CENTRO PAULA SOUZA

GOVERNO DO ESTADO SÃO PAULO

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Análise de Sistemas e Tecnologia da Informação

Carlos Eduardo Quinholi

ESTUDO COMPARATIVO ENTRE FIREWALLS EXECUTANDO EM LINUX SLAX E FREEBSD

Americana, SP
2013

CENTRO PAULA SOUZA
GOVERNO DO ESTADO SÃO PAULO

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Análise de Sistemas e Tecnologia da Informação

Carlos Eduardo Quinholi

ESTUDO COMPARATIVO ENTRE FIREWALLS EXECUTANDO EM LINUX SLAX E FREEBSD

Trabalho monográfico, desenvolvido em cumprimento à exigência curricular do Curso Superior de Análise de Sistemas e Tecnologia da Informação da Fatec Americana, sob orientação do Prof. Dr. José Luis Zem.

Área de concentração: Segurança da Informação.

Americana
2013

FICHA CATALOGRÁFICA elaborada pela

BIBLIOTECA – FATEC Americana – CEETPS

Quinholi, Carlos Eduardo

Q63e Estudo comparativo entre Firewalls executando em Linux Slax e Freebsd. / Carlos Eduardo Quinholi. – Americana: 2013.

55f.

Monografia (Graduação em Análise de Sistemas e Tecnologia da Informação). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.

Orientador: Prof. Dr. José Luíz Zem

1. Segurança em sistemas de informação I. Zem, José Luiz II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.518.5

Bibliotecária responsável Ana Valquiria Niaradi – CRB-8 região 6203

Carlos Eduardo Quinholi

ESTUDO COMPARATIVO ENTRE FIREWALLS EXECUTANDO EM LINUX SLAX E FREEBSD

Trabalho monográfico, desenvolvido em cumprimento à exigência curricular do Curso Superior de Análise de Sistemas e Tecnologia da Informação da Fatec Americana, sob orientação do Prof. Dr. José Luíz Zem.

Área de concentração: Segurança da Informação.

Americana, 6 de Dezembro de 2013.

Banca Examinadora:

_____

Prof. José Luíz Zem (Orientador)
Doutor
FATEC Americana

_____

Prof. Alexandre Garcia Aguado (Membro)
Mestre
FATEC Americana

_____

Prof. Edson Roberto Gaseta (Membro)
Especialista
FATEC Americana

AGRADECIMENTOS

Primeiramente, agradeço a Deus por me dar forças nos momentos mais difíceis e sabedoria para solucionar os problemas.

À minha esposa, Helem, por me ajudar e sempre me apoiando no que fosse possível, não me deixando desanimar nos momentos difíceis.

Ao meu filho, Heitor, que é a minha maior alegria neste mundo.

Ao meu orientador, Prof. Dr. José Luíz Zem, e a todos que me ajudaram a dar mais um passo rumo à conquista dos meus sonhos.

DEDICATÓRIA

Dedico este trabalho a minha esposa Helem e a meu filho Heitor, por sempre estarem comigo, me dando apoio e força nos momentos difíceis.

RESUMO

Este trabalho apresenta um estudo comparativo entre os firewalls de Sistemas Operacionais livres distintos, diante de um ataque de DoS - Denial-of-Service. Para isso foi criada uma rede virtual no VMware Player com três servidores, um com o Slax GNU / Linux e outro com UNIX FreeBSD, e um terceiro servidor com GNU / Linux Ubuntu onde foi instalado o software de monitoramento de rede Zabbix para coleta de dados. Cada um foi configurado com as regras básicas de proteção. No Slax foi o *iptables*, e o *ipfirewall* no FreeBSD, ambos com regras para bloquear ameaças como *Port Scanner*, *Ping of Death*, *IP Spoofing*, e DoS. Estes testes foram realizados para testar a eficiência de cada um dos firewalls contra um ataque de DoS. No final deste trabalho serão apresentados os resultados obtidos.

Palavras chave: *Firewalls.* DoS. Segurança. *Iptables. Ipfirewall.*

ABSTRACT

This work presents a comparative study between firewalls of the free operating systems distinct, facing an attack of DoS - Denial-of-Service. For this was set up a virtual network on VMware Player with three servers, one with the GNU / Linux Slax and other with UNIX FreeBSD, and a third server with GNU / Linux Ubuntu where it was installed the software network monitoring Zabbix to collection data. Each it was configured with the basic rules of protection. On Slax were the iptables, and the ipfirewall in the FreeBSD, both with rules to block threats such as Port Scanner, Ping of Death, IP Spoofing and DoS. These tests were performed to test the efficiency of each of the firewalls against a DoS attack. At the end of this work will be presented the results obtained.

Keywords: Firewalls. DoS. Security. Iptables. Ipfirewall.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURA E SIGLAS

IBM – *International Business Machines*

IBSYS – Sistema Operacional desenvolvido pela IBM – IBSYS 7090/94

CI – Circuito Integrado

E/S – Entrada / Saída

LSI – *Large Scale Integration*, em português, Integrador de Larga Escala

AT&T – *American Telephone and Telegraph*

ISO – *International Organization for Standardization*, em português, Organização Internacional de Padronização

IEC – *International Eletrotechnical Commission*, em português, Comissão Eletrotécnica Internacional

ICMP – *Internet Control Message Protocol*, em português, Protocolo de Controle de Mensagens da Internet

IGMPv1 – *Internet Group Management Protocol v1*, em português, Protocolo de Gerenciamento de Grupos da Internet versão 1

IGMPv3 – *Internet Group Management Protocol v3*, em português, Protocolo de Gerenciamento de Grupos da Internet versão 3

TCP – *Transmission Control Protocol*, em português, Protocolo de Controle de Transmissão

EGP – *Exterior Gateway Protocol*, em português, Protocolo de Acesso Exterior

UDP – *User Datagram Protocol*, em português, Protocolo de Datagrama de Usuário

RIPv1 – *Routing Information Protocol v1*, em português, Protocolo de Informação de Roteamento versão 1

RIPv2 – *Routing Information Protocol v2*, em português, Protocolo de Informação de Roteamento versão 2

DCCP – *Datagram Congestion Control Protocol*, em português, Protocolo de Controle de Congestionamento de Datagramas

RSVP – *Resource reSerVation Protocol*, em português, Protocolo de Reserva de Recursos

GRE – *Generic Routing Encapsulation*, em português, Encapsulamento Genérico de Roteamento

IPSec – *Internet Protocol Security (AH / ESP)*, em português, Protocolo de Segurança de Internet

EIGRP – *Enhanced Interior Gateway Routing Protocol*, *)*, em português, Protocolo Avançado de Roteamento de Acesso Interno

OSPF – *Open Shortest Path First*, em português, Menor Rota Primeiro Aberta

SYN – Flag de sincronização de pacotes do protocolo TCP

ACK – Flag de contagem de pacotes do protocolo TCP

IPv4 – *Internet Protocol* v4, em português, Protocolo de Internet versão 4

IPv6 – *Internet Protocol* v6 (06-06-2012), em português, Protocolo de Internet versão 6

IPFW – *Ipfirewall*, *Firewall* nativo do FreBSD

DoS – *Denial-of-Service*, em português, Negação de Serviço

CPU – *Central Processing Unit*, em português, Unidade Central de Processamento

SUMÁRIO

1 – INTRODUÇÃO

Atualmente, devido às facilidades de acesso a Internet, esta não é mais um lugar muito seguro. O aumento do conhecimento e da informação tornou-se um fator importante para que as organizações investissem na segurança da informação tornando-a um pré-requisito para qualquer sistema de informações. Hoje, do ponto de vista dos administradores, o mundo está dividido claramente em dois grupos – "os dos bonzinhos e os dos bandidos", dai a necessidade de se implantar sistemas de segurança como *firewalls*, *Proxy*, IDS, IPS e outros sistemas de segurança, para a prevenção e proteção das informações contra ataques e invasões.

O **objetivo** deste estudo é comparar a segurança dos *firewalls* utilizados nos Sistemas Operacionais GNU/Linux e FreeBSD. No caso o GNU/Linux Slax utilizando o *iptables* e o FreeBSD utilizando o *ipfirewall*, e o que **justifica** este estudo, é demonstrar, através de testes, algumas vulnerabilidades apresentadas pelos *firewalls iptables* e *ipfirewall*, utilizados nos Sistemas Operacionais livres.

A **metodologia** utilizada consiste em construir um cenário, onde o usuário, utilizando-se de ferramentas de testes de *stress*, contidas no Ubuntu BackTrack 5R3, iniciará um ataque de Negação de Serviço – DoS, testando os *firewalls* a fim de paralisar os serviços do Servidor. Com isto será possível comparar qual *firewall* possui menor vulnerabilidade aos ataques.

Para desenvolver este estudo, foram utilizados *software*s livres. Para virtualização o VMware® Player, versão 5.0.2, para o Sistema Operacional invasor, o GNU/Linux Ubuntu BackTrack 5R3, para os *firewalls:* GNU/Linux Slax 6.0.7 como *firewall* 01 e FreeBSD 9.0 como *firewall* 02.

Os testes se darão com a utilização da ferramenta de teste de *stress* T50, com a finalidade de testar as barreiras de segurança instaladas em cada um dos *firewalls*. Os dados serão coletados dos Sistemas Operacionais Slax e FreeBSD, através do software de monitoração Zabbix Server.

2 – HISTORIA E EVOLUÇÃO DOS SISTEMAS OPERACIONAIS BASEADOS EM UNIX

Neste capitulo será abordada uma breve história do surgimento dos sistemas operacionais e como estes estão ligados a evolução dos computadores. Também será apresentado as histórias dos Sistemas Operacionais UNIX, GNU/Linux e FreeBSD.

2.1 – Surgimento dos Sistemas Operacionais

Os Sistemas Operacionais (Sistemas Operacionais) baseado em diversos autores da área e mais especificamente em Silberchatz (2000), estiveram intimamente associados à arquitetura dos computadores nos quais eles rodam. O primeiro computador digital foi projetado por volta da década de 1820 pelo matemático inglês Charles Babbage e instituído como motor analítico. Esta máquina era uma máquina analítica, pois se tratava de um equipamento mecânico, já que a tecnologia da época não permitia. Um fato relevante é que Babbage percebeu que uma máquina computadora deveria consistir de dispositivos de entrada, memória, unidade central de processamento e dispositivo de saída. Babbage então contratou uma jovem britânica, Ada Lovelace como programadora. Ada Lovelace foi considerada a primeira programadora do mundo e a linguagem de programação Ada é uma homenagem a ela.

A primeira geração de computadores (1945-55) marca a época das válvulas e painéis de conectores. Em meados da década de 40, um grupo de jovens teve sucesso na construção de máquinas de cálculo utilizando milhares de válvulas. Estes grandes computadores eram muito lentos e ocupavam salas inteiras. Para poder operar estas máquinas era necessário o conhecimento do funcionamento do equipamento como um todo, pois quem projetava era o mesmo que construía e o mesmo que programava e operava, mantendo-a em funcionamento. Nesta época ninguém conhecia ou tinha ouvido falar em Sistema Operacional e as linguagens de programação eram desconhecidas. Esta geração ficou conhecida como geração das válvulas e painéis de conectores (SILBERCHATZ, 2000).

A Segunda Geração (1955-63), marca a fase dos transistores e sistemas de lote. Com o desenvolvimento dos transistores, os computadores se tornaram mais rápidos e confiáveis em relação à primeira geração, passando a ser fabricados em serie e vendidos a clientes. Nesta geração, com o surgimento das primeiras linguagens de programação, os programas deixaram de ser feitos diretamente no *hardware*, facilitando assim o processo de desenvolvimento de

programas, houve a distinção clara entre projetistas, construtores, operadores, programadores e o pessoal de manutenção. As máquinas eram armazenadas em salas com ar condicionado e operadas por equipes especiais. Para executar um programa ou conjunto de programas (*jobs*), o programador escrevia seu programa em papel em FORTRAN ou Assembly, depois transformava este em cartões perfurados para serem levados à sala onde se encontrava os operadores (SILBERCHATZ, 2000).

Os cartões eram passados para as fitas magnéticas para serem lidas pelo computador, o qual executava um programa de cada vez, gravando o resultado em outra fita magnética de saída que era levada para a sala de saída e ficava a disposição do programador. Neste processo perdia-se muito tempo de processamento e deslocamento dos operadores. Para reduzir este tempo, os operadores juntavam vários *Jobs* na sala de entrada e depois levavam para serem processados em computadores menores e mais baratos rodando Sistemas Operacionais típicos como FMS (FORTRAN *Monitor System*) e o IBSYS da IBM. Este tipo de processamento ficou conhecido como processamento em lotes ou *batch* (SILBERCHATZ, 2000).

A Terceira Geração (1965-80) foi marcada pelo surgimento dos computadores com circuitos integrados (CIs) e pela multiprogramação, com isto, o preço dos computadores foi diminuindo consideravelmente, possibilitando assim a sua aquisição por empresas. Nesta geração houve um aumento do processamento e o tamanho das máquinas teve uma grande redução. A maioria dos fabricantes possuíam duas linhas de produtos totalmente incompatíveis sendo a primeira, os computadores científicos baseados em palavras e a segunda, os comerciais, baseados em caracteres. Produzir estas duas linhas de produtos ficava muito caro, então a IBM lançou o System/360 com duas versões compatíveis o 1401 e o 7094, onde estes diferiam só no preço e no desempenho. Porem para atender todas as aplicações e periféricos disponíveis, a IBM teve que desenvolver um Sistema Operacional o OS/360. Este Sistema Operacional era muito complexo e grande, feito em assembler com milhões de linhas de programação escrita por milhares de programadores, seus *bugs* exigiam versões periódicas para corrigi-los. Apesar de todos os problemas, o OS/360 e os Sistemas Operacionais semelhantes atenderam a maioria dos seus clientes razoavelmente bem. Eles contribuíram com o desenvolvimento de varias técnicas-chaves em Sistema Operacional de segunda geração, sendo a mais importante delas a multiprogramação. A multiprogramação permitiu dividir a memoria em partições, permitindo que vários *Jobs* fossem carregados em cada partição. Enquanto um *job* esperava alguma operação de entrada/saída (E/S), o processador poderia processar outro *job*, reduzindo assim o tempo ocioso do processador (SILBERCHATZ, 2000).

Outro importante recurso da terceira geração descrito por Silberchatz (2000) foi à capacidade de ler *Jobs* de cartões para disco. Assim, sempre que um *job* terminasse sua execução, o Sistema Operacional carrega um novo *job* do disco para a partição da memória e executa-o. Esta técnica é chamada *spooling* (sigla de *simultaneous peripheral operation online* - operação periférica simultânea online). Entretanto, os Sistemas Operacionais ainda se comportavam como sistemas em lote e não exigiam interação com o usuário, então, muitos programadores sentiam falta das máquinas de primeira geração, que eram disponibilizadas por completa para eles e, assim, podiam depurar seus programas.

A evolução da multiprogramação proporcionou aos usuários, tempos de respostas mais rápidos e uma interface cada vez mais amigável. Então, cada programa na memória utilizaria o processador em pequenos intervalos de tempo. Esse sistema de divisão de tempo ficou conhecido como compartilhamento de tempo (*time-sharing*). A partir desta geração as vendas dos microcomputadores cresceu a larga escala, dando origem a uma nova indústria. O preço dos microcomputadores giravam em torno de 120 mil dólares e venderam muito. Foi também nesta época que surgiu o Sistema Operacional UNIX escrito por um cientista da *Bell Labs*, Ken Thompson e este se tornou muito popular entre o mundo acadêmico, órgãos do governo e empresas (SILBERCHATZ, 2000).

A Quarta Geração (a partir de 1980) é marcada pelo desenvolvimento dos circuitos LSI (*Large Scale Integration*), chips de silício contendo milhares de transistores por cm^2. Esta geração foi o alvorecer da era do computador pessoal. Os equipamentos desta geração foram se tornando cada vez menores, mais velozes e mais potentes em capacidade de processamento e principalmente, mais acessível em relação ao preço. Esses novos equipamentos, com alto poder de computação, especialmente a computação interativa, com excelentes gráficos, levaram ao crescimento da indústria de *software*s para computadores pessoais. Nesta geração dois Sistemas Operacionais inicialmente dominaram o cenário dos computadores pessoais: o MS-DOS (Microsoft) e o UNIX. O MS-DOS foi amplamente utilizado no IBM-PC e em computadores com tecnologia Intel, que após muitos anos evoluiu para o Windows. O UNIX é outro importante concorrente que dominou as estações de trabalho e principalmente os servidores de rede (SILBERCHATZ, 2000).

Outro crescimento importante descrito por Silberchatz (2000) foi na área de redes de computadores pessoais utilizando Sistemas Operacionais de redes e Sistemas Operacionais distribuídos. Nos Sistemas Operacionais de rede, os usuários podiam se conectar a máquinas remotas e copiar arquivos de uma máquina para a outra, já, em um Sistema Operacional

distribuído os programas dos usuários podiam ser executados em vários computadores, porém, estes visualizavam o sistema como único.

2.2 – O UNIX

O surgimento do UNIX, descrito por Tanembaum (2003), nasceu no Bell Lab em 1970, desenvolvido por Ken Thompson e Dennis Ritchie e sua equipe, com a finalidade de ajudar nos controles dos projetos internos do próprio laboratório. Nasceu como um sistema básico, voltado principalmente para programadores e cientistas. No ano de 1975, quando Ken Thompson estava trabalhando como professor assistente na Universidade da Califórnia, em Berkeley, ele continuou a desenvolver o sistema UNIX desenhado no Bell Lab. Outros professores e alunos aderiram ao desenvolvimento, implementando uma série de melhorias no sistema originalmente desenhado. Tais melhorias deu origem a um sistema operacional com algumas diferenças em relação ao sistema UNIX desenvolvido no Bell Lab, passando a ser conhecido como UNIX de Berkeley. Algumas empresas começaram a comercializar esta versão do sistema operacional, uma delas foi a SUN Microsystems com o SUN-OS.

Em 1979, a AT&T lançou comercialmente o UNIX, conhecido como "Versão 7". Após muitos problemas com esta versão, em 1982, foi lançada a versão chamada de "*System* III". A partir desta data, o UNIX passou a ter duas versões paralelas, uma comercializada pela AT&T e outra proveniente da Universidade da Califórnia em Berkeley. No ano de 1983, o UNIX era muito utilizado no meio acadêmico principalmente para as aplicações científicas. A AT&T, visando sua expansão no uso comercial, agregou uma série de características e facilidades no UNIX. No princípio houve muita resistência ao seu uso pelos usuários comerciais, pois o UNIX ainda possuía uma interface muito científica, não sendo nada amigável ao uso comercial. A versão comercial mais conhecida foi o "System V" (TANEMBAUM, 2003).

Em 1989 as maiores empresas na área de computação formaram dois grandes consórcios, com o objetivo de unificação e padronização de todos os sistemas UNIX existentes no mercado. Isto foi necessário para que houvesse uma portabilidade de todas as aplicações desenvolvidas para UNIX, aumentando seu uso no mercado comercial. Atualmente, existem diferenças de implementação em alguns comandos em algumas versões UNIX comercializadas, mesmo que estes possuem ambas as versões (TANEMBAUM, 2003).

2.3 – O GNU/Linux

O Linux foi o *kernel* desenvolvido por Linus Torvalds em 1991, com o propósito de ser um *kernel* poderoso, livre e que todos pudessem utiliza-lo e modifica-lo conforme suas necessidades. Torvalds utilizou como base o sistema operacional MINIX, "uma espécie de UNIX reduzido para estudos acadêmicos" (JAMIL; GOUVÊA (2006) *apud* BRITO (2008)) como forma alternativa de uso, pois o UNIX era um *software* proprietário e nem todos os *hardware*s disponíveis na época eram compatíveis com ele. Torvalds então desenvolveu o *kernel* Linux para atender suas necessidades diárias, pois não estava interessado em obter lucro. Quando Torvalds divulgou o Linux, muitos usuários aderiram a este e começaram a fazer melhorias (BRITO, 2008).

Antes do anúncio do *Kernel* Linux por Torvalds, o projeto GNU estava sendo desenvolvido por Richard Stallman e sua equipe, o qual se baseava em um sistema operacional totalmente livre, onde todos os usuários pudessem contribuir com seu aperfeiçoamento. Até o final da década de 1980, o projeto de Stallman ainda não estava concluído, pois eles não tinham um "*Kernel*" para o seu sistema. No ano de 1990, o *kernell* Linux evoluiu muito rápido, despertando o interesse de Stallman. Então este propôs um acordo com Torvalds, para a união de seus sistemas, e a partir deste acordo nasceu o GNU/Linux. Em meados de outubro de 1991 foi lançada a primeira versão do GNU/Linux e distribuída sobre a licença GPL, a qual garante a liberdade de compartilhar, modificar ou utilizar trechos de todos os programas e demais trabalhos que a ela se aplica e, o desenvolvedor pode cobrar pelo *software* e/ou trabalho, mas não pode restringir o acesso de outras pessoas ao programa ou código-fonte (BRITO, 2008).

2.4 – O FreeBSD

O FreeBSD surgiu a partir de um *patchkit* (pacote de correção de erros) do sistema operacional 386BSD, que na época, era considerado um bom sistema, mas que possuía alguns problemas com atualizações para correções erros. Então Nate Williams, Rod Grimes e Jordan Hubbard, da equipe de desenvolvedores do 386BSD, desenvolveram este *patchkit*, e apresentaram ao mantenedor do projeto Bill Jolitz. Com este *patchkit* era possível atualizar o sistema de forma bastante prática, mas Bill Jolitz não viu com bons olhos as intenções dos desenvolvedores, e retirou todo o apoio ao projeto bem como todo o planejamento futuro para o desenvolvimento do sistema (HANDBOOK, 2012).

Estes acontecimentos dataram de 1992 a 1993, sendo que em 1993 é que realmente podemos considerar que o FreeBSD foi concebido. O nome FreeBSD, foi sugerido por David Greenman, que deu continuidade ao trabalho de base já realizado pela equipe do 386BSD. Uma de suas metas com relação ao projeto era divulgar o sistema, que na época era praticamente desconhecido. Então foi negociado com a Walnut Creek CDROM, para a distribuição do CD com o FreeBSD, pois nesta época, eram poucas os que possuíam conexão com a Internet. Este feito abriu as portas para o FreeBSD, tornando-o muito usado no meio acadêmico, pois este era rápido, leve e se mostrou muito eficiente com a Internet (HANDBOOK, 2012).

O lançamento oficial do FreeBSD ocorreu em dezembro de 1993, em CD e pela Internet o qual chamou-se FreeBSD 1.0, e teve como base muitos componentes do 386BSD e da *Free Software Foundation*. O sucesso foi tanto, que em maio de 1994 surgiu o FreeBSD 1.1. Nesta época ainda corria um processo judicial envolvendo a Novell e a U.C Berlekey, sobre direitos autorais de partes do código fonte do Net/2 (4.3BSD-Lite), que na época eram a grande base do FreeBSD, onde a AT&T era proprietária e estes fora comprada pela Novell. A solução encontrada pela equipe de desenvolvimento do projeto foi retirar todos os trechos de propriedade da Novell do sistema e refazer todos os códigos do sistema que ficaram órfãos, mesmo assim, ainda foi lançada a versão FreeBSD 1.1.5.1 (HANDBOOK, 2012).

Mesmo com todos os problemas, o FreeBSD ganhou força e conseguiu reestruturar os códigos retirados, e em dezembro de 1994, foi lançado a versão 2.0 do FreeBSD sendo sua distribuição por CD. A Internet ajudou a difundir a nova versão do sistema, que, mesmo com alguns problemas, agradou ao público. Em junho de 1995 foi lançada a versão 2.0.5 e a partir de agosto de 1996 o FreeBSD 2.1.5 passou a ser utilizado em provedores de Internet e em empresas. Após a versão 2.1.5, o seu desenvolvimento tornou-se mais organizado, surgindo o ramo *STABLE*. Em novembro de 1996 iniciaram-se os trabalhos para o desenvolvimento da série 2.1-*STABLE*, e foi deste ponto em diante que o FreeBSD adquiriu as denominações para as fases de desenvolvimento *RELEASE* e *CURRENT* (HANDBOOK, 2012).

Em fevereiro de 1997 esta versão tornou-se *STABLE*, abrindo caminhos para o desenvolvimento da versão 2.2, que em abril de 1997 foi lançada, e sua última, a 2.2.8 foi em novembro de 1998. A partir do ano de 1999, surgiram duas fases de desenvolvimento, a série 3.X *STABLE*, e a 4.0-*CURRENT*. No mês de junho de 2000, após melhorias, a série 3.X se tornou *STABLE* e a série 4.X estava a caminho da versão *STABLE* e devido a sua confiabilidade, foi considerada por muitos a melhor versão já lançada para uso em ambiente de produção, pois contava com muitos adicionais como, suporte a *hardware* e melhorias em

sua performance em relação as versões anteriores. Tal confiabilidade agregou novos adeptos, muitos ex-usuários do GNU/Linux (HANDBOOK, 2012).

Em janeiro de 2003 foi lançada a versão 5.0-*RELEASE*. Nesta série, houve uma evolução muito grande no FreeBSD como suporte avançado a multiprocessamento simétrico, suporte avançado a aplicações *multithread* e a novas arquiteturas como o *UltraSparch*® e *ia64*. Por tal razão, as versões 5.X são consideradas versões de "Nova Tecnologia", enquanto a série 4.X atua como versões de "Produção". No final de 2005 saiu a versão 6.X e esta por sua vez foi até fevereiro de 2008 quando foi substituída pela versão 7.X. No ano seguinte, mais precisamente em novembro de 2009 saiu a versão 8.X, ficando até janeiro de 2012, quando foi lançado a versão 9.X (HANDBOOK, 2012).

2.5 – Evolução dos Sistemas Operacionais baseados em UNIX

A Figura 1 mostra a evolução dos Sistemas Operacionais baseados em UNIX desde a sua criação em 1969 por Ken Thompson, Dennis Ritchie e sua equipe (LEVENEZ, 2013).

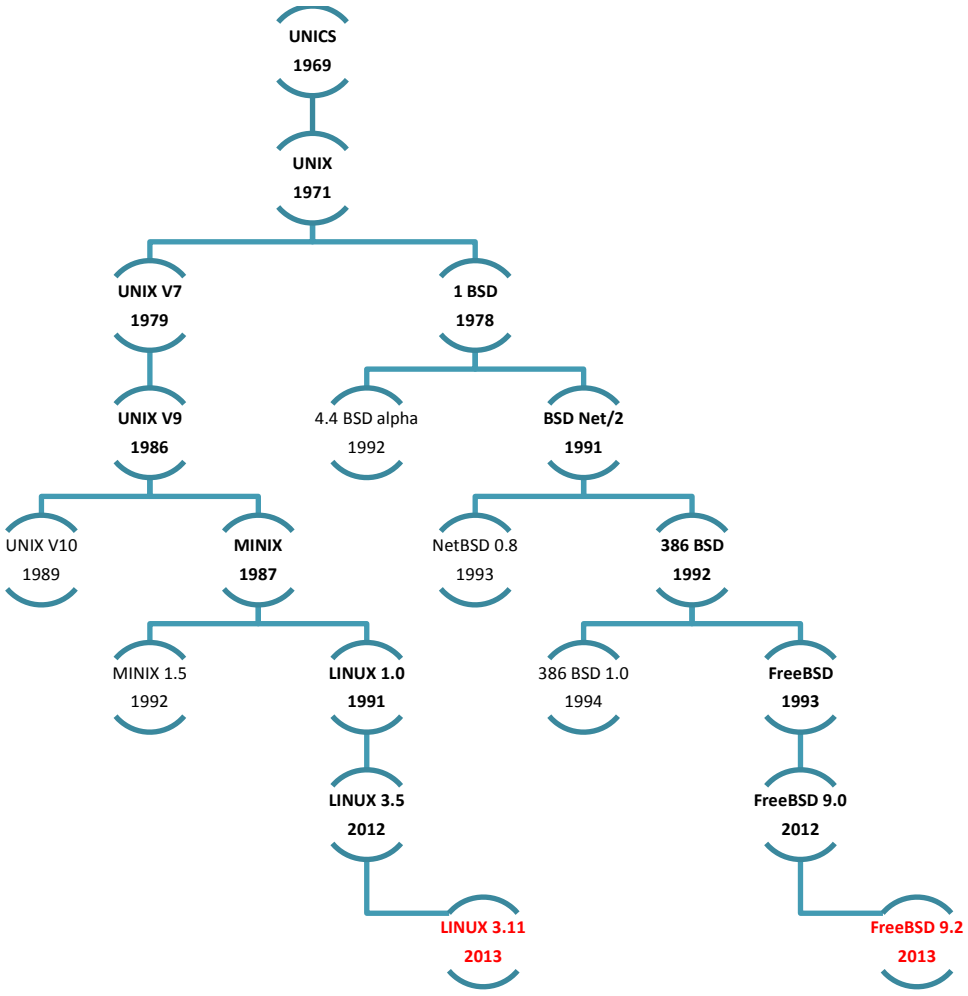

Figura 1: Evolução dos Sistemas Operacionais baseados em UNIX.
Fonte: Adaptação do original (LEVENEZ, 2013).

2.6 – Segurança de Computadores

De acordo com a ISO/IEC 17.799 (2000), as informações de uma organização são ativos importantes à continuidade de seus negócios, possuem valor e consequentemente precisam ser protegidos adequadamente. A segurança de informações protege as informações contra ameaças, assegurando a continuidade dos negócios, visando minimizar os prejuízos e maximizar o retorno dos investimentos e das oportunidades comerciais.

Para Tanembaum (2003, p. 498),

> "... a segurança de computadores é um tópico da atualidade e com um grande volume de pesquisa, embora a maioria não esteja diretamente ligada aos sistemas operacionais, mas sim com a segurança em redes como correio eletrônico, Web e comércio eletrônico, criptografia, Java ou gerenciamento de uma instalação de computadores seguros".

As ameaças dos Sistemas Operacionais podem vir de muitas maneiras, desde ataques internos como externos. Muitos ataques começam com uma tentativa de invasão por um cracker usando um dicionário de senhas comum e sendo bem sucedido. Para se evitar este tipo de ataque, é utilizado o uso de senhas fortes com trocas periódicas, senhas com desafios, cartões inteligentes e sensores biométricos. Dentre outros tipos conhecidos de ataques aos Sistemas Operacionais estão os cavalos de troia, conexão impostora, bomba lógica, alçapão e os ataques por transbordo de buffer.

Devido às facilidades de acesso a Internet, o aumento do conhecimento e da informação tornou-se um fator importante para que as organizações investissem na segurança da informação tornando-a um pré-requisito para qualquer sistema de informações. Para KUROSE (2006), há uma relação de dependência entre a segurança da informação e seus pilares:

Confidencialidade. Este pilar determina que somente o remetente e o destinatário pretendido possam ler e entender o conteúdo da mensagem transmitida e, caso esta mensagem venha a ser interceptada, a mesma devera estar codificada de maneira que o intruso não consiga decifrar o seu conteúdo. Atualmente a confidencialidade expressa o significado de comunicação segura.

Autenticação. O pilar de autenticação determina que o destinatário possa confirmar a identidade do remetente, confirmando que ele é quem alega ser.

Integridade e não repudio de mensagem. Este pilar determina que mesmo que o remetente e o destinatário consigam se autenticar reciprocamente, eles também precisam assegurar que o conteúdo da comunicação não tenha sido alterado, por acidente ou por má

intenção, durante a transmissão, como também que ambas as partes não neguem ter enviado ou recebido à mensagem.

Disponibilidade e controle de acesso. Neste pilar, origina-se um requisito fundamental para comunicação segura, estabelecendo que, somente entidades cadastradas podem ter acesso aos recursos da rede como também determina que os recursos de rede devam estar imunes a ataques para que seus recursos não sejam interrompidos.

Confidencialidade, autenticação, integridade e não repudiação de mensagens são considerados componentes fundamentais da comunicação segura há muito tempo. Influências da noção de comunicação segura são disponibilidade e controle de acesso, estas motivadas por preocupações muito reais com a segurança da infraestrutura de rede contra um potencial ataque violento. Um dos modos mais seguros para garantir que intrusos não possam causar danos é, antes de qualquer coisa, assegurar que seus pacotes não entrem na rede (KUROSE, 2006).

2.6.1 – *Firewalls*

Atualmente a Internet não é um lugar muito seguro, nela, os *crackers* [1]estão por toda parte, criando todo tipo de destruição. Do ponto de vista dos administradores, o mundo está dividido claramente em dois campos – "os bonzinhos e os bandidos".

Firewall é uma combinação de *hardware* e *software* que isola a rede interna de uma organização da internet em geral, permitindo que alguns pacotes passem e bloqueando outros. Um *firewall* permite que um administrador de rede controle o acesso entre o mundo externo e os recursos da rede que administra gerenciando o fluxo de tráfego de e para esses recursos. Embora organizações de grande porte possam utilizar vários níveis de *firewalls* distribuídos, localizar um *firewall* em um único ponto de acesso a rede, facilita a administração e a imposição de uma politica de segurança de acesso (KUROSE, 2006).

Kurose (2006) define que há dois tipos de *firewall*: *firewalls* de filtragem de pacotes que funcionam na camada de rede e *gateways* de camadas de aplicação que funcionam na camada de aplicação.

[1] Crackers - Pessoas aficionadas por informática que utilizam seu grande conhecimento na área para quebrar códigos de segurança, senhas de acesso a redes e códigos de programas com fins criminosos. Em alguns casos, o termo "Pirata Virtual" é usado como sinônimo para cracker.

2.6.1.1 – Filtros de pacotes

Para Kurose (2006) os filtros de pacotes primeiramente analisam os cabeçalhos de datagramas e então aplicam regras de filtragem especificadas por um conjunto definido pelo administrador, as quais determinam se o datagrama será descartado ou passará.

As decisões de filtragem são normalmente baseadas em: endereço IP de origem e destino; porta TCP ou UDP de origem e destino; tipo de mensagem ICMP; datagramas de inicialização de conexão usando bits TCP SYN ou ACK. Uma política de filtragem também pode ser baseada na combinação de endereços e números de porta (KUROSE, 2006).

Na Figura 2 é apresentado o cabeçalho IP cujos parâmetros que são analisados pelo *firewall* a nível de pacotes são: Protocolo, *Flags*, endereço de origem e endereço de destino.

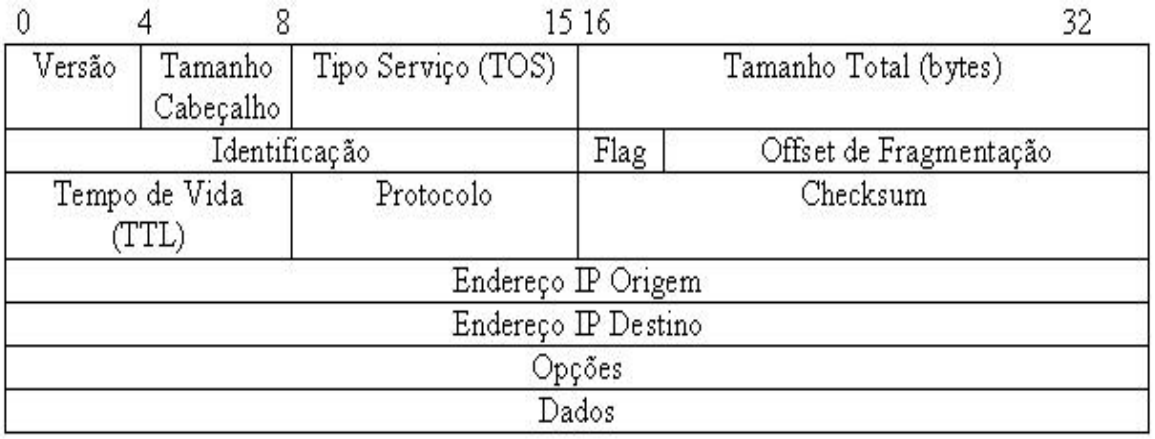

Figura 2: Cabeçalho IP
Fonte: (Souza, 2012)

Na Figura 3 é apresentado o cabeçalho TCP cujos parâmetros que são analisados pelo *firewall* a nível de pacotes são: Porta de origem, porta de destino, e as *flags*.

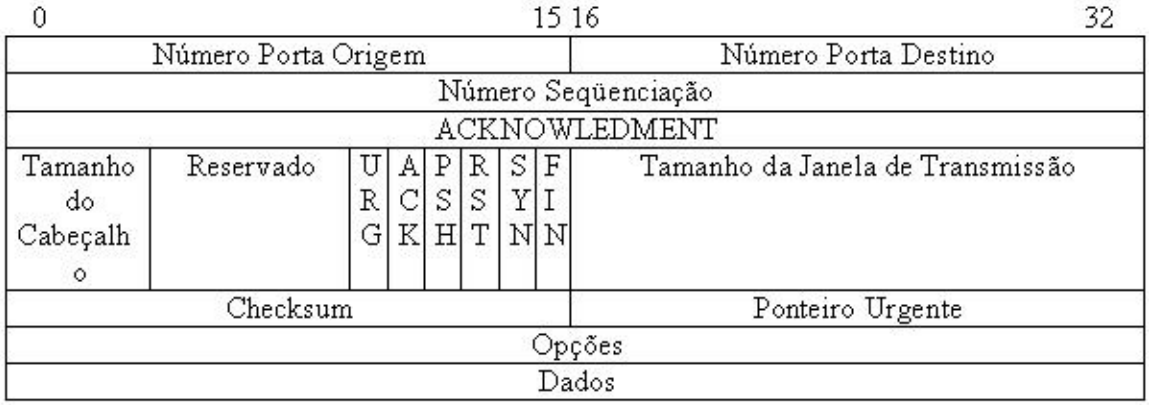

Figura 3: Cabeçalho TCP
Fonte: (Souza, 2012)

Infelizmente, basear a politica em endereços externos não oferece nenhuma proteção contra datagramas cujo endereço de fonte pertence a um hospedeiro que está na lista de permitidos, mas que, na verdade, foi enviado por outro hospedeiro (que muito provavelmente pertence a um bandido que falsificou o endereço de fonte dos datagramas) (KUROSE, 2006).

A filtragem pode também ser baseada no bit TCP ACK ter ou não valor. Esse artifício é bastante útil quando uma organização quer permitir que seus clientes internos se conectem com servidores externos, mas quer impedir que clientes externos se conectem com servidores internos. O primeiro segmento de todas as conexões TCP tem o bit ACK com valor 0, ao passo que todos os outros segmentos tem o bit ACK com valor 1. Assim, se uma organização quiser impedir que clientes externos iniciem uma conexão com servidores internos, ela simplesmente filtrara todos os segmentos que entram que tenham o bit ACK ajustado em 0. Essa politica elimina todas as conexões TCP originadas do exterior, mas permite conexões que se originam internamente (KUROSE, 2006).

2.6.1.2 – *Gateways* de aplicação

Para assegurar um nível mais refinado de segurança, os *firewalls* têm de combinar filtro de pacotes com *gateways* de aplicação. *Gateways* de aplicação fazem mais do que examinar cabeçalhos IP/TCP/UDP e tomam decisões com base em dados da aplicação. Um *gateway* de aplicação é um servidor específico de aplicação através do qual todos os dados da aplicação (que entram e que saem) devem passar (KUROSE, 2006).

Vários *gateways* de aplicação podem executar no mesmo hospedeiro, mas cada *gateway* é um servidor separado, com seus próprios processos. *Gateways* de aplicação não estão isentos de desvantagens. Em primeiro lugar, é preciso um *gateway* de aplicação diferente para cada aplicação diferente. Em segundo lugar, há um preço a pagar em termos de desempenho, visto que todos os dados serão repassados por meio do *gateway*. Isso se torna uma preocupação particularmente quando vários usuários ou aplicações estão utilizando o mesmo *gateway*. Por fim, é preciso fazer certo esforço extra de configuração. Há duas alternativas segundo Kurose (2006):

- Quando o usuário faz uma solicitação, o *software* cliente tem de saber como contratar o *gateway* em vez do servidor externo, e também como comunicar ao *gateway* de aplicação a qual servidor este deve se conectar, ou
- O usuário deve se conectar explicitamente ao servidor externo por meio do *gateway* de aplicação.

Para Kurose (2006) os *firewalls* não são, de maneira nenhuma, uma solução para os problemas de segurança. Eles introduzem um compromisso entre o grau de comunicação com o mundo exterior e o nível de segurança. Como os filtros não podem impedir a falsificação de endereços IP e de números de porta, eles frequentemente usam uma politica de tudo ou nada. Gateways também podem ter bugs no *software* que permitem intrusos penetrarem neles. Por fim, *firewalls* são ainda menos eficazes se as comunicações internas puderem alcançar o mundo externo sem passar por eles. As comunicações sem fio e os modens discados são dois exemplos disso.

2.6.2 – O *ipfirewall* do FreeBSD

O *ipfirewall* é o utilitário utilizado para implementar a filtragem de pacotes IP e controle de tráfego de rede, é uma ferramenta de *firewall* nativa com a qual o FreeBSD trabalha por padrão e, desde a versão 4.x o *ipfirewall* passou a trabalhar com o gerenciamento de conexões por estado - *statefull* e *statless*. Mantido pela equipe de desenvolvimento do FreeBSD, atualmente suporta as duas versões de protocolo IP: IPv4 e IPv6. Inicialmente o *ipfirewall* vem desabilitado a nível de *kernel*, sendo que este pode ser habilitado por meio da recompilação do *kernel*, ou inicializando este serviço (HANDBOOK, 2012).

A forma como o IPFW trabalha é parecida com a adotada em muitos outros filtros de pacotes (*Packet Filters*), com exceção do *ipfilter* (ipf), que opera com um padrão para tratar as regras que é menos eficiente e requer bem mais cuidado na hora de ajustar o *firewall*. Mas isso não tira a qualidade e o poder de implementação de *firewalls* do ipf, que tem também suas próprias vantagens (HANDBOOK, 2012).

Um *firewall* por filtragem de pacotes atua monitorando pacote-a-pacote de todas as conexões. A partir da versão 4.x do FreeBSD, o *ipfirewall* passou a gerenciar uma filtragem por estado (*stateful*) de conexões. Esse comportamento é sempre transparente, pois ninguém nota que existe um *firewall* presente, até que um evento aguardado seja bloqueado.

Um *firewall* pode ser implementado de diversas formas, mas todas elas podem ser simplificadas com base em duas políticas de filtragem: aberto e fechado. O *firewall* que segue uma política aberta permite que todos os pacotes sejam roteados por padrão, e bloqueia aqueles que pertencem a um tipo de conexão que não é desejada, ou seja, "abre tudo e bloqueia os indesejados". Já o *firewall* que segue uma política fechada, faz o inverso, bloqueando todo o roteamento de pacotes, e libera um a um o tráfego de conexões permitidas. Uma política fechada proporciona um *firewall* muito mais rígido, contudo sua configuração é bem mais trabalhosa, porque se pode facilmente bloquear o tráfego de algum serviço que

esteja sendo utilizado na rede. Alguns protocolos estabelecem conexões dinâmicas, que são bem mais difíceis de prever, mas tem-se que estar atento a esse tipo de situação (HANDBOOK, 2012).

O *ipfirewall* pode ser habilitado de duas formas: estática ou dinâmica. Na forma estática, adicionam-se as opções apropriadas ao *kernel* editando seu arquivo de configurações, e compilando o novo *kernel* para o sistema. Na forma dinâmica utilizamos a ferramenta *kldload ipfw* que carregará dinamicamente os módulos base no *kernel*. Os dois modos, estático e dinâmico funcionam bem, adicionando um *firewall* com configurações básicas, embora, a compilação estática proporciona um melhor desempenho, permitindo adicionar opções mais detalhadas de configuração.

Pra carregar o *ipfirewall* de forma estática, é necessário editar o *kernel*, adicionando as seguintes linhas no arquivo de configurações:

options IPFIREWALL
options IPFIREWALL_VERBOSE
options IPFIREWALL_FORWARD
options IPFIREWALL_VERBOSE_LIMIT=100
options IPFIREWALL_DEFAULT_TO_(ACCEPT ou DENY)
options IPV6FIREWALL
options IPV6FIREWALL_VERBOSE
options IPV6FIREWALL_FORWARD
options IPV6FIREWALL_VERBOSE_LIMIT=100
options IPV6FIREWALL_DEFAULT_TO_(ACCEPT ou DENY)

Onde:

A opção IPFIREWALL habilita o *firewall* ao iniciar o sistema.

A opção IPFIREWALL_VERBOSE permite logar o tráfego de pacotes com o *ipfirewall*, ecoando informações sobre atividade dos pacotes ao *syslogd*, em cada regra que tiver a opção "log" definida.

A opção IPFIREWALL_FORWARD permite que os pacotes sejam redirecionados para outros *hosts*, utilizando-se o comando "fwd" do *ipfirewall*. Redirecionando o pacote (*FORWARD*) para que ele siga de um ponto específico (*host*, rede, porta) para outro.

A opção IPFIREWALL_VERBOSE_LIMIT=100 permite definir um limite de pacotes que serão logados para uma regra específica. Dessa forma, o *syslogd* não será sobrecarregado com mensagens relativas às atividades do *ipfirewall*, os comentários do *kernel* para essa opção diz que isso é uma forma de evitar negação de serviço com os logs gerados para algumas regras de *firewall*, caso um possível ataque gere muitas ocorrências da mesma

regra. Por exemplo, a opção IPFIREWALL_VERBOSE_LIMIT=100, define, quando existirem 100 ocorrências da mesma regra, que as informações sobre a atividade de pacotes daquela regra deixará de ser logado.

A opção IPFIREWALL_DEFAULT_TO_ACCEPT no *kernel* define uma política aberta e gera uma regra de número 65000 que será automaticamente carregada como "*allow* ip", permitindo o tráfego de qualquer um para qualquer um. Já a opção IPFIREWALL_DEFAULT_TO_DENY define uma política fechada gerando uma regra número 65535, automaticamente carregada como "*deny* ip", bloqueando qualquer acesso.

Após a reconfiguração, deve-se efetuar a compilação e instalação do *kernel*. Para que as alterações possam ter efeito, é necessário reinicializar o sistema (HANDBOOK, 2012).

Para carregar o ipfw dinamicamente via linha de comando, usa-se o comando *kldload ipfw*, e para habilita-lo toda vez que o sistema inicializar, é preciso editar o arquivo **/etc/rc.conf**, como mostrado abaixo:

***firewall*_enable="YES",**

E para *firewalls* abertos:

***firewall*_type="OPEN"**

Pra habilitar as mesmas opções do *ipfirewall* estático sem recompilar o seu *kernel*, deve-se definir as variáveis correspondentes editando o arquivo **/etc/sysctl.conf**, como mostrado:

```
root@root~# sysctl -w net.inet.ip.fw.verbose=1
root@root~# sysctl -w net.inet.ip.fw.verbose_limit=100
root@root~# sysctl -w net.inet.ip.forwarding=1
```

A importância de se definir um *firewall* da forma correta, usando o arquivo de configuração **/etc/rc.*firewall*** via **rc.conf** é altamente recomendável, principalmente quando se utiliza um *shell script* que é executado sempre que o sistema iniciar e que carregue todas as regras definidas no arquivo de configuração. Para isso, deve-se ficar atento na hora de criar as regras (HANDBOOK, 2012).

2.6.3 – O *iptables* do GNU/Linux

Um dos vários *firewalls* desenvolvido para o GNU/Linux é o *iptables*, o qual é o responsável pela interface de configuração do *netfilter*, que foi introduzido no *kernel* do Linux a partir da versão 2.4. Com ele é possível editar a tabela de filtragem de pacotes nativa do

kernel, a qual consiste na comparação entre os dados do cabeçalho (*header*) do pacote analisado com as regras definidas (MORIMOTO, 2008).

Os pacotes analisados seguem os parâmetros predefinidos nas regras da tabela do *netfilter*, e executam três tipos de ação: *ACCEPT* (aceitar), onde este aceitará todos os pacotes que coincidirem com a regra, *DROP* (descartar), ele irá descartar todos os pacotes que coincidirem com esta regra ou *REJECT* (rejeitar) onde irá rejeitar todos os pacotes que coincidirem com a regra enviando uma mensagem informando o motivo da não aceitação (MORIMOTO, 2008).

O *iptables* possui uma estrutura lógica formada basicamente por tabelas, *chains* (cadeias) e regras, as quais consistem em uma forma de organizar as informações para serem processadas pelo *netfilter*. As tabelas representam a descrição da área de atuação das regras, sendo que, as existentes por padrão são: filter, nat e mangle. Entretanto, outras tabelas podem ser criadas. As chains estão relacionadas ao tipo de roteamento do pacote dentro da máquina Firewall. No iptables existem chains específicas para cada tabela. E as regras são as diretrizes que o *netfilter* toma como base para determinar a ação a ser tomada de acordo com as informações do cabeçalho do pacote. Estas regras se apresentam na forma: **Tabela – Opção – *Chain* – Parâmetros – Ação1** (MORIMOTO, 2008).

As *chains*, *INPUT* (Entrada) trata da entrada de dados na máquina onde está o *firewall*, *FORWARD* (Repasse), compreende o repasse dos pacotes que atravessam o *firewall* e *OUTPUT* (Saída), a qual abrange os pacotes que saem da máquina onde está o *firewall*. As *chains* possuem um papel importante dentro do *iptables*, pois são elas que diferenciam o destino dos pacotes desde sua chegada a máquina *firewall* (MORIMOTO, 2008).

Para habilitar, deve-se criar um script com todas as regras necessárias para o gerenciamento da rede, dentro de um arquivo de texto, salvando o como, por exemplo: **/etc/rc.d/init.d/meu_*firewall.sh***.

Em seguida, deve-se dar permissão de execução para o arquivo com o comando chmod como mostrado abaixo:

 # chmod +x /etc/rc.d/init.d/meu_*firewall*.sh

Para tornar a inicialização totalmente automática, deve-se editar o arquivo de inicialização do sistema, que fica na maioria das distribuições GNU/Linux no diretório **/etc/rc.d/rc.local**, e adicionar a seguinte linha no final do arquivo com o caminho do script: **/etc/rc.d/init.d/meu_*firewall.sh***. Assim todas as vezes que o sistema iniciar ira carregar as regras de *firewalls* durante a inicialização.

3 – DESENVOLVIMENTO

Neste capítulo serão abordadas as ferramentas e *softwares* utilizados para a realização dos testes. Também será mostrado um passo-a-passo para a realização dos testes.

3.1 – O Ambiente de testes

O ambiente todo foi montado com máquinas virtuais devido ao baixo custo, facilidade de manuseio, configuração e manipulação. Como base do sistema virtual, utilizou-se o *VMware Player* versão 5.0.2, por ser um *software* gratuito, de fácil manuseio e com suporte a grande maioria de Sistemas Operacionais disponíveis no mercado. Outro fator importante do *VMware Player* é com relação ao baixo consumo de recursos de *hardware*.

Para as máquinas virtuais, foi utilizado *software* livre, o *Ubuntu BackTrack 5R3*, o *GNU/Linux Slax 2.6*, o *FreeBSD 9.0* e, para coleta de dados e monitoração o *Ubuntu 12.1* onde foi instalado o *software* de monitoração *Zabbix 2.0.8*. As configurações de cada uma delas estão descritas no ANEXO I.

As máquinas virtuais foram conectadas entre si conforme mostrado na Figura 4 e na Figura 5. Foram utilizado duas redes distintas, sendo uma na faixa de IP 192.168.0.0/24 para a realização dos testes e outra na faixa de IP 172.16.0.0/24 para coleta de dados dos servidores *Slax* e *FreeBSD*, onde foram testados as vulnerabilidades de seus *firewalls* contra ataques de Negação de Serviços (DoS – *Denial-of-Service*).

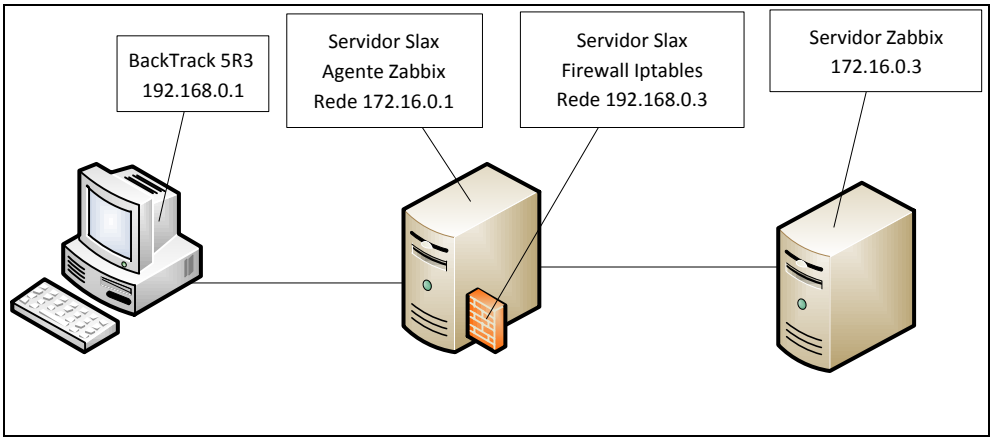

Figura 4: Rede Servidor Slax.
Fonte: Próprio autor.

Cada ambiente virtualizado foi configurado de maneira a se obter o melhor desempenho de acordo com as recomendações de cada desenvolvedor de cada *software*,

observando as limitações da máquina hospedeira. Os sistemas operacionais foram configurados com aplicações necessárias, de acordo com a necessidade para ocorrerem os testes, como pode ser observadas no ANEXO II.

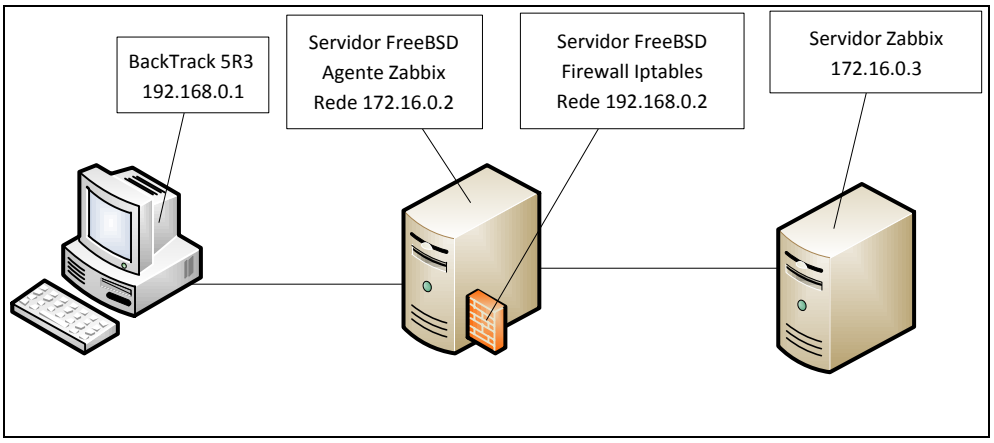

Figura 5: Rede Servidor FreeBSD.
Fonte: Próprio autor.

A máquina física hospedeira das máquinas virtuais é um *notebook* Dell *Inspiron* modelo 1545 com sistema operacional *Ubuntu 13.04* 64bits, cujas configurações se encontram no ANEXO II.

3.2 – Ferramentas utilizadas

Os testes basearam-se em ataques de Negação de Serviço (DoS–*Denial-of-Service*), o qual consistiu em fazer com que o servidor atacado tivesse dificuldades ou mesmo, foi impedido de executar suas tarefas. Para isso, em vez de invadir ou mesmo infectá-lo com *malwares*, o sistema operacional invasor faz com que a máquina receba uma quantidade excessiva de requisições até chegar ao ponto de não conseguir dar conta delas. Em outras palavras, o computador fica tão sobrecarregado que começa a negar o atendimento ao serviço. A função do *firewall* para este tipo de ataque é a de impedir que isto aconteça, detectando e bloqueando o ataque, evitando assim a sobrecarga do sistema. Segundo divulgado em blogs de administradores de sistemas, ataques DoS são os mais difíceis de serem barrados por *firewalls*, por utilizarem ferramentas que mascaram o IP do atacante e os protocolos utilizados.

Para o ataque foi utilizado a ferramenta de *software T50*, por ser uma das melhores neste tipo de ataque. Esta ferramenta foi projetada para testes de DoS e teve por

desenvolvedor o brasileiro, Nelson Brito. Outras ferramentas também poderiam ter sido usadas como o *inundator* e *hping3*, ambas possuem a mesma finalidade que a T50.

A ferramenta T50 realiza testes de *stress* em redes ou servidores utilizando a técnica de injeção de pacotes, mesclando 15 protocolos diferentes como TCP, UDP, ICMP, IGMPv2, IGMPv3, EGP, DCCP, RSVP, RIPv1, RIPv2, GRE, ESP, AH, EIGRP, OSPF. Ela capaz de mascarar o IP de ataque como as outras ferramentas citadas. Esta ferramenta consome muito processamento de CPU, por isso deve-se ter um bom equipamento com bastante capacidade de processamento.

Para o descobrimento de IPs em uma interface de rede, foi utilizado a ferramenta *netdiscover* presente no *BackTrack 5R3*. Com esta ferramenta é possível encontrar todos os IPs, bastando apenas efetuar um scanner na interface de rede como mostrado na Figura 6. Após a descoberta dos IPs da rede, foi efetuado um scanner no IP escolhido para determinar quais portas estavam disponíveis para a realização do ataque. Para isto utilizou-se a ferramenta *Zenmap*. Com esta ferramenta é possível descobrir quais portas estão abertas no IP escolhido, como também saber qual é a versão e o sistema operacional da máquina. Um exemplo disto pode ser observado nas Figuras 7 e 8.

```
Currently scanning: 10.23.63.0/8   |   Screen View: Unique Hosts

11 Captured ARP Req/Rep packets, from 9 hosts.   Total size: 660

  IP            At MAC Address      Count  Len   MAC Vendor
-----------------------------------------------------------------------

192.168.0.2     00:0c:29:51:6b:11    01    060   VMware, Inc.
192.168.0.3     00:0c:29:58:93:fa    01    060   VMware, Inc.
192.168.0.3     00:0c:29:58:93:04    01    060   VMware, Inc.
192.168.90.1    00:50:56:c0:00:01    01    060   VMWare, Inc.
192.168.90.254  00:50:56:e6:74:5e    01    060   VMWare, Inc.
172.16.0.1      00:0c:29:58:93:fa    01    060   VMware, Inc.
172.16.0.1      00:0c:29:58:93:04    01    060   VMware, Inc.
172.16.0.2      00:0c:29:51:6b:1b    03    180   VMware, Inc.
172.16.0.3      00:0c:29:48:b5:1c    01    060   VMware, Inc.
```

Figura 6: Descobertas de IPs com a ferramenta netdiscover.
Fonte: Próprio autor.

Para a coleta de dados dos servidores *Slax* e *FreeBSD*, foi montado um servidor com o *GNU/Linux Ubuntu 12.1*, e neste, instalado o *software Zabbix Server* para o monitoramento de cada um dos servidores. Nos servidores *Slax* e *FreeBSD*, foram instalados os agentes *Zabbix*, que coletam informações do estado da máquina como tráfego *input/output* da interface de rede, informações de uso da CPU e outros dados, enviando-os para o servidor

onde está instalado o *Zabbix Server*. As instalações e configurações do *Zabbix Server* e seus agentes estão, descritos no ANEXO III.

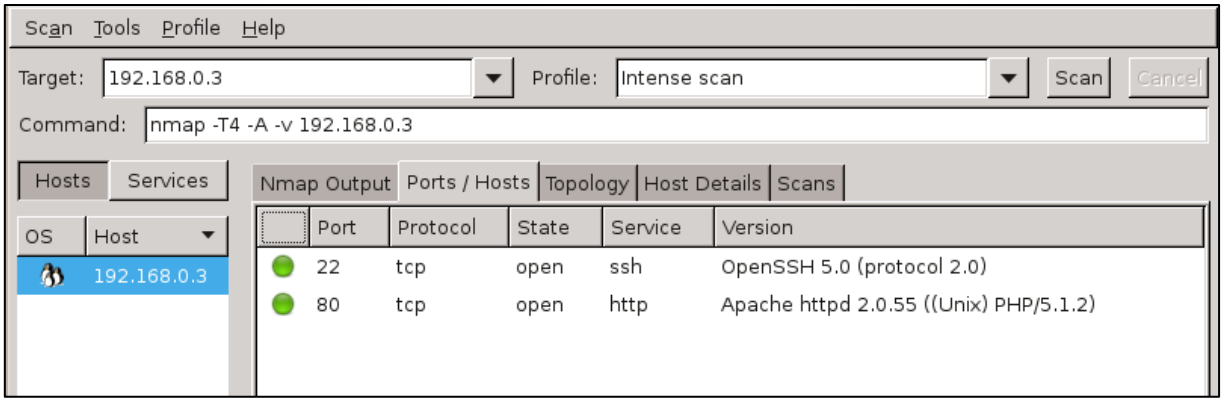

Figura 7: Escaneamento do IP 192.168.0.3 (GNU/Linux Slax) com Zenmap.
Fonte: Próprio autor

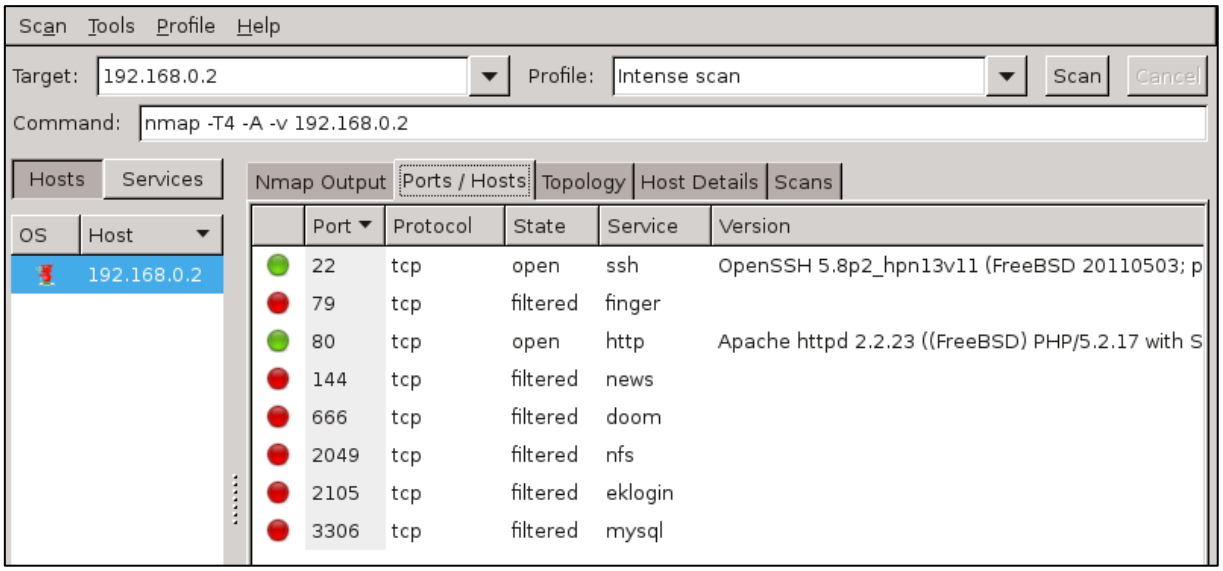

Figura 8: Escaneamento do IP 192.168.0.2 (UNIX FreeBSD) com Zenmap.
Fonte: Próprio autor

3.3 – Testes

Foi utilizado o sistema operacional *GNU/Linux Slax* para início dos testes, onde seu *firewall - iptables* foi configurado com as regras básicas de proteção, descritas no ANEXO IV.

Após a captura dos IPs pela ferramenta *netdiscover* e o escaneamento das portas disponíveis com scanner *Zenmap*, foi efetuado o ataque com a sintaxe mostrada na Figura 9. Esta sintaxe foi utilizada por apresentar um melhor desempenho de ataque, disparando contra o IP escolhido uma inundação de pacotes utilizando os 15 protocolos suportados pela

ferramenta. Isto exigiu o máximo de desempenho do *firewall*. Outras opções de utilização da ferramenta T50 podem ser visualizadas no manual da ferramenta pelo comando *man t50*.

```
root@bt:~# t50 192.168.0.3 --flood --turbo --saddr 192.168.0.1 --dport 22
entering in flood mode...
activating turbo...
hit CTRL+C to break.
T50 5.4.0 successfully launched on Oct 14th 2013 18:15:01
```

Figura 9: Sintaxe de ataque da ferramenta T50.
Fonte: Próprio autor

No servidor *Slax* foi simulado processos para que o uso de CPU e tráfego de rede ficassem próximos ao real de um servidor. Os processos ficaram em execução até o término dos testes. Durante os testes foram analisados o uso de processamento da CPU e o tráfego de rede *input/output* do servidor através dos dados enviados pelo agente de monitoramento para o *Servidor Zabbix*.

Para o segundo teste, utilizou-se o sistema operacional *UNIX FreeBSD 9.0*. Seu *firewall ipfw*, foi configurado igualmente o *firewall iptables* com as mesmas regras básicas de proteção, como descrito no ANEXO IV.

Os procedimentos foram os mesmos utilizados para o servidor *Slax*. Primeiramente efetuou-se a captura dos IPs pela ferramenta netdiscover e o escaneamento das portas disponíveis com scanner *Zenmap*. Depois realizou-se o ataque com a sintaxe mostrada na Figura 9. Esta sintaxe foi a mesma utilizada anteriormente, por apresentar um melhor desempenho de ataque, disparando contra o IP escolhido uma inundação de pacotes utilizando os 15 protocolos suportados pela ferramenta.

No servidor *FreeBSD* foi simulado processos para que o uso de CPU e tráfego de rede ficasse próximo ao real de um servidor. Os processos ficaram em execução até o término dos testes. Durante estes, foram analisados o uso de processamento da CPU e o tráfego de rede *input/output* do servidor através dos dados enviados pelo agente de monitoramento para o *Servidor Zabbix*.

4 – DISCUSSÃO DOS RESULTADOS

Neste capítulo serão abordados os resultados obtidos após realização dos testes de ataque DoS, testando as vulnerabilidades dos *firewalls*.

4.1 – Teste no Servidor Slax

A Figura 10 mostra o nível de processamento da CPU do servidor Slax antes do ataque de DoS, a qual estava com um nível de processamento estável. Na Figura 11 é possível observar que o tráfego de rede também estava estável com tráfego normal de *input/output*. O *firewall* foi configurado de modo a permitir o tráfego da rede interna, bem como barrar ataque DoS.

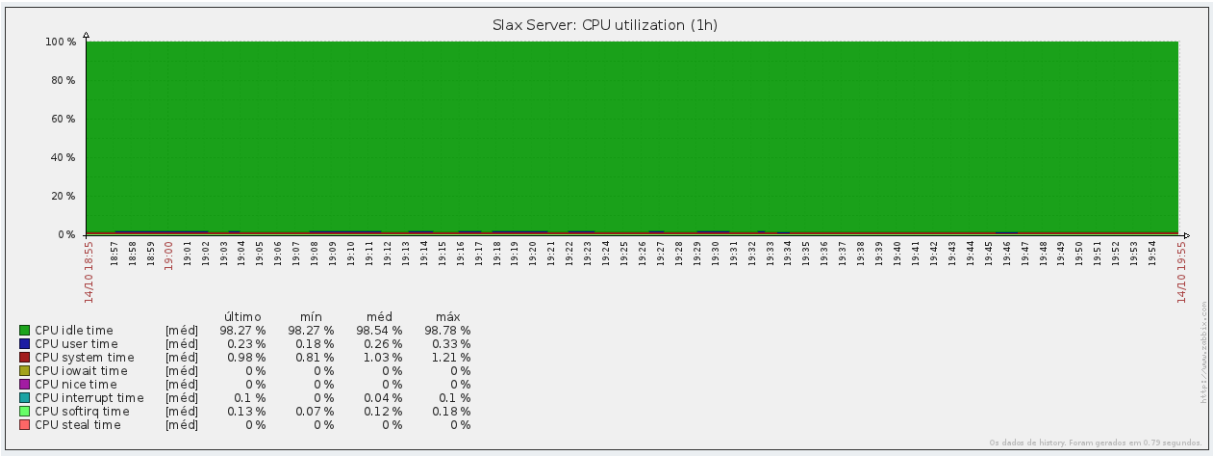

Figura 10: Gráfico utilização da CPU do servidor Slax antes do ataque DoS.
Fonte: Próprio autor

Após iniciado o ataque de DoS no servidor *Slax*, pode-se observar que o uso da CPU subiu para níveis elevados, conforme mostrado na Figura 11, e o tráfego de rede que estava subiu para níveis elevados, conforme mostrado na Figura 12, e o tráfego de rede que estava próximo de 4 Mbps, passou para mais de 32 Mbps como pode ser observado na Figura 13. Isto congestionou o tráfego, tornando o acesso ao servidor lento.

Como pode ser observado nos gráficos da Figura 12 e da Figura 13, no intervalo das 18h25 às 18h32 ocorreram falhas de coletas em decorrência da sobrecarga de requisições causadas pelo ataque de DoS, onde os processos do servidor pararam de responder. Mesmo com o *firewall* programado para filtrar ataques DoS, este não conseguiu impedi-lo, fazendo com que o servidor *Slax* passasse a negar serviços devido à sobrecarga.

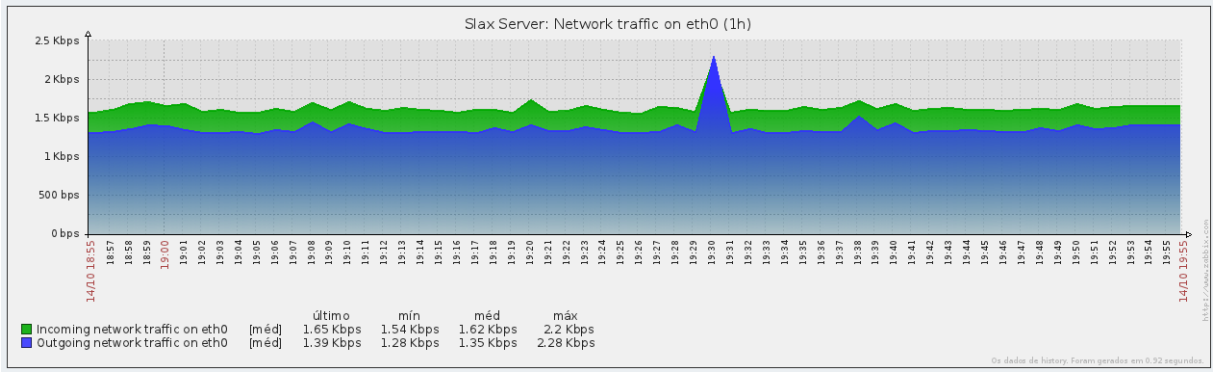

Figura 11: Gráfico de input/output de rede do servidor Slax antes do ataque DoS.
Fonte: Próprio autor

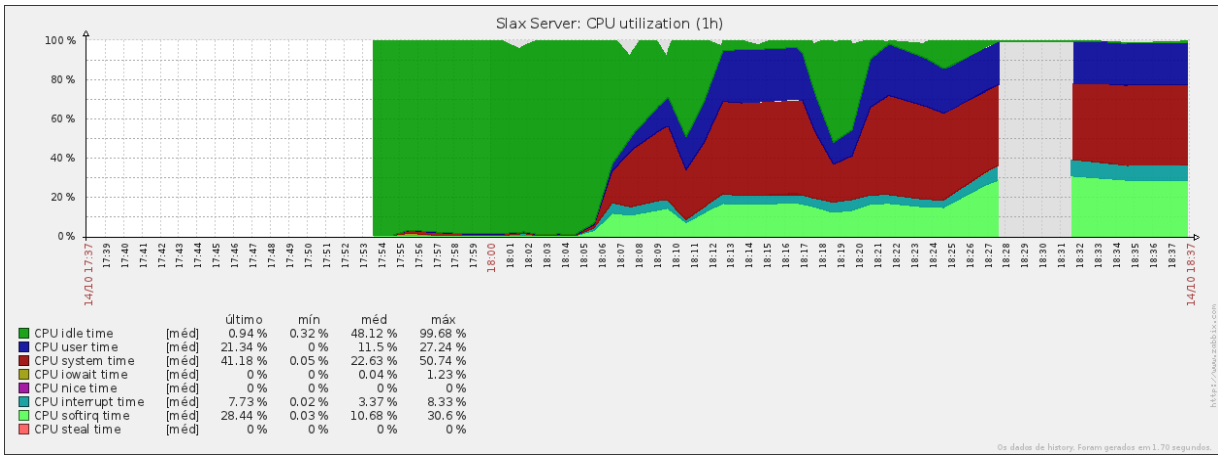

Figura 12: Gráfico de uso CPU durante ataque DoS no servidor Slax.
Fonte: Próprio autor

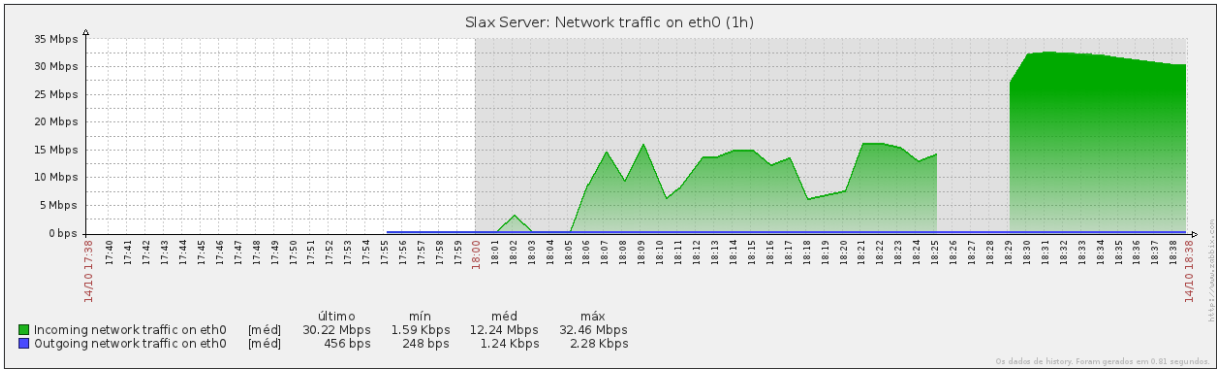

Figura 13: Gráfico de input/output de rede durante ataque DoS no servidor Slax.
Fonte: Próprio autor

4.2 – Teste no Servidor FreeBSD

A Figura 14 mostra o nível de processamento da CPU do servidor *FreeBSD* antes do ataque de DoS, a qual estava com um nível de processamento estável. Na Figura 15 é possível

observar que o tráfego de rede também estava estável com tráfego normal de *input/output*. O *firewall* foi configurado de modo a permitir o tráfego da rede interna, bem como barrar ataque DoS.

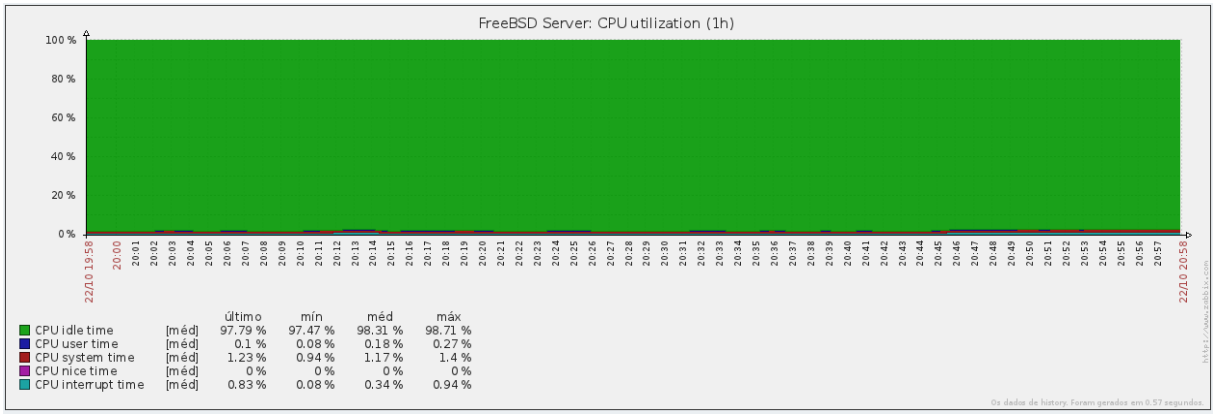

Figura 14: Gráfico de uso CPU do Servidor FreeBSD antes do ataque DoS.
Fonte: Próprio autor

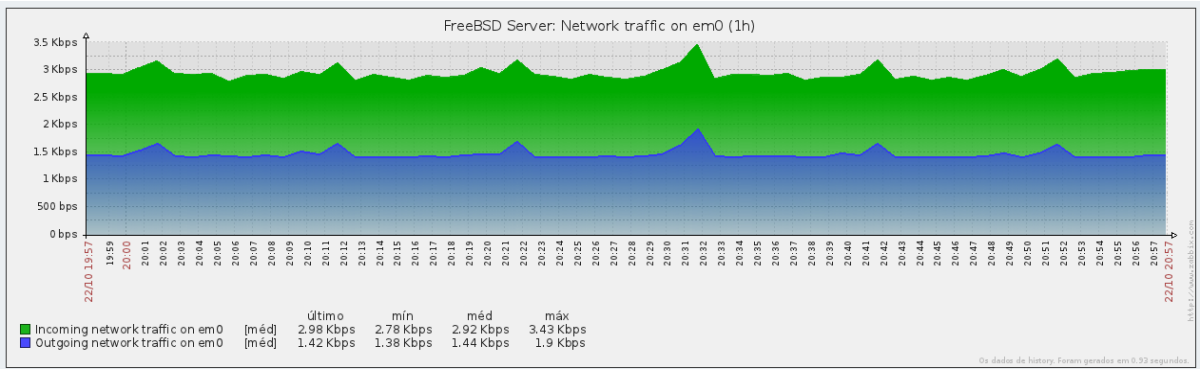

Figura 15: Gráfico de input/output de rede do Servidor FreeBSD antes do ataque DoS.
Fonte: Próprio autor

Após iniciado o ataque de DoS no servidor *FreeBSD*, pode-se observar que o uso da CPU subiu para a faixa máxima de processamento como pode ser visto na Figura 16, tornando o processamento de novos processos mais lentos pela sobrecarga do processador. Com relação ao tráfego de rede que estava em torno de 3,4Mbps, este passou para mais de 34Mbps como pode ser observado na Figura 17, congestionando o tráfego tornando lento o acesso ao servidor.

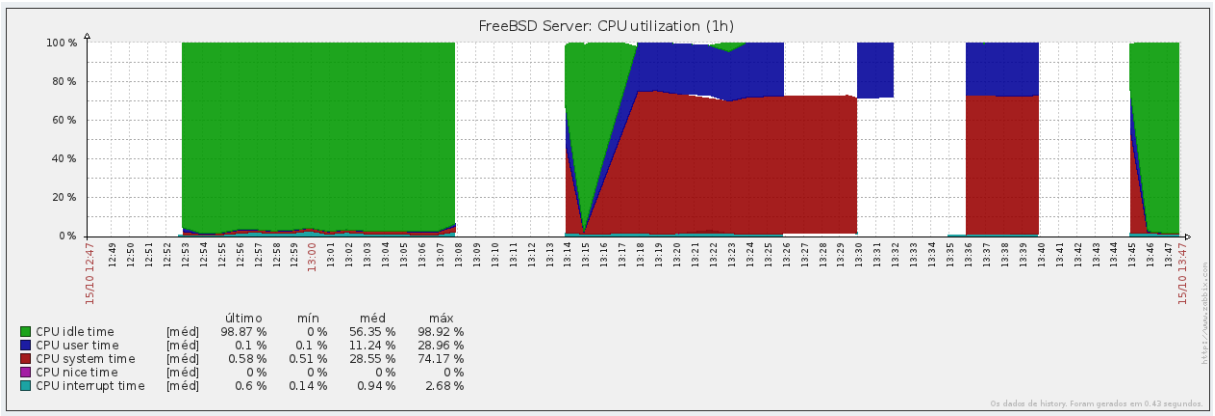

Figura 16: Gráfico de uso de CPU durante ataque DoS no Servidor FreeBSD.
Fonte: Próprio autor

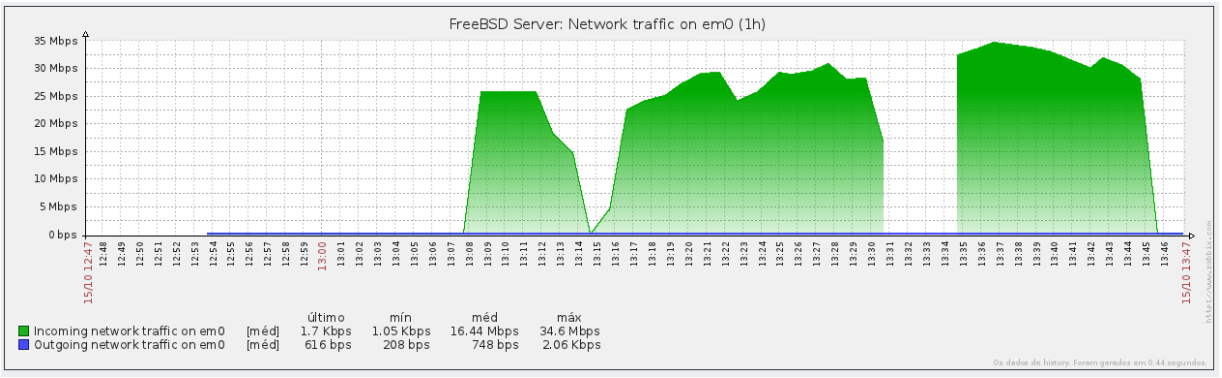

Figura 17: Gráfico de input/output durante ataque DoS no Servidor FreeBSD.
Fonte: Próprio autor

Como pode ser observado nos gráficos da Figura 16 e da Figura 17, no intervalo da 18h25 às 18h32 ocorreram falhas de coletas em decorrência da sobrecarga de requisições causadas pelo ataque de DoS, onde os processos do servidor congelaram. Mesmo com o *firewall* programado para filtrar ataques DoS, não foi possível impedir que o servidor *FreeBSD* passasse a negar serviços devido à sobrecarga.

5 – CONSIDERAÇÕES FINAIS

Os *firewalls* dos servidores Slax como no FreeBSD, não conseguiram impedir o ataque DoS. Administradores relatam que ataques DoS são os mais difíceis e, segundo eles, quase "impossível" construir uma barreira impenetrável que consiga barrá-los. Para amenizar os efeitos de ataques DoS, é recomendado que se faça uma barreira desde o provedor de Internet até os servidores locais, e mesmo assim ainda corre-se o risco de sofrer algum tipo de ataque.

Com isto, pode-se comprovar que não existe um *firewall* melhor que outro, porque um *firewall* sozinho não determina a segurança de nenhum sistema. Em algum momento ele se torna falho, e é ai que entram outros sistemas de segurança em conjunto com este para detectar e impedir o ataque.

Os ataques DoS também são difíceis de serem rastreados e bloqueados, porque utilizam técnicas de mesclagem de IPs e de protocolos, por isso é difícil de construir uma barreira eficaz e eficiente contra estes, somente com *firewall*.

Com este estudo foi possível comprovar o que Kurose (2006) afirmou sobre *firewalls*: "[...] *firewalls* não são, de maneira nenhuma, uma solução para os problemas de segurança. Eles introduzem um compromisso entre o grau de comunicação com o mundo exterior e o nível de segurança".

Administradores recomendam ter uma banda de Internet rápida e servidores com processamento suficiente que possam suportar um ataque por um determinado período de tempo até que se tomem medidas para controlar e conter o ataque, garantindo assim que o sistema não venha a paralisar.

REFERÊNCIAS

ABNT. **Tecnologia da informação:** código de prática para gestão da segurança de informações (NBR ISO-IEC 17799/2000). Rio de Janeiro: ABNT, 2000.

BRITO, Eduardo Ferreira, **Sistema operacional GNU/Linux:** um estudo sobre economia, estabilidade e segurança para tratamento das informações de microempresas. 2008, 46 p. Trabalho de Conclusão de Curso (Bacharel em Sistema da Informação) – Faculdade de Minas – FAMINAS - BH, Belo Horizonte, 2008. Disponível em: <http://www.ricardoterra.com.br/publications/students/2008_faminas_brito.pdf> Acesso em: 20 Set. 2012.

GUIAS E DODUMENTOS: **Zabbix Agente in FreeBSD.** 2013. Disponível em: <http://www.packetwatch.net/documents/guides/2010102202.php>. Acesso em: 05 Set. 2013.

HANDBOOK. **FreeBSD Handbook.** Disponível em: < http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/>. Acesso em 20 ago. 2012.

KUROSE, James F. e ROSS, Keith W. **Redes de computadores e a internet**: uma abordagem *top-down.* 3ª Ed. São Paulo: Pearson Addison Wesley, 2006.

LEVENEZ, Eric. **Evolução do Unix**. Disponível em: <http://www.levenez.com/unix/unix_a4.pdf> Acesso em: 22 Ago. 2013.

MARIGO, Renato Diniz. **Zabbix Server 2.0 no Ubuntu Server 12.04:** instalação e configuração. Disponível em: <http://www.vivaolinux.com.br/artigo/Zabbix-Server-20-no-Ubuntu-Server-1204-Instalacao-e-configuracao>. Acesso em: 05 Set. 2013.

MORIMOTO, Carlos E. **Linux – Redes e Servidores**: guia prático. 2ª Ed. Porto Alegre: GDH Press e Sul Editores, 2008.

NETO, Urubatan. **Dominando Linux *Firewall Iptables***. Rio de Janeiro: Editora Ciência Moderna Ltda., 2004.

SILBERSCHATZ, Abraham. **Sistemas operacionais:** conceitos e aplicações. 8ª Ed. Rio de Janeiro: Campus, 2000.

SOUZA, André Luís Wagner de. **Introdução a Redes de Computadores**. Disponível em < http://www.m8.com.br/andre/> Acesso em 21 Ago. 2012.

TANENBAUM, Andrew S. **Sistemas operacionais modernos**. 2ª Ed. São Paulo: Pearson Brasil, 2003.

TANENBAUM, Andrew S. **Redes de computadores**. 4ª Ed. Rio de Janeiro: Elsevier, 2003.

ANEXO I – CONFIGURAÇÃO DO *HARDWARE* DAS MÁQUINAS VIRTUAIS E HOST FÍSICO

Máquina virtual Ubuntu BackTrack 5R3

Hardware:

Processadores.....: 1 (2.4GHz)
Hard Disk (SCSI): 20GB
Memória.....: 728MB
Network Adapter.: Host Only
Sound Card.....: Auto detect
Display.....: Auto detect
CD/DVD (IDE)....: Auto detect
USB Controller....: Present

Máquina virtual Zabbix Server

Hardware:

Processadores.....: 1 (2.4GHz)
Hard Disk (SCSI).: 20GB
Memória.....: 636MB
Network Adapter.: Host Only
Sound Card.....: Auto detect
Display.....: Auto detect
CD/DVD (IDE)....: Auto detect
USB Controller....: Present

Máquina virtual Slax Server

Hardware:

Processadores.....: 1 (2.4GHz)
Hard Disk (SCSI)..: 8GB
Memória.....: 256MB
Network Adapter...: Host Only
Network Adapter 2: Host Only
Sound Card.....: Auto detect
Display.....: Auto detect
CD/DVD (IDE).....: Auto detect
USB Controller.....: Present

Máquina virtual FreeBSD Server

Hardware:

Processadores.....: 1 (2.4GHz)
Hard Disk (SCSI)..: 20GB
Memória.....: 256MB
Network Adapter...: Host Only
Network Adapter 2: Host Only
Sound Card.....: Auto detect
Display.....: Auto detect
CD/DVD (IDE).....: Auto detect
USB Controller.....: Present

Host físico Notebook Dell Inspiron 1545

Hardware:

Processadores.....: 2 (Intel Core Dois Duo - 2.4GHz)
Hard Disk (SCSI)..: 160GB
Memória.....: 4GB
Network Adapter...: Marvell Yukon PCI Fast Ethernet 10/100
Network Adapter 2: Intel Wireless 5100
Sound Card.....: Hight Definition Sound Card
Display.....: LCD
CD/DVD (IDE).....: DVD RW
USB Controller.....: USB Controler 2.0 Rev 8.0

ANEXO I I – CONFIGURAÇÃO DOS *SOFTWARE*S DAS MÁQUINAS VIRTUAIS E HOST FÍSICO

Máquina virtual Ubuntu BackTrack 5R3

*Software*s:

Pacote BackTrack...: netdiscover, zenmap, t50, hping3, inundator
Banco de dados....: MySQL Server 5.2, Postgree
PHP.....: PHP5
Front-end.....: Apache2

Máquina virtual Zabbix Server

*Software*s:

Banco de dados....: MySQL Server 5.2
PHP.....: PHP5
Front-end.....: Apache2
Monitoração.....: Zabbix Server 2.0.8

Máquina virtual Slax Server

*Software*s:

Front-end.....: Apache2
SSH.....: SSH2
Monitoração.....: Zabbix Agent for Linux 2.0.6

Máquina virtual FreeBSD Server

*Software*s:

Front-end.....: Apache2
SSH.....: SSH2
Monitoração.....: Zabbix Agent for FreeBSD 1.8.0

Host físico Notebook Dell Inspiron 1545

*Software*s:

Sistema Operacional..: Ubuntu 13.04 - 64bits
Banco de dados.....: MySQL Server 5.2
PHP.....: PHP5
Front-end.....: Apache2
Monitoração.....: Zabbix Server 2.0.8
Virtualização.....: Vmware Player 5.0.2 build-1031769

ANEXO III – INSTALAÇÃO ZABBIX SERVER E AGENTES

1 - Instalação do Zabbix Server no Ubuntu 12.1

1.1 - Preparando o sistema para a instalação.

- Banco de dados MySQL.
- Front-end Apache 2.
- PHP 5 e extensões do PHP.

Levando em consideração que seu apt-get está com os repositórios configurados:

 # apt-get update

1.2 - Instalando pacotes e as dependências

 # apt-get install mysql-server apache2 php5 php5-curl php5-dev php5-mysql php5-gd php5-xmlrpc openipmi libssh2-1 libssh2-1-dev libssh2-php fping libcurl3 libiksemel3 libiksemel-dev snmplib mysqld-dev libmysqld-piclib mysqlclient-dev make

2 - Instalando o Zabbix Server

2.1 - Baixar osource do Zabbix em:

http://www.zabbix.com/download.php

2.2 - Crie um diretório **/srv/zabbix/** e copie o source do Zabbix. Depois, extraia os arquivos:

 # tar -xvzf zabbix-[Versão].tar.gz

2.3 - Crie o usuário "zabbix" em seu sistema:

 # groupadd zabbix
 # useradd -g zabbix zabbix

3 - Preparando o banco de dados

3.1 - Preparar o Banco de dados doMySQL

3.1.1 - Entrar no MySQL e digitar a senha:

 # mysql -u root -p

3.1.2 - Criar o banco de dados:

mysql> create database zabbixdb;
mysql> quit;

Obs: Não esquecer o “ **;** ” no final de cada comando.
3.1.3 - Configurar a permissão ao usuário "**zabbix**":

 # mysql -u root -p -e "grant all privileges on zabbixdb.* to zabbix@localhost identified by 'zabbix';"

3.1.4 – Preparando o banco de dados do MySQL. Vá até o diretório descompactado do Zabbix (**/srv/zabbix/zabbix-2.0.8/database/mysql**), que possui os arquivos ".sql", e estando neste diretório, digite:

```
# mysql -u zabbix -p zabbixdb < schema.sql
# mysql -u zabbix -p zabbixdb < images.sql
# mysql -u zabbix -p zabbixdb < data.sql
```

Agora o banco de dados está preparado para instalação do Zabbix.

4 – Instalação do Zabbix

4.1 - Configurando os pacotes (sources)

Dentro do diretório do Zabbix (**/srv/zabbix/zabbix-2.0.8/**), compile-o com os seguintes parâmetros:

```
# ./configure --enable-server --enable-agent --with-mysql --enable-ipv6 --with-
snmp --with-libcurl3 --with-ssh2
```

Depois de compilá-lo com os parâmetros necessários execute o comando:

```
# make install
```

4.2 – Configurando os serviços

Verifique se as linhas **zabbix-agent** e **zabbix-trapper** estão no arquivo **/etc/services**, caso não esteja, adicione ao final do arquivo, as seguintes linhas:

```
zabbix-agent        10050/tcp    #Zabbix Agent
zabbix-agent        10050/udp    #Zabbix Agent
zabbix-trapper      10051/tcp    #Zabbix Trapper
zabbix-trapper      10051/udp    #Zabbix Trapper
```

Edite as seguintes linhas do arquivo **/usr/local/etc/zabbix_agentd.conf**:

```
PidFile=/tmp/zabbix_agentd.pid
LogFile=/tmp/zabbix_agentd.log
LogFileSize=1
DebugLevel=3
EnableRemoteCommands=1
LogRemoteCommands=1
Server=127.0.0.1
ListenPort=10050
Hostname=Zabbix Server
```

Edite as seguintes linhas do arquivo **/usr/local/etc/zabbix_server.conf**:

```
ListenPort=10051
LogFile=/tmp/zabbix_server.log
LogFileSize=2
PidFile=/tmp/zabbix_server.pid
DBHost=localhost
```

DBName=zabbixdb
DBUser=zabbix
DBPassword= zabbix
StartIPMIPollers=1
StartDiscoverers=5
Timeout=3
FpingLocation=/usr/bin/fping

5. Configurando o front-end PHP – Interface WEB

5.1 – Testando conexão com o ***Apache***

Para saber se o *Apache* está funcionando, digite em um navegador de internet o **IP do Servidor** ou ***localhost***:

http://localhost ou http://127.0.0.1

Deverá aparecer:

It works!
This is the default web page for this server.
The web server software is running
but no content has been added, yet

5.2 – Ajustando as configurações do PHP

É necessário ajustar algumas informações do PHP, para os pré-requisitos do Zabbix. Para isto edite o arquivo "**/etc/php5/apache2/php.ini**", alterando as seguintes opções:

post_max_size = 16M
max_execution_time = 300
max_input_time = 300
date.timezone = American/Sao_Paulo

Após editar o arquivo, reinicie o Apache:

 # /etc/init.d/apache2 restart

6 - Configurando a inicialização do sistema

6.1 - Entre no diretório **/srv/zabbix/zabbix-2.0.8/misc/init.d/debian** e copie os arquivos para "**/etc/init.d**":

 # cp zabbix-agent /etc/init.d
 # cp zabbix-server /etc/init.d

6.2 - Dê permissão de execução para estes arquivos:

 # chmod +x /etc/init.d/zabbix-server /etc/init.d/zabbix-agent

6.3 - Inicie os serviços:

 # /etc/init.d/zabbix-server start
 # /etc/init.d/zabbix-agent start

6.4 - Verifique se os processos estão rodando:

 # ps –ef | grep zabbix

Deverão aparecer as seguintes linhas:

```
zabbix  2189     1  0 01:13 ?  00:00:00 /usr/local/sbin/zabbix_server
zabbix  2191  2189  0 01:13 ?  00:00:00 /usr/local/sbin/zabbix_server
zabbix  2192  2189  0 01:13 ?  00:00:00 /usr/local/sbin/zabbix_server
zabbix  2193  2189  0 01:13 ?  00:00:00 /usr/local/sbin/zabbix_server
zabbix  2194  2189  0 01:13 ?  00:00:00 /usr/local/sbin/zabbix_server
```

6.5 - Atualize os arquivos de inicialização do sistema:

 # update-rc.d -f zabbix-server defaults
 # update-rc.d -f zabbix-agent defaults

Como estamos configurando um servidor Ubuntu, o diretório default do Apache é **/var/www**. É aconselhável criar um diretório **zabbix**:

 # mkdir /var/www/zabbix/

Entre no diretório dos arquivos fontes **/srv/zabbix/zabbix-2.0.8/frontends/php** e copie todo o conteúdo para **/var/www/zabbix/**:

 # cp -a * /var/www/zabbix/
 # chown -R www-data:www-data /var/www/zabbix/

Agora, já pode acessar o Zabbix através da URL:

http://127.0.0.1/zabbix ou http://localhost/zabbix

Após toda preparação, é necessário terminar a instalação pela interface gráfica como mostrado nas figuras que se seguem.

A figura 18 mostra a tela de boas vindas da instalação gráfica.

Figura 18: Tela de abertura da interface gráfica.
Fonte: Próprio autor.

A tela de checagem dos pré-requisitos mostra se todas as configurações e dependências estão em ordem, como é possível ver na figura 19.

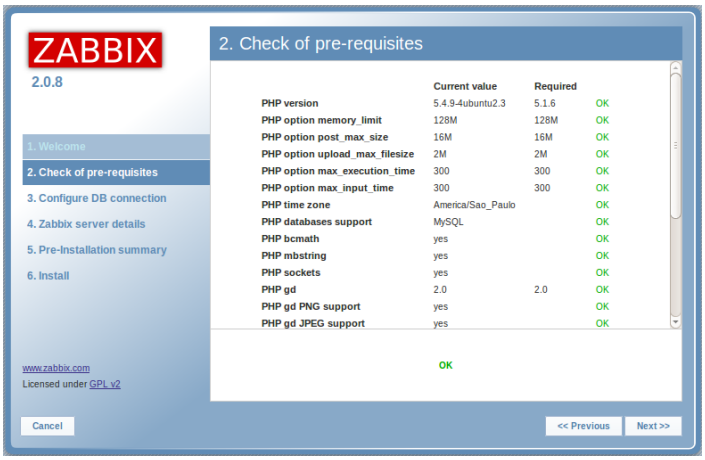

Figura 19: Checagem dos pré-requisitos.
Fonte: Próprio autor.

A figura 20 mostra a tela de configuração da conexão com o banco de dados do MySQL.

Figura 20: Configuração da conexão com o Banco de Dados do MySQL.
Fonte: Próprio autor.

Na tela de detalhes do zabbix server, deve-se dar um nome ao servidor como mostrado na figura 21:

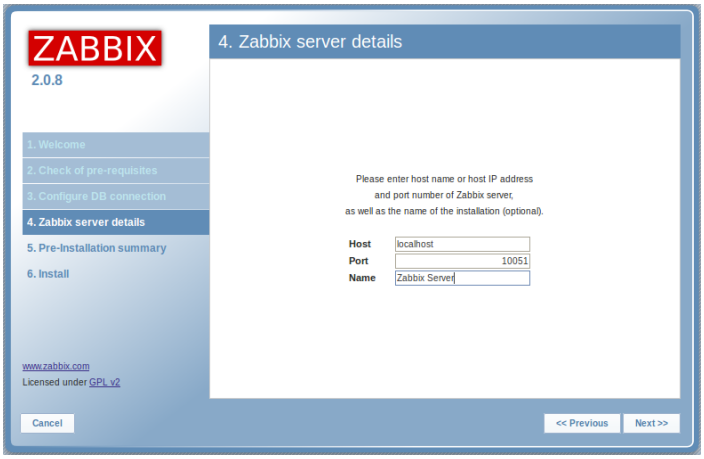

Figura 21: Identificação do Servidor.
Fonte: Próprio autor.

Na figura 22, é possível observar o sumário da pré-instalação do zabbix.

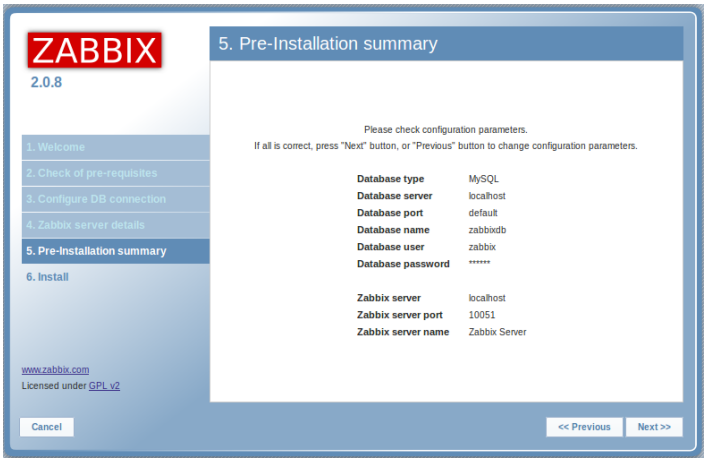

Figura 22: Sumário do que foi configurado.
Fonte: Próprio autor.

Se tudo estiver correto aparecerá à tela de “Instalação do *frontend* zabbix efetuada com sucesso” como mostrado na figura 23.

Figura 23: Término da Instalação.
Fonte: Próprio autor.

Na tela de login como mostrado na figura 24, o *Username* é "**admin**" e a *Password* é "**zabbix**".

Figura 24: Tela de Login.
Fonte: Próprio autor.

Configuração dos Agentes Zabbix

Instalação do Agente Zabbix no GNU/Linux Slax - Instalação e configuração

Primeiro passo

1 - Baixar o source do Zabbix, em:

http://www.zabbix.com/download.php

2 - Crie o diretório **/etc/zabbix/** e copie o source do Zabbix.

2.1 - Extraia os arquivos:

 # tar -xvzf zabbix-[Versão].tar.gz

3 - Crie o usuário e o grupo **zabbix**.

 # groupadd zabbix
 # useradd -g zabbix zabbix

4 – Configuração do agente

4.1 - Mude as permissões do arquivo de configuração do agente

 # chown zabbix:zabbix /etc/zabbix/zabbix-2.0.6/conf/zabbix_agentd.conf

4.2 - Edite estas configurações mínimas no arquivo /**etc/zabbix/zabbix-2.0.6/conf/zabbix_agentd.conf**

Server=172.16.0.3
ListenPort=10050
StartAgents=3
ServerActive=172.16.0.3
Hostname=Slax-Server
Timeout=3

5 - Edite o arquivo **/etc/rc.d/rc.local** para iniciar o serviço automaticamente inserindo a linha de comando mostrada no final do arquivo.

 # Inicia o Agente Zabbix
/etc/zabbix/zabbix-2.0.6/sbin/zabbix_agentd -c /etc/zabbix/zabbix-2.0.6/conf/zabbix_agentd.conf

6 – Liberando acesso no *firewall* para o agente zabbix

 # Liberando acesso ao zabbix server
$IPTABLES -A INPUT -p tcp --syn -s 192.168.0.4 --dport 10050 -j ACCEPT
$IPTABLES -A OUTPUT -p tcp --syn -s 192.168.0.3 --dport 10050 -j ACCEPT
$IPTABLES -A FORWARD -p tcp --syn -s 192.168.0.4 --dport 10050 -j ACCEPT

Instalação do Agente Zabbix no UNIX FreeBSD

1 - Instalação do Agente Zabbix

#pkg_add -r zabbix-agent

2 - Crie o grupo **zabbix** e o usuário **zabbix** no grupo

 # groupadd zabbix
 # useradd -g zabbix zabbix

3 - Verifique se estas linhas foram adicionadas no arquivo **/etc/services** durante a instalação. Caso não esteja, adicione-as.

zabbix-agent 10050/tcp #Zabbix Agent
zabbix-trap 10051/tcp #Zabbix Trap
zabbix-agent 10050/udp #Zabbix Agent

zabbix-trap 10051/udp #Zabbix Trap

4 - Mude as permissões dos arquivos:

 # chown zabbix:zabbix /usr/local/etc/zabbix/zabbix_agent.conf
 # chown zabbix:zabbix /usr/local/etc/zabbix/zabbix_agentd.conf

5 - Configurando o agente.

5.1 - Edite estas configurações no arquivo **/usr/local/etc/zabbix/zabbix_agent.conf**

Server=172.16.0.3
Timeout=3

5.2 - Edite estas configurações no arquivo **/usr/local/etc/zabbix/zabbix_agentd.conf**

PidFile=/tmp/zabbix_agentd.pid
LogFile=/tmp/zabbix_agentd.log
LogFileSize=4
SouceIP=172.16.0.3
Server=172.16.0.3
ServerActive=172.16.0.3:10050
Hostname=FreeBSD-Server
ListenPort=10050
StartAgents=3
Timeout=3

5.3 - Edite o arquivo **/etc/rc.conf** para que o agente zabbix inicie automaticamente, adicionando a seguinte linha:

zabbix_agent_enable="YES"

6 - Iniciando o serviço

 # /usr/local/etc/rc.d/zabbix_agentd start

Este comando deve retornar

Starting zabbix_agentd

6.1 - Visualize se o processo está em execução:

 # /usr/local/etc/rc.d/zabbix_agentd status

Este comando deve retornar

zabbix_agentd is runing as pid 723 731 732 733 734 735

7 - Liberando o acesso do agente zabbix pelo *firewall*

```
# Libera conexao do agente zabbix
${fwcmd} add 1300 allow ip from 192.168.0.4 to 192.168.0.2 10050 via em0
${fwcmd} add 1400 allow ip form 192.168.0.2 10050 to 192.168.0.4 via em0
```

ANEXO IV – CONFIGURAÇÃO DOS *FIREWALL*– IPTABLES E IPFIREWALL

Configuração do *Firewall* IPtables – Arquivo **rc.*firewall***

```
#!/bin/bash
#
# usage: rc.firewall start|stop|status
...

ALLOWED_PORTS="22 80 10050 10051"
#---------------------------------------------------------

if [ "$1" = "start" ]; then

SYSCTLW="/sbin/sysctl -q -w"
IPTABLES="/usr/sbin/iptables"

$SYSCTLW net.ipv4.ip_forward=1

# Firewall initialization, remove everything, start with clean tables
$IPTABLES -F      # remove all rules
$IPTABLES -X      # delete all user-defined chains

for PORT in $ALLOWED_PORTS; do
   $IPTABLES -A INPUT -p tcp --dport $PORT -j ACCEPT
   $IPTABLES -A OUTPUT -p tcp --dport $PORT -j ACCEPT
done

# Liberando acesso ao zabbix server
$IPTABLES -A INPUT -p tcp --syn -s 192.168.0.4 --dport 10050 -j ACCEPT
$IPTABLES -A OUTPUT -p tcp --syn -s 192.168.0.3 --dport 10050 -j ACCEPT
$IPTABLES -A FORWARD -p tcp --syn -s 192.168.0.4 --dport 10050 -j ACCEPT

# Ignora Pings
$IPTABLES -A INPUT -p icmp --icmp-type echo-request -j DROP
$IPTABLES -A OUTPUT -p icmp --icmp-type echo-request -j ACCEPT

# Proteção contra syn flood
$IPTABLES -A FORWARD -p tcp --syn -m limit --limit 2/s -j ACCEPT

# Proteção contra ping da morte
$IPTABLES -A FORWARD -p icmp --icmp-type echo-request -m limit --limit 1/s -j
ACCEPT

# Proteção contra scanners de rede
$IPTABLES -A FORWARD -p tcp --tcp-flags SYN,ACK,FIN,RST RST -m limit
  --limit 1/s -j ACCEPT
```

```
# Protege contra IP-Spoofing
$IPTABLES -A FORWARD -s 192.168.0.3 -i eth0 -j DROP
$IPTABLES -A FORWARD -s 192.168.0.3 -j DROP

# Descarta pacotes mal formados (ataques DoS)
$IPTABLES -A INPUT -m state --state INVALID -j DROP

# Abre interface de loopback, essencial para que programas gráficos funcionem.
$IPTABLES -A INPUT -i lo -j ACCEPT

# Impede a abertura de novas conexões, efetivamente bloqueando o acesso externo ao
servidor, com exceção das portas e faixas de endereço.
$IPTABLES -A INPUT -p tcp --syn -j DROP

echo "Regras de Firewall ativadas"

elif [ "$1" = "stop" ]; then
iptables -F
iptables -X
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P INPUT ACCEPT

elif [ "$1" = "status" ]; then
iptables -L -v

else
echo "usage: $0 start|stop|status"
fi
```

Configuração do *Firewall IPFirewall* (ipfw) do FreeBSD - Arquivo **rc.*firewall***

```
#!/bin/sh
fwcmd="/sbin/ipfw"

# Limpa as regras de firewalls existentes
${fwcmd} -q flush

# Proteção contra IP-Spoofing
${fwcmd} add 0100 deny log not verrevpath in

# Bloquear pedidos de echo e registro de log
${fwcmd} add 0200 deny log icmp from any to 192.168.0.2 in icmptypes 8

# Liberando o acesso localhost para processos internos
${fwcmd} add 0300 allow all from any to any via lo0
${fwcmd} add 0400 deny all from any to 127.0.0.0/8

# Proteção contra ping da morte...
${fwcmd} add 0500 reject icmp from any to 192.168.0.2 out icmptypes 0
```

```
${fwcmd} add 0600 reject icmp from any to 192.168.0.2 in icmptypes 8

# Proteção contra pacotes fragmentados (Defesa contra ataque DoS)
${fwcmd} add 0700 deny log all from any to any in frag

# Bloqueando IP's ou portas perigosas...
${fwcmd} add 0800 deny log tcp from any to any 57,77,79,87,144,194,209,469,666,750
${fwcmd} add 0900 deny log tcp from any to any
51,754,760,761,1525,1989,1992,2049,2105,3306
${fwcmd} add 1000 deny log udp from any to any 57,77,79,87,144,194,209,469,666,750
${fwcmd} add 1100 deny log udp from any to any
751,754,760,761,1525,1989,1992,2049,2105,3306

# Libera tráfego proveniente da intranet
${fwcmd} add 1200 allow all from 192.168.0.2 80 to any

# Libera conexao ao zabbix server
${fwcmd} add 1300 allow ip from 192.168.0.4 to 192.168.0.2 10050 via em0
${fwcmd} add 1400 allow ip form 192.168.0.2 10050 to 192.168.0.4 via em0

# Libera acesso do ping para fora da rede
${fwcmd} add 1500 allow icmp from 192.168.0.2 to any

# Permite gravacao de log
${fwcmd} add 1600 allow log logamount 500 ip from any to any
```