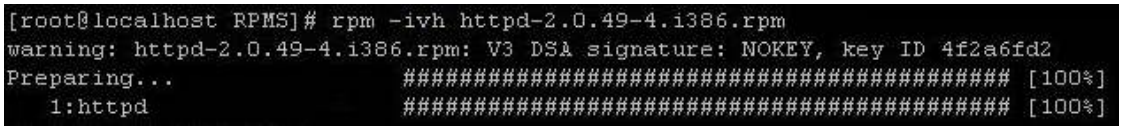

Figura 47 – Instalação do Pacote Apache

Em seguida, executar o comando: chkconfig httpd on para habilitar o serviço no SERVIDOR e o comando: service httpd start para inicializa-lo.

6.7 ESTRUTURA IMPLEMENTADA

6.7.1 SISTEMA OPERACIONAL

Está sendo utilizado para testes o Sistema Operacional da RedHat (Fedora Core 2), kernel 2.6.5-1.358 no SERVIDOR NIDS HYDRA, o Sistema Operacional da SlackWare (versão 9.1) e o Sistema Operacional MS Windows XP©.

6.7.2 EQUIPAMENTO

Nesta sessão é citado os equipamentos utilizados para o desenvolvimento e implementação do NIDS HYDRA.

O SERVIDOR onde está sendo testado possui a seguinte configuração:

- Processador Pentium 200

- 128MB RAM
- 6GB HD
- NIC Realtek 8139 (Fast Ethernet)
- System Board ASUS – P5A-B

Micro de Teste 01

- Processador Atlhon XP 2600+
- 512MB RAM
- 80GB HD
- Asound 10/100 (Fast Ethernet)
- System Board MSI – KT4-Ultra

Micro de Teste 02 (NoteBook Compaq Armada 1500c)

- Processador Celeron 366
- 160MB RAM
- 4.1GB HD
- PCMCIA - NIC Realtek 8139 (Fast Ethernet)

6.7.3 CENÁRIO

O cenário de teste utilizado baseia-se em três equipamentos (micro computador PC) descritos acima e um equipamento de rede (HUB) de 10/10 Mbits de 12 portas, conforme *Figura 48*.

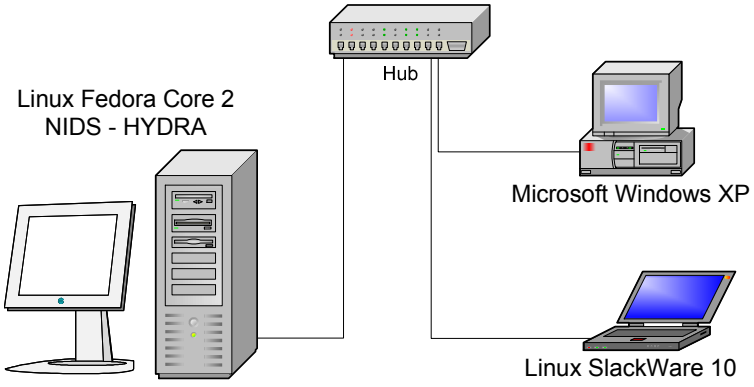

Figura 48 – Ambiente Utilizado para Desenvolvimento e Implementação

Também foi instalado no SERVIDOR uma versão do SNORT (2.4.0-1) para testes e comparações em relação a pacotes detectados.

6.8 TESTES E RESULTADOS

Foi levado em consideração nos testes dois pontos importantes, sendo esses a funcionalidade do sistema no que foi proposto , assim como a utilização de recursos de acordo com a utilização.

Quanto a funcionalidade, foi testado utilizando-se da ferramenta NESSUS (scanner de vulnerabilidades) somente os plugins de FTP e RPC (devido a grande quantidade de plugins, foi escolhido dois para análise e teste). Conforme *Figura 49 e 50*, em um primeiro momento (utilizado o Nessus como scanner e o sistema NIDS HYDRA ativo) foram detectados 1 alerta crítico, 1 aviso e 11 informações. Vale citar que a máquina checada não possui o agente HYDRA.

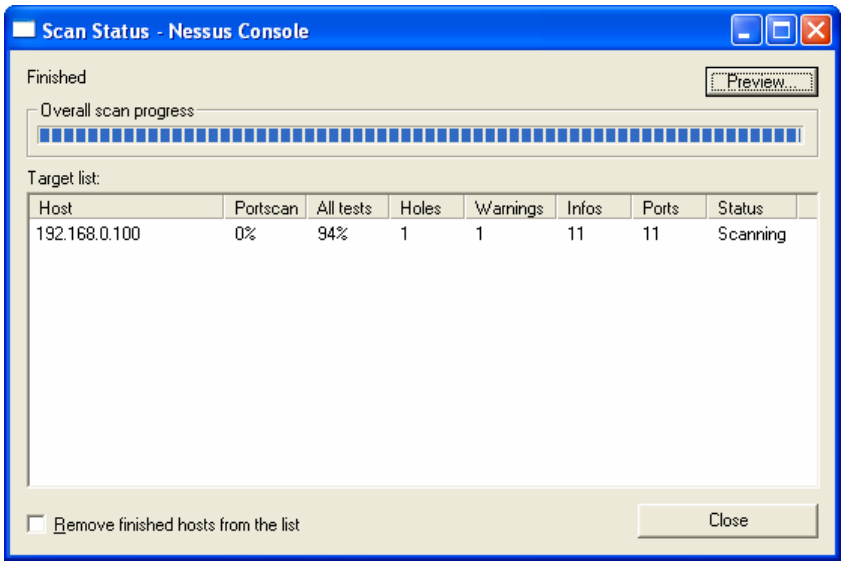

Figura 49 – Resultado do Nessus (exibindo 1 alerta crítico, 1 aviso e 11 info)

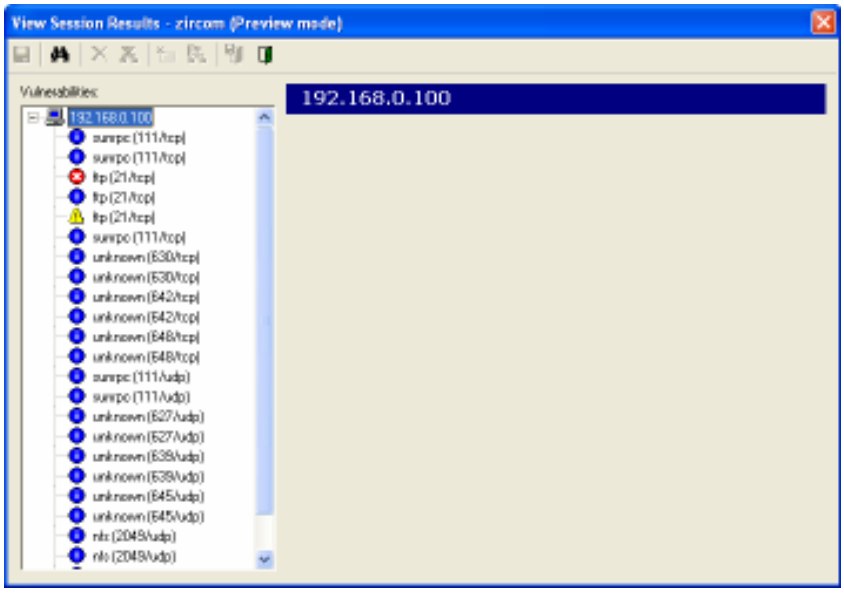

Figura 50 – Resultado do Nessus Detalhado

Neste ponto, o agente HYDRA está carregado e novamente o scanner foi utilizado, podendo constatar de acordo com a *Figura 51*, que utilizando os mesmos plugins, não foi mais detectado alerta crítico, aviso ou informações, comprovando assim a eficácia do sistema perante o que foi proposto.

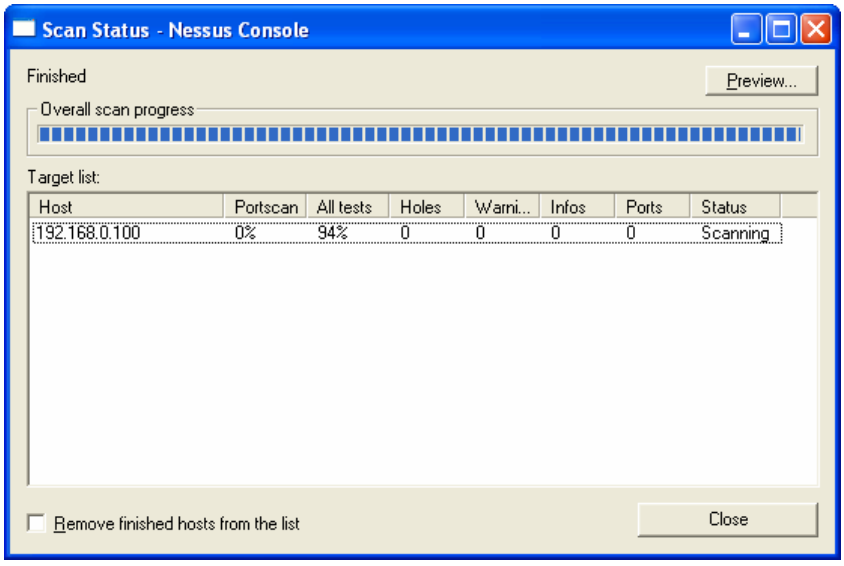

Figura 51 – Resultado do Nessus com HYDRA (exibindo 0 alerta crítico, 0 aviso e 0 infos)

Quanto a utilização de recursos do *Hardware*, como memória, cpu e rede. É apresentado de forma independente a utilização dos principais aplicativos que trabalham em conjunto com o NIDS HYDRA, como o MySQL e Web Server (Apache). Conforme *Figura 52* pode-se verificar que quando o WebServer Apache (httpd) é carregado, é alocado um processamento em aproximadamente 90% de CPU por um tempo determinado. Após a carga, o mesmo processamento retorna ao estado "normal".

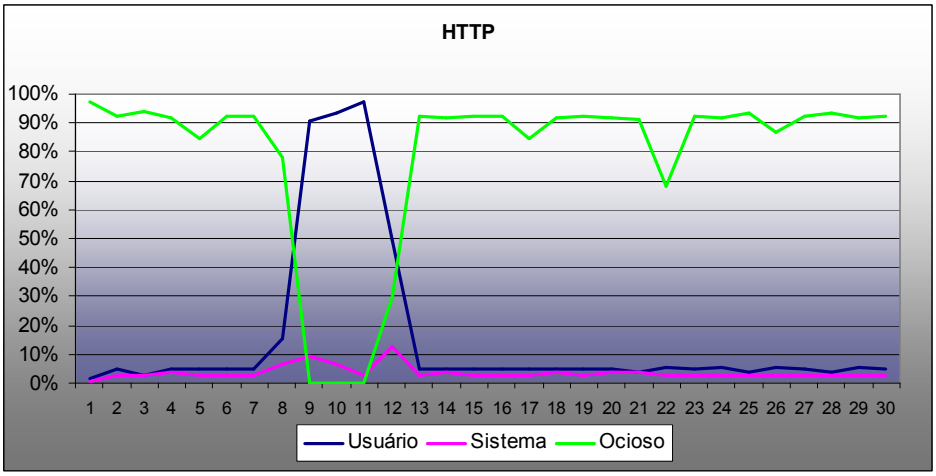

Figura 52 – Gráfico de Utilização de CPU – Apache (HTTPd)

Em relação a utilização de memória efetuada pelo Web Server Apache, fica claro de acordo com a *Figura 53* que não é utilizado muito desse recurso (aproximadamente 5MB). Essa demanda por memória pode variar de acordo com as funcionalidades que possam vir a ser habilitadas para algumas novas funcionalidades no sistema HYDRA.

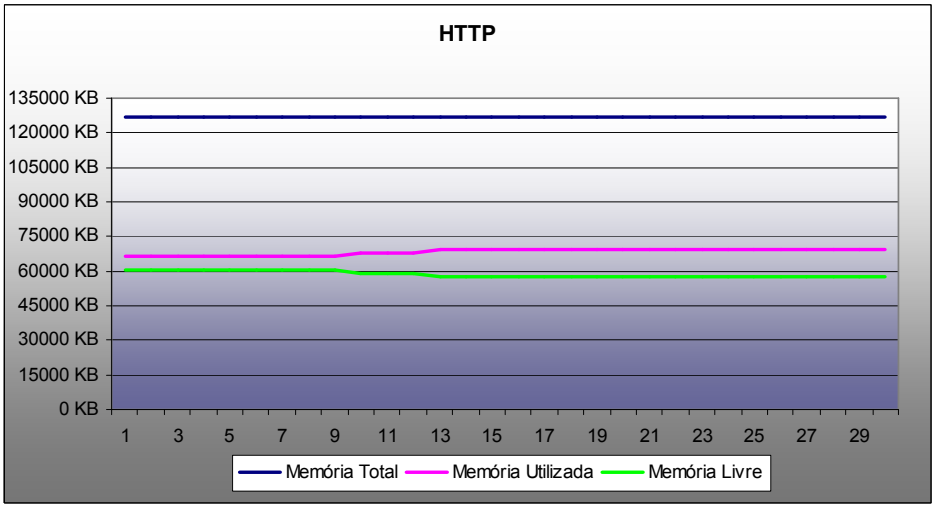

Figura 53 – Gráfico de Utilização de Memória – Apache (HTTP)

Na *Figura 54* pode-se verificar que quando o MySQL é carregado, é alocado um processamento mínimo de CPU por um tempo determinado. Após a carga, o mesmo processamento retorna ao estado quase que normal e assim se mantém. Vale enfatizar dois pontos:

- Quando o MySQL é incializado, aproximadamente 20% do processamento é efetuado pelo usuário efetivo (mysql) e aproximadamente outros 20% do processamento é efetuado pelo usuário de sistema (root). Conclui-se que tal aplicação não trabalha 100% independente do sistema, ou seja, o mesmo inicializa sub processos que devem ser gerenciados pelo sistema.
- A taxa de processamento em relação ao MySQL varia de acordo com a utilização do sistema em relação ao SGDB, caso ocorra uma quantidade excessiva de solicitação ao banco, um processamento maior será alocado, diferente de como se não houvesse requisição ao SGDB.

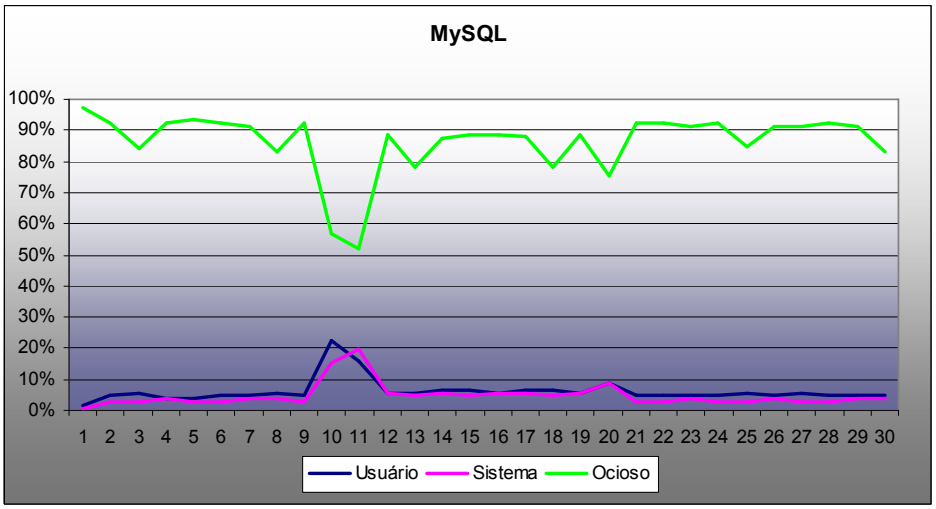

Figura 54 – Gráfico de Utilização de CPU – MySQL

Em relação a sistemas gerenciadores de banco de dados, nota-se na *Figura 55* que o MySQL consome uma quantidade excessiva de memória, pois isso é um fato característico desses sistemas, ainda assim é considerado um dos mais rápidos, flexível e leve dentre os da atualidade. No teste executado o consumo de memória está em aproximadamente 15MB, fazendo-se assim necessário uma quantidade maior de memória RAM para a implementação do sistema SGDB MySQL.

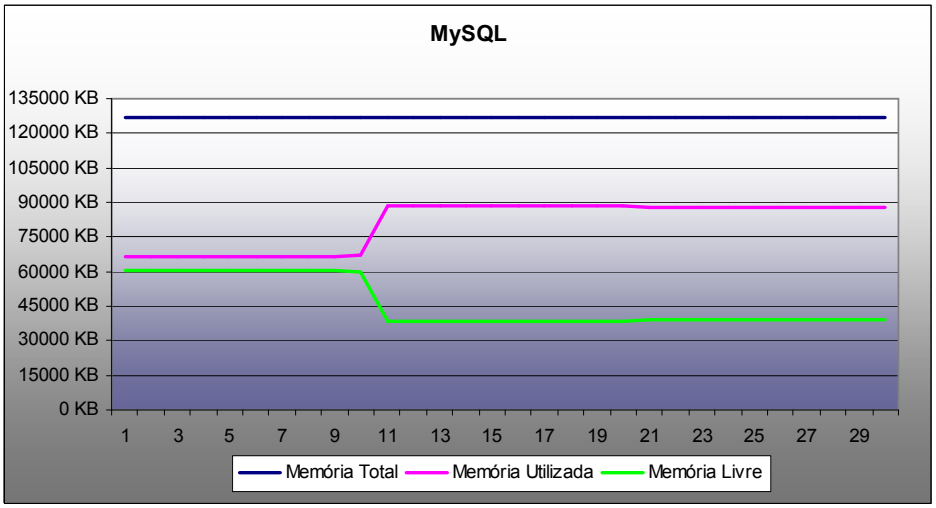

Figura 55 – Gráfico de Utilização de Memória – MySQL

Na *Figura 56* pode-se verificar que quando o NIDS HYDRA é carregado, o processamento é praticamente inalterado, ou seja, seu processamento é muito baixo, mantendo constante o seu estado. Esse comportamento é variável, pois de acordo com a quantidade de pacotes a serem analisados (trafego na rede), o sistema estará computando e tratando a procura de um que satisfaça a condição pré-estabelecida nas regras.

O percentual de utilização varia entre 2% e 8%, que representa um índice baixo e libera o uso desse requisito computacional para outros fins.

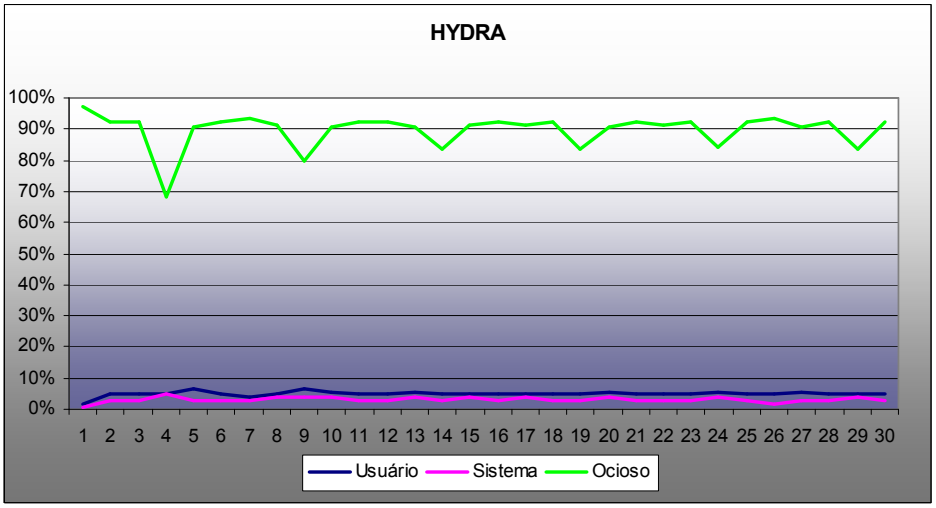

Figura 56 – Gráfico de Utilização de CPU – HYDRA

Em relação a utilização de memória, na *Figura 57* fica claro e evidente o bom desempenho do NIDS HYDRA sobre o assunto, se mantendo de forma constante e regular durante todo processo de análise onde sua capacidade foi amplamente testada. Apesar de um

nível considerado alto é preciso considerar que a natureza desses sistemas (IDS e / ou NIDS) demanda uma estrutura de dados bem elaborada e que são carregadas no processo de inicialização, depois esses dados ficam estruturados como listas encadeadas que vão garantir rápidas consultas e pesquisas na fase de análise de regras. Vale citar que com o passar do tempo, a quantidade de regras aumenta, implicando assim em uma quantidade maior de alocação em memória.

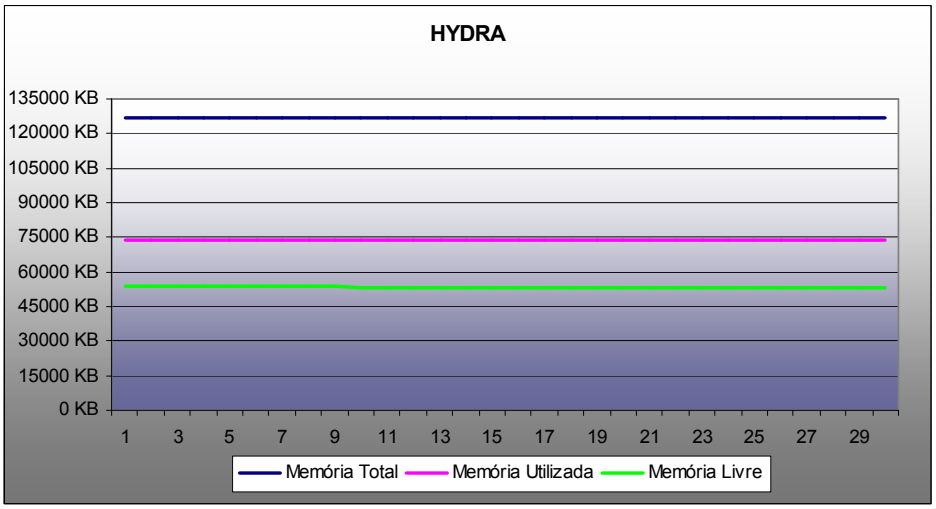

Figura 57 – Gráfico de Utilização de Memória – HYDRA

Em uma análise dos três sistemas trabalhando juntos, como se pode ver na *Figura 58*, a utilização de CPU no inicio se mantém constante e abaixo dos 10% (sistema com o MySQL, Web Server e NIDS HYDRA desativados), fato esse que se modifica um pouco e alcança os 40% (inicialização do SGDB MySQL) e atinge o pico máximo de 95% de requisição de CPU (inicialização do Web Server), comprometendo a execução de outros processos durante um período curto. Depois foi observado e comprovado uma liberação dos recursos que demandavam o uso intenso de CPU voltando a uma regularidade de 10% para menos com o andamento do teste e assim se mantendo constante.

É importante observar que cada processo é inicializado como um usuário da aplicação, sendo o HYDRA, Web Server Apache e partes do MYSQL como sistema (root).

O resultado esperado para o NIDS HYDRA no desempenho computacional referente a CPU foi satisfatório quando comparado com outros sistemas NIDS em relação a processamento e utilização de memória, cujo consumo de CPU foi muito superior ao atingido pelo NIDS HYDRA, sendo o mesmo uma ferramenta que considera esse recurso muito importante desde a sua concepção.

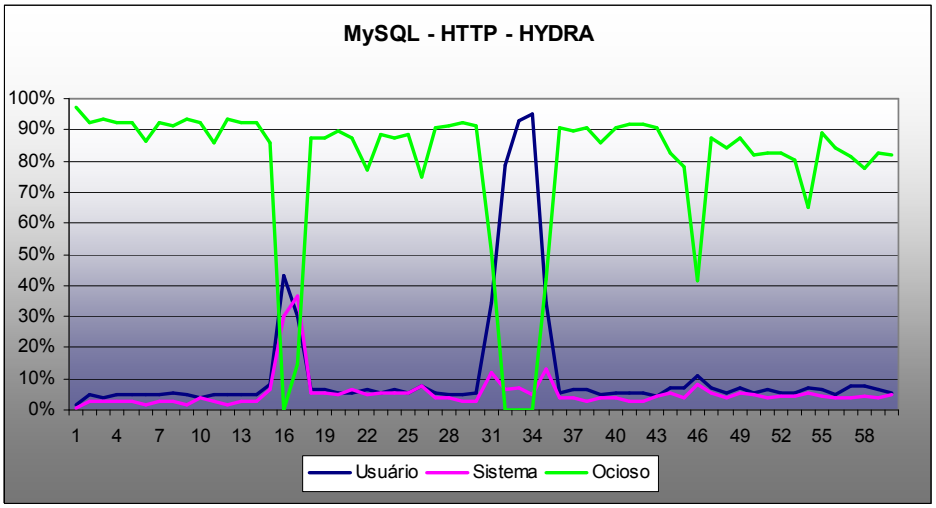

Figura 58 – Gráfico de Utilização de CPU – Apache, MySQL e HYDRA

Quanto ao uso de memória a *Figura 59* mostra que esse recurso é muito requisitado pois como já foi dito um NIDS trabalha com uma estrutura de dados complexa. Deve-se lembrar (de acordo com a *Figura 55*) que o MySQL é o que mais consome memória do sistema. A verificação de demanda deste recurso serve para indicar um planejamento adequado para a implementação do sistema com a quantidade ideal para seu perfeito funcionamento, e se possível em quantidade extra.

O resultado esperado foi alcançado em conseqüência dos detalhes de implementação e estratégias de compilação que conferem ao NIDS HYDRA um uso racional desse recurso computacional juntamente com a preocupação de se evitar bugs e brechas na programação que possam ser exploradas.

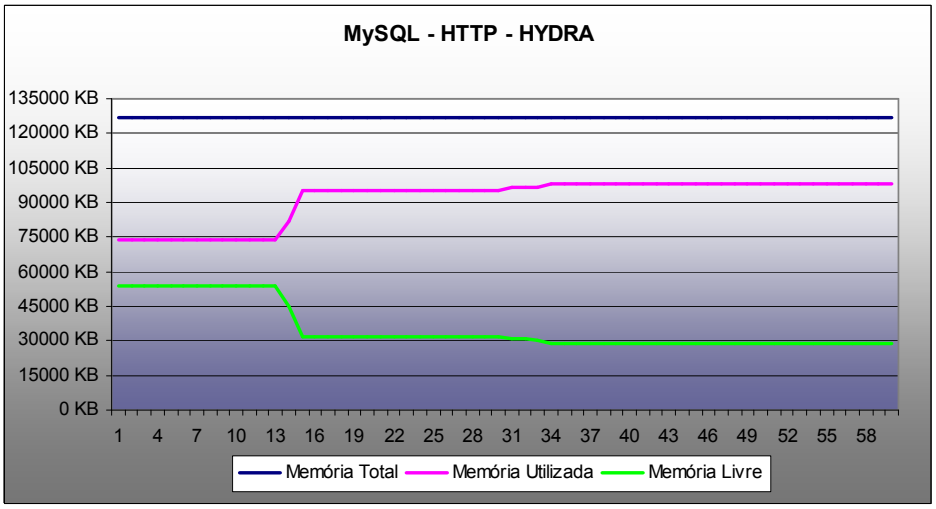

Figura 59 – Gráfico de Utilização de Memória – Apache, MySQL e HYDRA

Quanto a obtenção dos dados acima, foi utilizado o recurso do próprio Sistema Operacional para coleta dos dados, utilizando-se do utilitário top[6] e importando os dados para uma planilha Excel, para posteriormente estar gerando os gráficos.

6.8.1 SOFTWARES UTILIZADOS

Foram utilizados alguns *softwares* para levantar as possíveis vulnerabilidades nos equipamentos localizados na rede, assim como simular tipos de ataques variados, descritos logo abaixo:

NESSUS

O Nessus é um programa de varredura de portas de TCP desenvolvido na linguagem de programação C e construído na plataforma Linux. É uma ferramenta composta de duas partes: um SERVIDOR e um cliente. Ele foi desenvolvido para ser rápido, seguro e possui uma arquitetura modular que permite ajustá-lo às diversas necessidades dos usuários ou administradores de rede. Possui mais de 1.200 testes, sendo que esse número cresce de acordo com o surgimento de novos ataques. Eles são divididos em 23 categorias diferentes como, por exemplo, “Denial of Service”, “Port scanners”, “SMTP problems”, entre outros. A ferramenta pode ser utilizada via linha de comando ou interface gráfica.

NMAP

Nmap (*Network Mapper*), uma ferramenta de varredura disponível na *Internet* que fornece recursos de varredura TCP e UDP básicos. Esse programa de varredura é considerado um dos mais completos, já que reúne em um mesmo pacote várias funções. [FER2003]

[6] Utilitário para exibir as tarefas no Sistema Operacional Linux

Capítulo 7

Apresenta um procedimento fácil e prático para a implementação do sistema NIDS HYDRA.

7 INSTALAÇÃO

Para se instalar a ferramenta, primeiro deve ser efetuado os passos do capítulo 5.6.1, Instalação do MySQL, Web Server e PHP. Em seguida, deve ser feito o *download* do código fonte do NIDS HYDRA e a Estrutura do Site de Visualização a partir do site de demonstração – http://www.hydra.hpages.net/ ou a partir da mídia digital que acompanha a monografia.

CÓDIGO FONTE

Copiar o arquivo hydra.<data de atualização>.tar para uma pasta no linux e executar o comando "**tar xvf hydra.<data de atualização>.tar**", neste momento será criado uma sub pasta com o nome de hydra, entre no diretório e execute o comando "**make**" e em seguida "**make install**". Agora o pacote será compilado e instalado em um path padrão (/apps/hydra), sendo esses o binário, arquivo de configuração e o arquivo de regras.

Obs.: Não foi definido neste trabalho um pacote específico (como rpm ou deb) de acordo com a distribuição, mas sim o fonte com o makefile apropriado para que o sistema esteja operacional. Quanto ao arquivo de inicialização, fica a critério do administrador definir a melhor forma de carregamento, mas uma base já é criada automaticamente em "/etc/init.d" chamado de hydrad.

SITE DE VISUALIZAÇÃO

Copiar o arquivo www.<**data de atualização**>.zip para o diretório configurado como root do Web Server e executar o comando "**unzip www.<data de atualização>.zip**", neste momento será criado uma sub pasta com o nome de hydra, entre no diretório e altere o arquivo de configuração conn.php. Neste arquivo será necessário a alteração de quatro campos somente, sendo eles servidor (nome ou endereço IP do servidor de Banco de Dados), usuário (usuário com acesso as tabelas do NIDS HYDRA), senha (senha com que o usuário pode acessar o Banco de Dados) e bd (nome do Banco de Dados utilizado pelo NIDS HYDRA), sendo o último normalmente PF.

Obs.: Pode ser feito uma configuração no Web Server, de forma que o NIDS HYDRA seja a pagina principal, mas o mesmo não será descrito nesta monografia.

BANCO DE DADOS

Para a implementação do Banco de Dados, no arquivo hydra.<data de atualização>.tar existe uma pasta chamada db, ao extrair, pode-se executar o comando "**mysql < PF.<data de atualização>.sql**". Assim será criada a estrutura utilizada pelo front end.

Obs.: Deve ser alterado o arquivo de configuração referente ao acesso a Banco de Dados localizado em "/apps/hydra" chamado de hydra.conf. O mesmo possui informações como Servidor, Usuário, Senha e Banco de Dados.

Capítulo 8

Apresenta conclusões, descrevendo os objetivos alcançados e planos para aprimorar a ferramenta em trabalhos futuros.

8 CONCLUSÕES

O NIDS Hydra foi desenvolvido com foco educacional, sendo que ainda há muito por ser implementado e aprimorado, mas neste projeto, pode-se dizer que realmente ocorreu uma absorção de conhecimento, pois foi completamente desenvolvido (levando-se em consideração as funcionalidades de algumas outras ferramentas do gênero), fato que geralmente não ocorre em projetos voltados para IDS ou NIDS.

Conforme estimado no início do projeto, o NIDS HYDRA funcionou de acordo com o esperado. A suíte composta da aplicação de captura de pacotes (TCP e UDP) e análise de regras, os agentes para comunicação entre clientes e SERVIDOR NIDS, assim como o front end para visualização dos dados referente a ataques via WEB.

Pode-se afirmar que o sistema HYDRA não utilizou tanto recurso do equipamento, de acordo com o gráfico mostrado na *Figura 60*, pode-se verificar que no instante 15 (momento de inicialização do HYDRA), ocorreu um pico de processamento, sendo que o mesmo será alterado de acordo com o sistema (quantidade de regras x quantidade de pacote a ser analisado), logo, quanto maior o número de regras e maior a quantidade de pacotes a ser analisado, maior o processamento alocado pelo HYDRA.

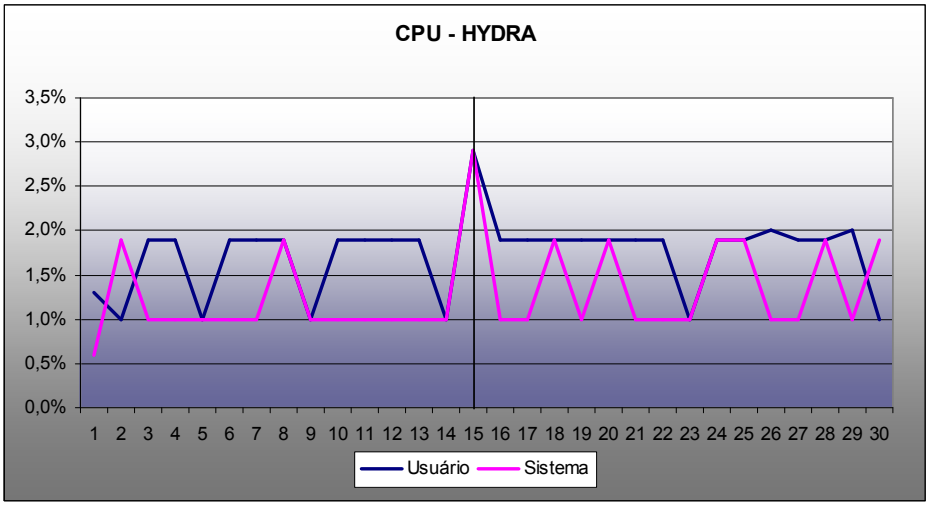

Figura 60 – Gráfico de Utilização de Memória –HYDRA

Quanto as metas traçadas para esse projeto, além de se conhecer as técnicas utilizadas geralmente em ataques a sistemas computacionais (verificação de vulnerabilidades, utilização de aplicativos para conquista de privilégios em sistemas remoto, levantamento de informação), foi também testado os sistemas NIDS Snort e Firestorm. Vale citar que como o projeto foi testado em um ambiente dotado de um equipamento de rede (HUB), o mesmo não possui capacidade de gerenciar o domínio de colisão, repassando para todas as portas uma cópia dos pacotes. Caso seja utilizado um switch, pode ser utilizado a opção de *Port Span* ou *Port Mirror*.

8.1 SUGESTÕES PARA TRABALHOS FUTUROS

Este trabalho possui precedentes para uma série de outras funcionalidades que podem ser adicionadas ao sistema, como por exemplo:

REGRAS EM BASE DE DADOS

As regras ficam armazenadas em banco de dados, de forma a ter uma portabilidade mais flexível e uma segurança maior, se comparado a regras em arquivos.

PROTOCOLO IPV4, IGMP, IPX E ICMP

Suporte a uma variedade de protocolos diferentes, tornando assim o NIDS mais "sensível" aos possíveis ataques.

CONFIGURAÇÕES EM BANCO DE DADOS

Dados referente a configuração, como nome de usuário, senhas, dentre outras informações, estarem armazenados em Banco de Dados de forma a tornar o ambiente mais seguro, por exemplo com a senha criptografada.[7]

FRONTEND PARA ADMINISTRAÇÃO DAS REGRAS VIA WEB

No módulo de navegação para consultas em geral (WEB) pode haver uma área específica para administração, tanto das regras, como habilitar e desabilitar falsos positivos, importar e exportar regras, dentre outras funções administrativas.

ATAQUES DOS (DENY OF SERVICE)

Módulo para análise e resposta (proteção) a ataque do tipo DoS, pois tal ataque não é baseado em assinatura, e sim em quantidade de requisição.

[7] Pode ser utilizado a função crypt para criptografar um dado.

MULTI THREAD

Desenvolvimento do sistema baseado em multi thread (processamento paralelo) para um melhor desempenho e diminuição de consumo centralizado de recurso do equipamento.

SCANNER DE VULNERABILIDADE

Implementação de um scanner de vulnerabilidades, pois assim a própria ferramenta seria capaz de verificar possíveis vulnerabilidades e reportar ao administrador, diminuindo assim a probabilidade de invasão em algum de seus equipamentos.

Capítulo 9

Apresenta as Referências Bibliográficas e Webgráficas, onde especifica livros e sites utilizados para pesquisa e consulta para o desenvolvimento do sistema.

---

9 REFERÊNCIAS

[AISC] Associação Internacional de Segurança em Computadores, Disponível em: <http://www.icsa.net>. Acesso em: 10 out. 2004.

[ANO2001] ANONIMO. *Segurança Máxima*. 3. ed. Rio de Janeiro: Campus, 2001.

[BRI2002] BRITO, Gustavo Arantes de; RESIO, Gustavo Ferreira. Sistema de Detecção de Intrusão. Universidade Federal de Goiás. Disponível em: <www.eee.ufg.br/cepf/pff/2002/ee_20.pdf>. Acesso em: 20 Ago. 2005.

[CAM2001] CAMPELLO, Rafael Saldanha; WEBER, Raul Fernando. Apostila "Sistema de Detecção de Intrusão", Instituto de Informática – UFRGS. Disponível em: <http://www.inf.ufrgs.br/~gseg/producao/minicurso-ids-sbrc-2001.pdf - 2001>. Acesso em: 20 Ago. 2005

[CAT2005] CATHO Online, Companhias afirmam que já sofreram alguma forma e ataque, Disponível em: <http://www3.catho.com.br/sites/telecom/jornal/inputer_view.phtml?id=4714>. Acesso em: 20 jun. 2005.

[CERI] CERIAS - *Intrusion Detection*, Disponível em: <http://www.cerias.purdue.edu/about/history/coast/intrusion-detection.welcome.html>. Acesso em: 27 jan. 2005.

[CERT] CERT - *Coordination Center*, Disponível em: <http://www.cert.org>. Acesso em: 16 ago. 2005.

[COM2005] *Common Intrusion Detection Framework*, Disponível em: <http://www.isi.edu/gost/cidf/>. Acesso em: 10 fev 2005.

[COND] Conduta Ética. Tipos de invasão e as principais técnicas utilizadas. Disponível em: <http://condutaetica.cfu.com.br/tipos_de_invasao.html>. Acesso em: 08 set. 2005.

[COS2004] COSTA, Cristiano Reinaldo Itapoan da. Sistema Para Grupos de Respostas a Incidentes de Segurança em Computadores. Canoas, 2004. (Bacharel em Ciência da Computação) - Universidade Luterana do Brasil - Câmpus Canoas.

[CVEX] Common Vulnerabilities Exposed. Disponível em: <http://cve.mitre.org>. Acesso em:12 dez. 2004.

[DET2005] Em busca de uma Padronização para Sistemas e Detecão de Intrusão., Disponível em: <http://www.frontthescene.com.br/artigos/IDMEF_IDXP_CIDF_1_2.pdf >. Acesso em: 10 jul. 2005.

[EIRC] Estatísticas dos Incidentes Reportados ao CERTbr, Disponível em: <http://www.nbso.nic.br/stats/incidentes >. Acesso em: 20 mai. 2005.

[FER2003] FERREIRA, Bárbara Chiavaro. Sistemas de Detecção de Intrusão. Gravataí, 2003. Monografia (Bacharel em Ciência da Computação) - Universidade Luterana do Brasil - Câmpus Gravataí. Disponível em: <www.ulbra.tche.br/~roland/tcc-gr/ monografias/2003-2-tc2-Barbara_Chiavaro_Ferreira.pdf>. Acesso em: 20 Ago. 2005

[FIRE] Firestorm NIDS. Disponível em: <http://www.scaramanga.co.uk/firestorm>. Acesso em:09 set. 2005.

[GLOS] Glossário de Segurança focado em *Internet*, Disponível em: <http://www.ietf.org/rfc/rfc2828.txt>. Acesso em: 15 fev 2005.

[HEA1990] HEADY, Richard; LUGER, George; MACCABE, Arthur et al. *The Architecture of a Network Level Intrusion Detection System*. University of New Mexico - Department of Computer Science, 1990.
[HOL1991] HOLBROOK, Paul; REYNOLDS, Joyce. Site Security Handbook. *Network Working Group*. 1991. Disponível em: <http://www.ietf.org/rfc/rfc1244.txt>. Acesso em: 27 ago. 2005.
[HONE] Projeto HoneyNet, Disponível em: <http://www.honeynet.org/misc/project.html>. Acesso em: 10 mar. 2005.
[IETF] *Internet Engineering Task Force*. Disponível em: <http://www.ietf.org/>. Acesso em: 20 jul. 2004.
[IRIN] Incidentes Reportados na *Internet*. Disponível em: <http://www.cert.org/stats/cert_stats.html#incidents>. Acesso em: 10 out. 2004.
[JUN2003] JUNIOR, Julio Steffen. Sistemas de Detecção de Intrusão. Novo Hamburgo, 2003. Monografia (Bacharel em Ciência da Computação) - Centro Universitário FEEVALE, 2003.
[LOU2004] LOUZADA, Carlos Renan Schick. Sistema de Detecção de Intrusão Para Redes de Computadores Baseado na Análise de Padrões em Shellcodes. Canoas, 2004. Monografia (Bacharel em Ciência da Computação) - Universidade Luterana do Brasil - Câmpus Canoas. Disponível em: <http://www.ulbra.tche.br/~tcc-canoas/defesas2004-2.html>. Acesso em: 20 Ago. 2005.
[MIM2005] Especificação S/MIME, Disponível em: <http://www.ietf.org/rfc/rfc2311.txt>. Acesso em: 20 fev. 2005.
[NETF] Grupo NetFilter, Disponível em: <http://www.netfilter.org>. Acesso em: 14 dez 2004.
[PAR1994] PARKER, Donn B. *Demonstrating the elements of information security with threats*. *In Proceedings of the 17th National Computer Security Conference*. pages 421-430, 1994.
[PTA1998] PTACEK, Thomas H.; NEWSHAN, Timothy N. *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection*. Canada: Secure Networks Inc, 1998.
[RAD1997] RADATZ, Jane.The IEEE Standard Dictionary of Electrical and Electronics Terms. *IEEE Computer Society Style Guide*. 1997. Disponível em: <http://www.computer.org/portal/site/ieeecs/menuitem.c5efb9b8ade9096b8a9ca0108bcd45f3/index.jsp?&pName=ieeecs_level1&path=ieeecs/publications/author/style&file=index.xml&xsl=generic.xsl&>. Acesso em: 27 ago. 2005.
[REV1999] Revista de Informação e Tecnologia. Artigo: Ferramentas IDS, 1999. Disponível em: < http://www.revista.unicamp.br/infotec/artigos/frank4.html> Acesso em: 29 Ago. 2005.
[RFC2402] The *Internet* Engineering Task Force. Disponível em: <http://www.ietf.org/rfc/rfc2402.txt>. Acesso em: 10 mar. 2005.
[RFC2828] The *Internet* Engineering Task Force. Disponível em: <http://www.ietf.org/rfc/rfc2828.txt>. Acesso em: 10 mar. 2005.
[SCA2000] SCAMBRAY, Joel; MCCLURE, Stuart; KURTZ, George. Hacking Exposed: *Network Security Secrets & Solutions*. 2000. Disponível em: <http://www.hackingexposed.com>. Acesso em: 30 ago. 2005.
[SDIN] Sistemas de Detecção de Intrusão. Seminários Ravel CPS-760. Disponível em: <http://www.lockabit.coppe.ufrj.br/downloads/academicos/IDS.pdf >. Acesso em: 20 nov 2004.
[SECN] Primeiro Portal de Segurança da informação no Brasil. Disponível em: <http://www.securenet.com.br> Acesso em: 29 Ago. 2005.
[SMTP] Pop before SMTP, Disponível em: <http://spam.abuse.net/adminhelp/smPbS.shtml>. Acesso em: 25 nov. 2005.

[SNOR] Snort IDS. Disponível em: <http://www.snort.org>. Acesso em: 15 mar. 2005.
[STR1999] STRAUCH, Suzana Beatriz de Miranda. Aspécctos de Segurança no Protocolo IP. 1999. (Pós-Graduaçao em Computação) – Instituto de Informática – UFRGS. Disponível em: <www.modulo.com.br/pdf/mo7e001p.pdf>. Acesso em: 20 Ago. 2005.
[STR2002] STREBE, M. e PERKINS, C. Firewalls. São Paulo: Makron Books, 2002.
[THO2004] THOMSON, Laura; WELLING, Luke. Tutorial MySQL. 1. ed. Rio de Janeiro: Editora Ciência Moderna, 2004.
[UNIX] *Unix System Administration*. Disponível em: <http://wks.uts.ohio-state.edu/sysadm_course/html/sysadm-1.html>. Acesso em: 20 nov. 2004.
[WHIT] *Whitehats*. Disponível em: <http://www.whitehats.com/>. Acesso em: 10 fev. 2005.
[WIKI] Wikipedia. A Enciclopédia livre. Disponível em: <http://pt.wikipedia.org/wiki/Pagina_principal> Acesso em: 29 Ago. 2005.
[ZWI2001] ZWICKY, Elizabeth D.; COOPER, Simon; CHAPMAN, D. Brent. Construindo *Firewalls para a Internet*. 2. ed. Rio de Janeiro: Campus, 2001.

Capítulo 10

Código do sistema NIDS HYDRA, seguido dos arquivos de configuração e estrutura da Base de Dados.

10 APÊNDICE

10.1 CÓDIGO DO SISTEMA

10.1.1 MAKEFILE

```
# Makefile criado para o HYDRA

CC=/usr/bin/gcc
RM=/bin/rm -rf
CP=/bin/cp
CHKCONFIG=/sbin/chkconfig
INIT=/etc/init.d
SERVICE=/sbin/service
MKDIR=/bin/mkdir

all:DEPENDENCIAS
      $(CC) -g -o hydra hydra.c Geral.o LerConfig.o \
                  AlertaCliente.o Regras.o \
                  Inserir.o /usr/lib/mysql/libmysqlclient.a \
                    /usr/lib/libz.a /usr/lib/libpcap.a
DEPENDENCIAS:
      $(CC) -g -c Geral.c
      $(CC) -g -c LerConfig.c
      $(CC) -g -c AlertaCliente.c
      $(CC) -g -c Regras.c
      $(CC) -g -c Inserir.c

clean:
      $(RM) *.o
      $(RM) hydra

install:
      $(MKDIR) -p /apps/hydra
      $(CP) hydra /apps/hydra
      $(CP) hydra.conf /apps/hydra
      $(CP) hydra.regras /apps/hydra
      $(CP) init/hydrad /etc/init.d
```

10.1.2 HYDRA.C

```
/*
        NIDS - HYDRA

        hydra.c
*/
```

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <pcap.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>

// Includes utilizadas no Sistema
#include "Regras.h"
#include "LerConfig.h"
#include "Inserir.h"

// Variaveis utilizadas no Sistema
char IP_ORIG[20];
char IP_DEST[20];
int PORTA_ORIGEM;
int PORTA_DESTINO;
char flag_tcp[8];


// Header para inclusao dos dados em Banco de Dados

pcap_t *cap;                  // descritor da captura
pcap_dumper_t *dcap;          // descritor para despejo em arquivo
int ethhdr, mod, hex;

void uso (char *);
void sair ();
char *takerule (register char **);
pcap_handler monitor (u_char *, struct pcap_pkthdr *, u_char *);

// Herdando as Listas udp_raiz e tcp_raiz
extern lno *udp_raiz;
extern lno *tcp_raiz;

// Funcao Principal
int main (int argc, char **argv)
{

  char *dev, *arquivo, *expr, ebuf[PCAP_ERRBUF_SIZE];
  struct bpf_program filtro;
  bpf_u_int32 net, mask;           // Ponteiro para armazenar IP e mascara
da Rede
  int opt, snaplen = BUFSIZ, promisc = 0;

  while ((opt = getopt (argc, argv, "i:r:w:s:px" )) != EOF)
    {
      switch (opt)
      {
      case 'i':
        dev = optarg;
        break;
      case 'r':
        mod = 1;
        arquivo = optarg;
        break;
      case 'w':
        mod = 2;
        arquivo = optarg;
```

```
        break;
      case 's':
        snaplen = atoi (optarg);
        break;
      case 'p':
        ++promisc;
        break;
      case 'x':
        ++hex;
        break;
      default:
        uso (argv[0]);
        break;
      }
    }

  if (argc == 1)
    uso (argv[0]);

  if (argv[optind] != NULL)
    expr = takerule (&argv[optind]);      // função util do tcpdump
  else
    expr = "";

/*
 * Funcao responsavel pela leitura do arquivo de configuracao (conf
utilizada em Banco de Dados)
 */
  LerConf();

/*
 * Funcao responsavel por subir as regras, ou seja, montar as listas de
protocolo TCP e UDP em memoria
 */
  SubirRegras();


/*
 * Checa o modo selecionado, se 1, abre descritor com configuração
 * para despejo, senão abre descritor para captura.
 */
  if (mod == 1)
    {

/*
 * Função utilizada para abrir descritor com configuração para despejo de
 * arquivo captura anteriormente salvo
 */

      {
        fprintf (stderr, "pcap: %s\n", ebuf);
        exit (1);
      }
    }
  else
    {
      // Responsavel por colocar a interface em modo promisco
      if ((cap = pcap_open_live (dev, snaplen, promisc, 1000, ebuf)) ==
NULL)
      {
        fprintf (stderr, "pcap: %s\n", ebuf);
```

```
        exit (1);
      }

      if (pcap_lookupnet (dev, &net, &mask, ebuf) < 0)
        {
          fprintf (stderr, "pcap: %s\n", ebuf);
          exit (1);
        }
     }

  // Procura um ligação de dados correspondente com a captura aberta,
obrigatorio para decodificações
  switch (pcap_datalink (cap))
    {
    case DLT_NULL:
      printf ("Data Link Type: NULL\n");
      ethhdr = 4;
      break;
    case DLT_EN10MB:
      printf ("Data Link Type: Enthenet 10MB\n");
      ethhdr = 14;
      break;
    case DLT_EN3MB:
      printf ("Data Link Type: Enthenet 3MB\n");
      ethhdr = 14;
      break;
    case DLT_PPP:
      printf ("Data Link Type: PPP\n");
      ethhdr = 4;
      break;
    case DLT_SLIP:
      printf ("Data Link Tupe: SLIP\n");
      ethhdr = 16;
      break;
    case DLT_FDDI:
      printf ("Data Link Type: FDDI\n");
      ethhdr = 21;
      break;
    case DLT_RAW:
      printf ("Data Link Type: RAW\n");
      ethhdr = 0;
      break;
    default:
      printf ("Dispositivo não suportado\n");
      exit (1);
    }

  // Quando ocorrer algum sinal, a função sair() será chamada para
finalização correta do código
  signal (SIGTERM, sair);
  signal (SIGINT, sair);
  signal (SIGHUP, sair);
  signal (SIGKILL, sair);

  // Função loop de recepção e processo de cada pacote mais rica em
recursos proprios
  if ((pcap_loop (cap, -1, (pcap_handler) monitor, NULL)) < 0)
    {
      fprintf (stderr, "pcap: %s\n");
      exit (1);
    }
```

```
}


void uso (char *prog)
{
  printf ("Uso: %s [-i interface]\n", prog);
  exit (0);
}

// função para execução das funções de encerramento da libpcap
void sair ()
{
  if (mod == 2)
    pcap_dump_close (dcap);
  pcap_close (cap);
  printf ("Captura Finalizada\n");
  exit (0);
}

// Função util do tcpdump
char *takerule (register char **argv)
{
  register char **p;
  register u_int len = 0;
  char *buf;
  char *src, *dst;
  p = argv;
  if (*p == 0)
    return 0;
  while (*p)
    len += strlen (*p++) + 1;
  buf = (char *) malloc (len);
  if (buf == NULL)
    {
      perror ("malloc");
      printf ("it is better to end now.\n");
      exit (2);
    }
  p = argv;
  dst = buf;
  while ((src = *p++) != NULL)
    {
      while ((*dst++ = *src++) != '\0');
      dst[-1] = ' ';
    }
  dst[-1] = '\0';
  return buf;
}


/*
 * Função callback chamada dentro da função pcap_loop()
 * para decodificação de cada pacote
 */
pcap_handler monitor (u_char * user, struct pcap_pkthdr * pkthdr, u_char *
packet)
{
  struct ip *ip;
  struct tcphdr *tcp;
  u_char *ips, *ipd;
```

```
/*
 * Especificação do tamanho real do quadro de rede,
 * para decodificação exato de pacote.
 */
  ip = (struct ip *) (packet + ethhdr);
  tcp = (struct tcphdr *) (packet + ethhdr + sizeof (struct ip));

  // armazenando endereço ip de origem e destino
  ips = (u_char *) & (ip->ip_src);
  ipd = (u_char *) & (ip->ip_dst);

// Descobrindo o Protocolo
char TYPE[5];
switch (ip->ip_p)
{
case 1:
  {
    strcpy(TYPE,"ICMP");
    break;
  }
case 6:
  {
    strcpy(TYPE,"TCP");
    break;
  }
case 17:
  {
    strcpy(TYPE,"UDP");
    break;
  }
}

  // Atribuindo o IP Origem e Destino em suas respectivas vars
  sprintf(IP_ORIG,"%d.%d.%d.%d", ips[0],ips[1],ips[2],ips[3]);
  sprintf(IP_DEST,"%d.%d.%d.%d", ipd[0],ipd[1],ipd[2],ipd[3]);

  // Atribuindo data atual na var DATA no formato ano-mes-dia
  time_t dh;
  char DATA[30];
  dh = time(0);
  strftime(DATA,30,"%Y-%m-%d", localtime(&dh));

  PORTA_DESTINO = (int) ntohs (tcp->dest);
  PORTA_ORIGEM = (int) ntohs (tcp->source);

  if (tcp->res1 == 1 )
    strcat(flag_tcp,"1");
  if (tcp->res2 == 1 )
    strcat(flag_tcp,"2");
  if (tcp->urg == 1 )
    strcat(flag_tcp,"U");
  if (tcp->ack == 1 )
    strcat(flag_tcp,"A");
  if (tcp->psh == 1 )
    strcat(flag_tcp,"P");
  if (tcp->rst == 1 )
    strcat(flag_tcp,"R");
  if (tcp->syn == 1 )
    strcat(flag_tcp,"S");
  if (tcp->fin == 1 )
    strcat(flag_tcp,"F");
```

```
  /*
   * Funcao responsavel por analisar o pacote e comparar com as assinaturas
   * cadastradas no sistema
   */

VerificarRegra(IP_ORIG,PORTA_ORIGEM,IP_DEST,PORTA_DESTINO,TYPE,packet,flag_
tcp);

  // Responsavel por zerar a variavel flag_tcp
  strcpy(flag_tcp,"");
}
```

10.1.3 LERCONFIG.C

```
/*
        NIDS - HYDRA

        LerConfig.c
*/

#include <stdio.h>
#include "LerConfig.h"

// Responsavel por ler o arquivo de configuracao referente ao Banco de
Dados
void LerConf()
{
  d_raiz = (struct DADOS *)malloc(sizeof(struct DADOS));

  int i=0, x;
  char ch[200];
  char c1[198], c2[198], aux[198], valor[198];
  FILE *fp;

  if ((fp = fopen("hydra.conf","r")) == NULL)
  {
    printf("Erro na abertura do arquivo !!!\n");
    exit(1);
  }
  while (fgets(ch, sizeof(ch), fp))
  {
    if (ch[0] != '#')
    {
      for (i=0;i<strlen(ch);i++)
      {
        sprintf(valor,"%c",ch[i]);
        if (strcmp(valor,"=") == 0)
        {
          x=i+1;
          i= strlen(ch);
          for (x;x<strlen(ch)-1;x++)
          {
            sprintf(aux,"%c",ch[x]);
            strcat(c2,aux);
        }
        }
        else
```

```
        {
          strcat(c1,valor);
        }
      }
    }

    if (strcmp(c1,"servidor") == 0)
    {
      strcpy(d_raiz->servidor,c2);
    } else if (strcmp(c1,"usuario") == 0)
    {
      strcpy(d_raiz->usuario,c2);
    } else if (strcmp(c1,"senha") == 0)
    {
      strcpy(d_raiz->senha,c2);
    }else if (strcmp(c1,"bd") == 0)
    {
      strcpy(d_raiz->bd,c2);
    }
  strcpy(c1,"");
  strcpy(c2,"");
  }
  fclose(fp);
}
```

10.1.4 LERCONFIG.H

```
/*
        NIDS - HYDRA

        LerConfig.h
*/

#include <stdio.h>

struct DADOS{
  char servidor[30];
  char usuario[30];
  char senha[20];
  char bd[20];
};

struct DADOS *d_raiz;

void LerConf();
```

10.1.5 REGRAS.C

```
/*
        NIDS - HYDRA

        Regras.c
```

```
*/

#include <stdio.h>
#include <string.h>
#include "AlertaCliente.h"
#include "Regras.h"

#define SIZE_TCP 58
#define SIZE_UDP 42

int gl_regras_tcp=0, gl_regras_udp=0;
char ch[1024];

// Declaracao das listas (TCP e UDP) utilizadas no sistema
lno *tcp_raiz;
lno *tcp_obj;
lno *tcp_aux;

lno *udp_raiz;
lno *udp_obj;
lno *udp_aux;

ln1 *ln1_raiz;
ln1 *ln1_obj;
ln1 *ln1_aux;

lvalor *lvalor_raiz;
lvalor *lvalor_obj;
lvalor *lvalor_aux;

ltcp_flag *ltcp_flag_obj;

ln1 *InsereN1(FILE *fp, int cont);
ltcp_flag *InsereFlags(char *flags_tcp);
void CriaListaRegras(char *id, char *ip_origem, char *porta_origem, char
*ip_destino, char *porta_destino, char *protocolo,char *flags_tcp, ln1*
ln1_aux);
int VerificarLista(ln1 *l, int base, u_char *packet);
void VerificarRegra(char *ip_origem, int porta_origem, char *ip_destino,
int porta_destino, char *protocolo, u_char *packet, char *flag_tcp);
int HexatoInt();

/*
 * Responsavel por "subir" as regras, ou seja, montar em memoria
 * tudo o que esta pre-definido no arquivo de regras
 */
void SubirRegras()
{
  tcp_raiz = NULL;
  udp_raiz = NULL;

  int i;
  int pos=0;
  char valor[1024];

  ln1 *aux;

  char id[30];
  char ip_origem[30];
  char porta_origem[6];
  char ip_destino[30];
```

```
  char porta_destino[30];
  char protocolo[30];
  char flags_tcp[8];

  strcpy(id,"");
  strcpy(ip_origem,"");
  strcpy(ip_destino,"");
  strcpy(porta_origem,"");
  strcpy(porta_destino,"");
  strcpy(protocolo,"");
  strcpy(flags_tcp,"");

  FILE *fp;

  // Responsavel pela abertura do arquivo de regras, por default
hydra.regras
  if ((fp = fopen("hydra.regras","r")) == NULL)
  {
    printf("Erro na abertura do arquivo de Regras - hydra.regras !!!\n");
    exit(1);
  }

/*
 * Nesta parte, e feita a leitura do arquivo de regras ate o campo que
antecede o
 * campo de dados
 */
  while (fgets(ch, sizeof(ch), fp))
  {
    if (ch[0] != '#')
    {
      for (i=0;i<strlen(ch);i++)
      {
        sprintf(valor,"%c",ch[i]);
        if ((strcmp(valor,";") == 0) || (strcmp(valor,"\n") == 0))
        {
          if ((strcmp(valor,"\n") == 0))
          {
          pos=0;
          }
        else
        {
            pos++;
          }
        }
        else
        {
          if (pos  ==  0)
          {
            strcat(id,valor);
          }
          else if (pos == 1)
        {
            strcat(ip_origem,valor);
        }
          else if (pos == 2)
        {
            strcat(porta_origem,valor);
        }
          else if (pos == 3)
        {
```

```
            strcat(ip_destino,valor);
        }
          else if (pos == 4)
        {
            strcat(porta_destino,valor);
        }
          else if (pos == 5)
        {
            strcat(protocolo,valor);
        }
          else if (pos == 6)
        {
            strcat(flags_tcp,valor);
        }
          else if (pos == 7)
        {
          pos=0;

            if ( valor == "!" )
              aux = NULL;
            else
            /*
               * Neste ponto, e declarado um auz, onde o mesmo recebe
recursivamente
             * o aninhamento das regras de campo, caso existam
             */
              aux = InsereN1(fp, i);

          // Este break tem como funcao a nao continuacao de leitura do
ponteiro do arquivo (fp)
          break;
        }
        }
      }
  /*
   * Neste ponto e chamado para cada linha lida, a funcao, sendo passado
todos os
   * parametros necessarios para a montagem da lista em memoria
   */

CriaListaRegras(id,ip_origem,porta_origem,ip_destino,porta_destino,protocol
o,flags_tcp,aux);

      // Abaixo temos todas as variaveis locais sendo zeradas para evitar
qualquer tipo de lixo
      strcpy(id,"");
      strcpy(ip_origem,"");
      strcpy(ip_destino,"");
      strcpy(porta_origem,"");
      strcpy(porta_destino,"");
      strcpy(protocolo,"");
      strcpy(flags_tcp,"");
    }
  }
  // Descritor do arquivo sendo fechado
  fclose(fp);
}

// Responsavel pela insercao de novos elementos na lista encadeada
ln1 *InsereN1(FILE *fp, int cont)
{
```

```
      if ( ch[cont] == '\n' )
      {
        return NULL;
      }
      else
        {
        int i, x, pos=0;
        char aux1[10];
        char aux_hexa[10];
        char valor_hexa[10];
        char valor_str[10];
        char offset[10];
        char size[10];
        char valor_t1[10];
        int flag_hexa=0;
        int flag_str=0;
        int valor_int;
        strcpy(offset,"");
        strcpy(size,"");
        strcpy(valor_hexa,"");
        strcpy(valor_str,"");

        // Definicao do tipo
        ln1 *ln1_obj;

        // Alocacao da area em memoria
        ln1_obj = (ln1 *)malloc(sizeof(ln1));

        // Registros prox e valor sendo setados para NULL "default"
        ln1_obj->prox = NULL;
        ln1_obj->valor = NULL;

        // Codigo responsavel por coletar o offset e size
        for (i=cont;i<strlen(ch);i++)
          {
          sprintf(aux1,"%c",ch[i]);

          if (strcmp(aux1,",") == 0)
          {
            pos++;
          }
          else if (pos == 0)
          {
            strcat(offset,aux1);
            ln1_obj->offset = atoi(offset);
          }
          else if (pos == 1)
          {
            strcat(size,aux1);
            ln1_obj->size = atoi(size);
          }
          else if (pos == 2)
            {
            if (strcmp(aux1,";") == 0)
            {
              pos=0;
            strcpy(offset,"");
            strcpy(size,"");

            // Responsavel pela Recursao - verifica as regras secundarias
...
```

```
            ln1_obj->prox = InsereN1(fp,i+1);
              break;
            }
            else
              {
            /*
             * Responsavel por verificar a gramatica utilizada nas regras,
ou seja,
             * o que estiver entre {} representa HEXA e o que estiver entre
[]
             * representa caracter
             */
              if (strcmp(aux1,"{") == 0)
            {
                flag_hexa = 1;
            }

              if (strcmp(aux1,"}") == 0)
            {
                flag_hexa = 0;
            }

              if (strcmp(aux1,"[") == 0)
            {
                flag_str = 1;
            }

              if (strcmp(aux1,"]") == 0)
            {
                flag_str = 0;
            }

              if ((strcmp(aux1,"{") != 0) && (strcmp(aux1,"}") != 0) &&
(flag_hexa == 1))
            {
              strcat(valor_hexa,aux1);
            }
            if (strcmp(aux1,"}") == 0)
            {
              lvalor_obj = (lvalor *)malloc(sizeof(lvalor));
              lvalor_obj->prox = NULL;

              // StrtoInt - Responsavel por converter um caracter em seu
cod char
              lvalor_obj->valor = StrtoInt(valor_hexa);

              /*
               * Responsavel pela verificacao da lista, caso vazio, sera o
primeiro
               * caso contrario, sera percorrido a lista ate o ultimo
elemento e
               * adicionado ao final da mesma
               */
              if ( ln1_obj->valor == NULL )
              {
                ln1_obj->valor = lvalor_obj;
              }
              else
              {
                lvalor_aux = ln1_obj->valor;
```

```
                while ( lvalor_aux->prox != NULL )
                  lvalor_aux =  lvalor_aux->prox;

                    lvalor_aux->prox = lvalor_obj;
              }

              strcpy(valor_hexa,"");
                }

              if ((strcmp(aux1,"[") != 0) && (strcmp(aux1,"]") != 0) &&
flag_str == 1)
            {
              valor_int = (int)(unsigned char)aux1[0];

              lvalor_obj = (lvalor *)malloc(sizeof(lvalor));
              lvalor_obj->prox = NULL;

              lvalor_obj->valor = valor_int;

              if ( ln1_obj->valor == NULL )
              {
                ln1_obj->valor = lvalor_obj;
              }
              else
              {
                lvalor_aux = ln1_obj->valor;

                while ( lvalor_aux->prox != NULL )
                  lvalor_aux =  lvalor_aux->prox;

                    lvalor_aux->prox = lvalor_obj;
              }
            }
            if (strcmp(aux1,"]") == 0)
            {
              flag_str == 0;
                }
              }
            }
      }
  return ln1_obj;
  }
}

// Responsavel por informar quais os flags estao habilitados nos pacotes
TCP
ltcp_flag *InsereFlags(char *flags_tcp)
{
  int qtde = strlen(flags_tcp), i;

  ltcp_flag_obj=(ltcp_flag *)malloc(sizeof(ltcp_flag));

  ltcp_flag_obj->fin = 0;
  ltcp_flag_obj->syn = 0;
  ltcp_flag_obj->rst = 0;
  ltcp_flag_obj->psh = 0;
  ltcp_flag_obj->ack = 0;
  ltcp_flag_obj->urg = 0;
  ltcp_flag_obj->res1 = 0;
  ltcp_flag_obj->res2 = 0;
```

```
  for(i=0;i<qtde;i++)
  {
    if(flags_tcp[i] == 'F')
      ltcp_flag_obj->fin = 1;
    if(flags_tcp[i] == 'S')
      ltcp_flag_obj->syn = 1;
    if(flags_tcp[i] == 'R')
      ltcp_flag_obj->rst = 1;
    if(flags_tcp[i] == 'P')
      ltcp_flag_obj->psh = 1;
    if(flags_tcp[i] == 'A')
      ltcp_flag_obj->ack = 1;
    if(flags_tcp[i] == 'U')
      ltcp_flag_obj->urg = 1;
    if(flags_tcp[i] == '1')
      ltcp_flag_obj->res1 = 1;
    if(flags_tcp[i] == '2')
      ltcp_flag_obj->res2 = 1;
    if(flags_tcp[i] == '!')
      ltcp_flag_obj->nenhum = 1;
  }
  return ltcp_flag_obj;
}

// Resposanvel por criar as listas (TCP e UDP) em memoria
void CriaListaRegras(char *id, char *ip_origem, char *porta_origem, char
*ip_destino, char *porta_destino, char *protocolo,char *flags_tcp, ln1*
ln1_aux)
{
  // Testando o tipo de protocolo
  if (strcmp(protocolo,"TCP") == 0)
  {
    tcp_obj=(lno *)malloc(sizeof(lno));

  /*
   * Responsavel pela verificacao da lista, caso vazio, sera o primeiro
   * caso contrario, sera percorrido a lista ate o ultimo elemento e
   * adicionado ao final da mesma
   */
    if (tcp_raiz == NULL)
    {
      strcpy(tcp_obj->id,id);
      strcpy(tcp_obj->ip_origem,ip_origem);
      strcpy(tcp_obj->porta_origem,porta_origem);
      strcpy(tcp_obj->ip_destino,ip_destino);
      strcpy(tcp_obj->porta_destino,porta_destino);
      strcpy(tcp_obj->protocolo,protocolo);

      tcp_obj->dados = ln1_aux;

      tcp_obj->prox = NULL;
      tcp_obj->flags = InsereFlags(flags_tcp);
      tcp_raiz = tcp_obj;
    }
    else
    {
      tcp_aux = tcp_raiz;
      while (tcp_aux->prox != NULL)
        tcp_aux = tcp_aux->prox;

      tcp_aux->prox = tcp_obj;
```

```
    strcpy(tcp_obj->id,id);
    strcpy(tcp_obj->ip_origem,ip_origem);
    strcpy(tcp_obj->porta_origem,porta_origem);
    strcpy(tcp_obj->ip_destino,ip_destino);
    strcpy(tcp_obj->porta_destino,porta_destino);
    strcpy(tcp_obj->protocolo,protocolo);

    tcp_obj->dados = ln1_aux;

    tcp_obj->prox = NULL;
    tcp_obj->flags = InsereFlags(flags_tcp);
  }
}
// Testando o tipo de protocolo
else if (strcmp(protocolo,"UDP") == 0)
{
  udp_obj=(lno *)malloc(sizeof(lno));

/*
 * Responsavel pela verificacao da lista, caso vazio, sera o primeiro
 * caso contrario, sera percorrido a lista ate o ultimo elemento e
 * adicionado ao final da mesma
 */
  if (udp_raiz == NULL)
  {
    strcpy(udp_obj->id,id);
    strcpy(udp_obj->ip_origem,ip_origem);
    strcpy(udp_obj->porta_origem,porta_origem);
    strcpy(udp_obj->ip_destino,ip_destino);
    strcpy(udp_obj->porta_destino,porta_destino);
    strcpy(udp_obj->protocolo,protocolo);

    udp_obj->dados = ln1_aux;

    udp_obj->prox = NULL;
    udp_obj->flags = NULL;
    udp_raiz = udp_obj;
  }
  else
  {
    udp_aux = udp_raiz;
    while (udp_aux->prox != NULL)
      udp_aux = udp_aux->prox;

    udp_aux->prox = udp_obj;

    strcpy(udp_obj->id,id);
    strcpy(udp_obj->ip_origem,ip_origem);
    strcpy(udp_obj->porta_origem,porta_origem);
    strcpy(udp_obj->ip_destino,ip_destino);
    strcpy(udp_obj->porta_destino,porta_destino);
    strcpy(udp_obj->protocolo,protocolo);

    udp_obj->dados = ln1_aux;

    udp_obj->prox = NULL;
    udp_obj->flags = NULL;
  }
}
else
  printf("Protocolo %s nao testado \n", protocolo);
```

```
}

// Responsavel por verificar a lista, de acordo com o pacote corrente
int VerificarLista(ln1 *l, int base, u_char *packet)
{
  if (l == NULL)
  {
    return 0;
  }
  else
  {
    lvalor *lvalor_aux = l->valor;
    int inicio = base + l->offset;
    int i=0, value=0;

    for(i=0;i<l->size;i++)
    {
      // Responsavel por comparar os valores da lista com o pacote
      if (HexatoInt(packet[inicio + i]) != lvalor_aux->valor)
      {
        value=1;
        break;
      }

      lvalor_aux = lvalor_aux->prox;
    }

    if (value == 1)
      return 1;
    else
      if (inicio == base)
        return VerificarLista(l->prox,base+l->offset,packet);
      else
        return VerificarLista(l->prox,base+l->offset+l->size,packet);
  }
}

// Responsavel por verificar as flags (TCP)
int VerificarFlags(char *flag_tcp, lno *l)
{
  int i, valor_flag = 0;
  int qtde = strlen(flag_tcp);


  for (i=0;i<qtde;i++)
  {
    if (l->flags->nenhum == 1)
    {
      return valor_flag;
    }

    if(flag_tcp[i] == 'A')
      if (l->flags->ack != 1)
      {
        valor_flag=1;
        break;
      }
    if(flag_tcp[i] == 'F')
      if (l->flags->fin != 1)
      {
        valor_flag=1;
```

```
        break;
      }
    if(flag_tcp[i] == 'S')
      if (l->flags->syn != 1)
      {
        valor_flag=1;
        break;
      }
    if(flag_tcp[i] == 'U')
      if (l->flags->urg != 1)
      {
        valor_flag=1;
        break;
      }
    if(flag_tcp[i] == 'P')
      if (l->flags->psh != 1)
      {
        valor_flag=1;
        break;
      }
    if(flag_tcp[i] == 'R')
      if (l->flags->rst != 1)
      {
        valor_flag=1;
        break;
      }
    if(flag_tcp[i] == '1')
      if (l->flags->res1 != 1)
      {
        valor_flag=1;
        break;
      }
    if(flag_tcp[i] == '2')
      if (l->flags->res2 != 1)
      {
        valor_flag=1;
        break;
      }
  }
  return valor_flag;
}

/*
 * Resposanvel por Verificar os campos da regra, como IPs e Portas (TCP e
UDP)
 * de forma independente, caso os campos basicos estejam de acordo, a
verificacao
 * continua nos campos de dados e flags, quando necessario
 */
void VerificarRegra(char *ip_origem, int porta_origem, char *ip_destino,
int porta_destino, char *protocolo, u_char *packet, char *flag_tcp)
{
  int cont_rules=0, cont_pct=0, id_aux;
  char aux1[30];

  time_t dh;
  char DATA[30], comando[1024];
  dh = time(0);
  strftime(DATA,30,"%Y-%m-%d", localtime(&dh));

  lno *lista;
```

```
  if (strcmp(protocolo,"TCP") == 0)
  {
    lista = tcp_raiz;

    while(lista != NULL)
    {
      if (strcmp(lista->ip_origem,"ALL") != 0)
      {
        cont_rules++;
        if (strcmp(lista->ip_origem,ip_origem) == 0)
          cont_pct++;
      }

      if (strcmp(lista->porta_origem,"ALL") != 0)
      {
        cont_rules++;
        sprintf(aux1,"%d",porta_origem);
        if (strcmp(lista->porta_origem,aux1) == 0)
          cont_pct++;
      }

      if (strcmp(lista->ip_destino,"ALL") != 0)
      {
        cont_rules++;
        if (strcmp(lista->ip_destino,ip_destino) == 0)
          cont_pct++;
      }

      if (strcmp(lista->porta_destino,"ALL") != 0)
      {
        cont_rules++;
        sprintf(aux1,"%d",porta_destino);
        if (strcmp(lista->porta_destino,aux1) == 0)
          cont_pct++;
      }

      // Comparacao das Regras Basicas
      if (cont_rules == cont_pct)
      {
        // Comparacao dos Flags
        if (VerificarFlags(flag_tcp, lista) == 0)
        {
        // Comparacao na lista
          if (VerificarLista(lista->dados,SIZE_TCP,packet) == 0)
          {
            printf("REGRA DETECTADA\nID = %s\n\n", lista->id);

          // Insercao da tentativa em Banco de Dados
            InsereBD(lista-
>id,ip_origem,porta_origem,ip_destino,porta_destino,protocolo,DATA);
            sprintf(comando,"iptables -A INPUT -s %s -p %s --dport %d -j
DROP", ip_origem,protocolo,porta_destino);

          // Envio da regra via BroadCast para todos os clientes ativos na
rede
            EnviaRegra(comando);
          }
        }
      }
      lista = lista->prox;
```

```
      cont_rules=0;
      cont_pct=0;
    }
  }

  if (strcmp(protocolo,"UDP") == 0)
  {
    lista = udp_raiz;

    while(lista != NULL)
    {
      if (strcmp(lista->ip_origem,"ALL") != 0)
      {
        cont_rules++;
        if (strcmp(lista->ip_origem,ip_origem) == 0)
          cont_pct++;
      }

      if (strcmp(lista->porta_origem,"ALL") != 0)
      {
        cont_rules++;
        sprintf(aux1,"%d",porta_origem);
        if (strcmp(lista->porta_origem,aux1) == 0)
          cont_pct++;
      }

      if (strcmp(lista->ip_destino,"ALL") != 0)
      {
        cont_rules++;
        if (strcmp(lista->ip_destino,ip_destino) == 0)
          cont_pct++;
      }

      if (strcmp(lista->porta_destino,"ALL") != 0)
      {
        cont_rules++;
        sprintf(aux1,"%d",porta_destino);
        if (strcmp(lista->porta_destino,aux1) == 0)
          cont_pct++;
      }
      // Comparacao das Regras Basicas
      if (cont_rules == cont_pct)
      {
        // Comparacao na lista
        if (VerificarLista(lista->dados,SIZE_UDP,packet) == 0)
        {
          printf("REGRA DETECTADA\nID = %s\n\n", lista->id);

          // Insercao da tentativa em Banco de Dados
        InsereBD(lista-
>id,ip_origem,porta_origem,ip_destino,porta_destino,protocolo,DATA);
          sprintf(comando,"iptables -A INPUT -s %s -p %s --dport %d -j
DROP", ip_origem,protocolo,porta_destino);

        // Envio da regra via BroadCast para todos os clientes ativos na
rede
          EnviaRegra(comando);
        }
      }
```

```
      lista = lista->prox;

      cont_rules=0;
      cont_pct=0;
    }
  }
}

// Responsavel pela conversao de Hexa para Inteiro
int HexatoInt(u_char *packet)
{
  char aux[10],c1[10],c2[10];
  int v1=0, v2=0, valor=0;
  strcpy(c1,"");
  strcpy(c2,"");

  sprintf(aux,"%x", packet);

  sprintf(c1,"%c",aux[0]);
  sprintf(c2,"%c",aux[1]);

  if (strlen(aux) == 2)
  {
    if (isdigit(aux[0]) == 0)
    {
      switch(aux[0])
      {
        case 'a': v1 = 10;
                  break;
        case 'b': v1 = 11;
                  break;
        case 'c': v1 = 12;
                  break;
        case 'd': v1 = 13;
                  break;
        case 'e': v1 = 14;
                  break;
        case 'f': v1 = 15;
                  break;
      }
    }
    else
      v1 = atoi(c1);


    if (isdigit(aux[1]) == 0)
    {
      switch(aux[1])
      {
        case 'a': v2 = 10;
                  break;
        case 'b': v2 = 11;
                  break;
        case 'c': v2 = 12;
                  break;
        case 'd': v2 = 13;
                  break;
        case 'e': v2 = 14;
                  break;
        case 'f': v2 = 15;
                  break;
```

```
      }
    }
    else
      v2 = atoi(c2);

    valor = (v1 * 16) + (v2 * 1);

  }
  else if (isdigit(aux[0]) == 0)
       {
         switch(aux[0])
         {
           case 'a': v2 = 10;
                break;
           case 'b': v2 = 11;
                break;
           case 'c': v2 = 12;
                break;
           case 'd': v2 = 13;
                break;
           case 'e': v2 = 14;
                break;
           case 'f': v2 = 15;
                break;
         }
         valor = v2;
       }
       else
         valor = atoi(c1);

  return valor;
}
```

10.1.6 REGRAS.H

```
/*
        NIDS - HYDRA

        Regras.h
*/

#include <stdio.h>
#include <string.h>
#include "AlertaCliente.h"

struct NO {
  char id[30];
  char ip_origem[30];
  char porta_origem[30];
  char ip_destino[30];
  char porta_destino[30];
  char protocolo[30];
  struct NO *prox;
  struct N1 *dados;
  struct TCP_FLAG *flags;
};
```

```
struct N1 {
  int offset;
  int size;
  struct N1 *prox;
  struct VALOR *valor;
};

struct VALOR {
  int valor;
  struct VALOR *prox;
};

struct TCP_FLAG {
    int fin;
    int syn;
    int rst;
    int psh;
    int ack;
    int urg;
    int res1;
    int res2;
    int nenhum;
};

typedef struct NO lno;
typedef struct N1 ln1;
typedef struct VALOR lvalor;
typedef struct TCP_FLAG ltcp_flag;

void SubirRegras();
int VerificarFlags(char *flag_tcp, lno *l);
```

10.1.7 INSERIR.C

```
/*
        NIDS - HYDRA

        Inserir.c
*/

#include <stdio.h>
#include <mysql/mysql.h>
#include "Inserir.h"
#include "LerConfig.h"

char id[10], ip_origem[15], ip_destino[15], dados[30] , data[10];
int port_origem, port_destino;

int  tipo_protocol;
char *error;

/*
 * Responsavel por inserir os dados em Banco de Dados (MySQL) de acordo
 * com os parametros passados
 */
```

```
int InsereBD(char *id, char *ip_origem, int port_origem, char *ip_destino,
int port_destino, char *tipo_protocol, char *data)
{
  MYSQL conexao;
  int res;
  mysql_init(&conexao);

  if (mysql_real_connect(&conexao,d_raiz->servidor,d_raiz->usuario,d_raiz-
>senha,d_raiz->bd,0,NULL,0))
  {
    char *query = "INSERT INTO db_alert(id,ip_origem, port_origem,
ip_destino, port_destino, tipo_protocol, data)
values('%s','%s','%d','%s','%d','%s','%s')";
    char *sql;
    int estado;

    sql = (char *)malloc(255 * sizeof(char));
    error = (char *)NULL;

    // Faz a escrita no banco de dados
    sprintf(sql, query, id, ip_origem, port_origem, ip_destino,
port_destino, tipo_protocol, data);

    estado = mysql_query(&conexao, sql);
    free(sql);

    // Insere dados na tabela
    if (res)
      printf("Registros inseridos %d\n",
(int)mysql_affected_rows(&conexao));
    else
      printf("Erro na inserção %d : %s\n", mysql_errno(&conexao),
mysql_error(&conexao));

     mysql_close(&conexao);
  }
  else
  {
    printf("Falha de conexao\n");
    printf("Erro %d : %s\n", mysql_errno(&conexao), mysql_error(&conexao));
  }
}
```

10.1.8 INSERIR.H

```
/*
        NIDS - HYDRA

        Inserir.h
*/

#include <stdio.h>
#include <mysql/mysql.h>

struct DADOS *d_raiz;
```

```
int InsereBD(char *id, char *ip_origem, int port_origem, char *ip_destino,
int port_destino, char *tipo_protocol, char *data);
```

10.1.9 ALERTACLIENTE.C

```
/*
      NIDS - HYDRA

      AlertaCliente.c
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define SIZE 1024

/*
 * Responsavel por enviar o alerta via BroadCast, utilizando a porta 5000
 */
EnviaRegra(char *comando)
{
  struct sockaddr_in me, server;                // Estrutura socket
  int sock=socket(AF_INET,SOCK_DGRAM,0);             // Define tipo de
socket
  int adl=sizeof(me), value=1;

  setsockopt(sock,SOL_SOCKET,SO_BROADCAST,&value,sizeof(value)); // Define
o socket tipo broadcast
  bzero((char *)&me,adl);

  me.sin_family=AF_INET;                        // Define o tipo de familia
socket
  me.sin_addr.s_addr=htonl(INADDR_ANY);         // Endereco IP local
  me.sin_port=htons(0);                         // Porta local (0=auto
assign)

  bind(sock,(struct sockaddr *)&me,adl);
  bzero((char *)&server,adl);

  server.sin_family=AF_INET;                    // Define o tipo de familia
socket
  server.sin_addr.s_addr=inet_addr("255.255.255.255");     // Broadcast

  // No caso mais corrente (rede de classe C pode usar o endereco
255.255.255.255
  server.sin_port=htons(5000);                        // Porta do servidor

  // Envia comando com ou sem parametros
  sendto(sock,comando,SIZE,0,(struct sockaddr *)&server,adl);

  close(sock);
}
```

10.1.10 ALERTACLIENTE.H

```
/*
        NIDS - HYDRA

        AlertaCliente.h
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int EnviaRegra(char *comando);
```

10.1.11 GERAL.C

```
/*
        NIDS - HYDRA

        Geral.c
*/

// Responsavel pela conversao de caracter para inteiro
int StrtoInt(char *aux)
{
  char c1[10],c2[10];
  int v1=0, v2=0, valor=0;

  if (strlen(aux) == 2)
  {
    if (isdigit(aux[0]) == 0)
    {
      switch(aux[0])
      {
        case 'a': v1 = 10;
                  break;
        case 'b': v1 = 11;
                  break;
        case 'c': v1 = 12;
                  break;
        case 'd': v1 = 13;
                  break;
        case 'e': v1 = 14;
                  break;
        case 'f': v1 = 15;
                  break;
      }
    }
```

```
    else
      v1 = (unsigned char)(aux[0]) - 0x30;


    if (isdigit(aux[1]) == 0)
    {
      switch(aux[1])
      {
        case 'a': v2 = 10;
                  break;
        case 'b': v2 = 11;
                  break;
        case 'c': v2 = 12;
                  break;
        case 'd': v2 = 13;
                  break;
        case 'e': v2 = 14;
                  break;
        case 'f': v2 = 15;
                  break;
      }
    }
    else
      v2 = (unsigned char)(aux[1]) - 0x30;

    valor = (v1 * 16) + (v2 * 1);

  }
  else if (isdigit(aux[0]) == 0)
       {
         switch(aux[0])
         {
           case 'a': v2 = 10;
                break;
           case 'b': v2 = 11;
                break;
           case 'c': v2 = 12;
                break;
           case 'd': v2 = 13;
                break;
           case 'e': v2 = 14;
                break;
           case 'f': v2 = 15;
                break;
         }
         valor = v2;
       }
       else
         valor = (unsigned char)(aux[0]) - 0x30;


  return valor;
}
```

10.1.12 GERAL.H

```
/*
        NIDS - HYDRA

        Geral.h
*/

#include "Geral.c"

int StrtoInt(char *aux);
```

10.1.13 AGENT.C

```
/*
        NIDS - HYDRA

        agent.c
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define SIZE 1024

struct no {
  char regra[SIZE];
  struct no *prox;
};

typedef struct no lregra;

lregra *raiz;
lregra *obj;
lregra *aux;

int ChecaRegra();

CargaCliente()
{
  raiz = NULL;
  obj = NULL;
  aux = NULL;


  struct sockaddr_in me, client;  /* estrutura socket */
  int sock=socket(AF_INET,SOCK_DGRAM,0); /* define o tipo de socket */
  int adl=sizeof(me);
  char comando[SIZE];

  bzero((char *)&me,adl);

  me.sin_family=AF_INET;   /* define o tipo de familia de socket */
  me.sin_addr.s_addr=htonl(INADDR_ANY); /* endereco IP local */
```

```
  me.sin_port=htons(5000); /* porta local  */

  if(-1!=bind(sock,(struct sockaddr *)&me,adl)) /* resolução de nomes */
    do
    {
      recvfrom(sock,comando,SIZE,0,(struct sockaddr *)&client,&adl); /*
recebe o comando */

      if (raiz == NULL)
      {
        obj = (lregra *)malloc(sizeof(lregra));
        strcpy(obj->regra,comando);
        obj->prox = NULL;

        raiz = obj;
        system(comando);     /* executa o comando */
        printf("ALERTA ENVIADO -> %s \n", comando);
      }
      else
      {
        if (ChecaRegra(comando) == 0)
        {
          aux = raiz;

          obj = (lregra *)malloc(sizeof(lregra));
          strcpy(obj->regra,comando);
          obj->prox = NULL;

        while (aux->prox != NULL)
            aux = aux->prox;

          aux->prox = obj;

          system(comando);     /* executa o comando */
        }
      }
    } while(1);
  else
    puts("Porta ocupada");

  close(sock);
}

int ChecaRegra(char *comando)
{
  aux = raiz;

  int valor = 0;

  while(aux != NULL)
  {
    if(strcmp(aux->regra,comando) == 0)
    {
      valor = 1;
    }
    aux = aux->prox;
  }
  return valor;
}

main()
```

```
{
  CargaCliente();
}
```

10.2 ARQUIVO DE CONFIGURAÇÃO – HYDRA.CONF

```
# Arquivo de Configuracao
# Defina abaixo as informacoes corretas referente
# ao banco de dados.
servidor=localhost
usuario=pf
senha=pf00
bd=PF
```

10.3 ARQUIVO DE REGRAS – HYDRA.REGRAS

```
# Arquivo de Regras do NIDS Hydra
#
# Todas as linhas comecando com # estarao em comentario
#
# Quanto a Sintaxe do arq, segue abaixo
#
# ID da Regra ; IP_Origem ; Porta_Origem ; IP_Destino ; Porta_Destino ;
Tipo_Protocolo ; Flags TCP ; Dados
#
# Obs.: Para representacao de TODOS, utilizaremos a mascara ALL
#
1;ALL;ALL;ALL;161;UDP;16,3,{ff}{ff}{8a};14,1,{ac};21,1,{aa};
2;ALL;ALL;ALL;ALL;TCP;F;!;
3;ALL;ALL;ALL;111;UDP;;12,4,{00}{01}{86}{a0};4,4,{00}{00}{00}{03};16,4,{00}
{01}{86}{a5};8,4,{00}{00}{00}{00};
4;ALL;ALL;ALL;69;UDP;0,2,{00}{01};2,6,[passwd];
5;ALL;ALL;ALL;69;UDP;2,2,[..];
```

10.4 BANCO DE DADOS

```
CREATE DATABASE `PF`;
USE `PF`;

CREATE TABLE `db_alert` (
  `cod` int(4) NOT NULL auto_increment,
  `id` varchar(10) NOT NULL default '',
  `ip_origem` varchar(15) default NULL,
  `port_origem` varchar(5) default NULL,
  `ip_destino` varchar(15) default NULL,
  `port_destino` varchar(5) default NULL,
  `tipo_protocol` char(3) default NULL,
  `data` date default NULL,
```

```
  PRIMARY KEY  (`cod`)
) TYPE=InnoDB;


CREATE TABLE `db_desc` (
  `id` int(4) NOT NULL default '0',
  `sumario` varchar(200) default NULL,
  `impacto` varchar(200) default NULL,
  `info_detalhado` varchar(200) default NULL,
  `cenario_ataque` varchar(200) default NULL,
  `facilidade_ataque` varchar(200) default NULL,
  `falso_positivo` varchar(100) default NULL,
  `falso_negativo` varchar(100) default NULL,
  `acao_corretiva` varchar(200) default NULL,
  `contribuicao` varchar(200) default NULL,
  `referencia` varchar(200) default NULL
) TYPE=InnoDB;
```