CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Segurança da Informação

José Ricardo Rodrigues

Segurança do FreeBSD atuando como Gateway

Americana, SP

2014

CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Segurança da Informação

José Ricardo Rodrigues

Segurança do FreeBSD atuando como Gateway

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso de Tecnologia em Segurança da Informação, para obtenção do título de Tecnólogo em Segurança da Informação

Orientador: Prof. Marcus Vinícius Lahr Giraldi

Americana, SP

2014

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

Rodrigues, José Ricardo
R613s Segurança do FreeBSD: atuando como gateway. / José Ricardo Rodrigues. – Americana: 2014.
62f.

Monografia (Graduação de Tecnologia em Segurança da Informação). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.
Orientador: Prof. Marcus Vinícius Lahr Giraldi

1.Segurança em Sistemas de informação 2. Sistemas operacionais I. Giraldi, Marcus Vinícius Lahr II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.518.5
681.3.066

José Ricardo Rodrigues

Segurança do FreeBSD atuando como Gateway

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Segurança da Informação pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.

Americana, 25 de junho de 2014.

Banca Examinadora:

Marcus Vinícius Lahr Giraldi (Presidente)
Graduado
FATEC Americana

Alexandre Garcia Aguado (Membro)
Mestre
FATEC Americana

Luiz Carlos Degrande Júnior (Membro)
Graduado
FATEC Americana

AGRADECIMENTOS

Em primeiro lugar a Deus, a minha família e namorada. Agradeço também ao orientador Marcus Lahr, que teve participação decisiva no projeto prático.

RESUMO

Tendo em vista o avanço tecnológico, a popularização da Internet e a exploração de falhas de segurança nos protocolos de comunicação, fez com que as empresas tomassem medidas necessárias para proteção de seus dados.

O Gateway de Perímetro é um dos exemplos importantes, pois ele atua como “porta de entrada” e “porta de saída” em uma rede computacional, e requer atenção especial no que diz respeito à segurança.

Este trabalho realiza um estudo sobre as características e conceitos do sistema operacional FreeBSD para atuar como um Gateway seguro. Além disso, também são apresentados outros conceitos relacionados demonstrando conhecimentos importantes para compreensão do funcionamento de um Gateway.

Palavras Chave: *FreeBSD; Firewall; IPFW*.

ABSTRACT

Given the technological advances, the popularization of the Internet and how to exploiting security flaws in communication protocols, meant that companies take measures to protect data. The Perimeter Gateway is one of the important examples, because it acts as a "gateway" and "way out" in a computer network, and requires special attention with regard to safety.
This paper conducts a study on the features and concepts of the FreeBSD operating system to act as a secure gateway. Moreover, they are also related concepts presented demonstrating significant knowledge to understanding the operation of a Gateway.

Keywords *FreeBSD; Firewall; IPFW*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO

Tendo em vista o avanço tecnológico, computadores tornando-se mais acessíveis devido ao preço e a popularização da Internet, as empresas precisaram se adequar devido à informatização iminente. Não apenas a necessidade imposta pela natureza da evolução tecnológica, o computador tornou-se uma ferramenta indispensável para realizar praticamente qualquer tarefa, como uma emissão de nota fiscal, envio e recebimento de email, acesso remoto às câmeras da empresa, vendas, pagamentos etc.

Entretanto, a exploração de falhas de segurança dos protocolos de comunicação, códigos de sistemas e banco de dados, fez com que as empresas tomassem medidas necessárias para proteção da informação como o bem mais valioso. Por meio da política de segurança da informação, que traz consigo as medidas de segurança necessárias para a organização além da reeducação dos funcionários para com o uso consciente dos computadores, são bons exemplos utilizados em grandes corporações que obtiveram resultados positivos na redução de incidentes de segurança.

Todavia, devido ao fato de que “uma rede totalmente segura não existe”, pois a partir do momento que computadores são ligados em rede, eles passam a ser alvo dos mais variados tipos de ataques, vindos tanto da rede externa (Internet) como da própria rede interna, e passa a existir a possibilidade das informações serem roubadas ou destruídas. Cabe ao administrador de rede e especialistas em segurança criar ou utilizar mecanismos para dificultar o trabalho dos invasores (NAKAMURA e GEUS, 2007).

Um Gateway de Perímetro é um dos exemplos primários em uma política de segurança da informação, pois ele atua como “porta de entrada” e “porta de saída” em uma rede computacional, e requer atenção especial no que diz respeito à segurança. Não basta apenas o administrador de rede pensar em uma configuração avançada de Firewalls deixando de se preocupar com a proteção dos arquivos vitais do sistema operacional atuante. Isto significa utilizar recursos disponíveis do sistema para proteger os arquivos binários, por exemplo, além de aprimorar o controle de acesso.

O objetivo geral deste trabalho foi realizar um estudo sobre as principais características de segurança do sistema *FreeBSD* para atuar como um Gateway, bem como realizar um estudo prático apresentando o funcionamento e eficiência dos conceitos mencionados.

Como objetivos específicos, este trabalho tem o intuito de:

- Apresentar conceitos relacionados ao *FreeBSD;*
- Apresentar conceitos relacionados à segurança da informação;
- Conceituar de uma forma geral o firewall;
- Apresentar alguns dispositivos nativos de segurança do FreeBSD utilizados para atuar como um Gateway seguro;
- Projetar e desenvolver um servidor gateway com *FreeBSD*, bem como realizar testes de segurança.

Este trabalho está organizado em 5 capítulos, a saber:

- Capítulo 1 – Apresenta a contextualização do tema, além dos objetivos e a organização estrutural do trabalho;
- Capítulo 2 – Apresenta conceitos importantes utilizados para compreensão do leitor;
- Capítulo 3 – Trata dos dispositivos nativos de segurança do sistema, como o IPFW e níveis de segurança do kernel, por exemplo;
- Capítulo 4 – Apresenta ao leitor o funcionamento do servidor com base nos conceitos teóricos apresentados nos capítulos anteriores, além de testes de segurança realizados para comprovar a eficiência do FreeBSD;
- Capítulo 5 – Apresenta a conclusão do trabalho mostrando os pontos importantes do estudo realizado sobre o FreeBSD, além de propostas de continuidade no campo de pesquisa.

2. CONCEITUALIZAÇÃO E TERMINOLOGIA

Neste capítulo são apresentados conceitos fundamentais para a compreensão da área de pesquisa e do projeto apresentados.

2.1. O *FreeBSD*

Segundo Lucas (2002), *FreeBSD* é um sistema operacional *UNIX-Like*[1] baseado diretamente do *UNIX* original produzido pela AT&T em 1970, disponível gratuitamente e utilizado extensivamente por provedores de Internet (ISP), dispositivos embarcados e em qualquer ambiente onde a confiabilidade é fundamental.

Segundo *FreeBSD* (2013), o sistema tem muitas características valiosas. Algumas destas são:

- Multitarefa preemptiva com ajustes dinâmicos de prioridade, que garantem o compartilhamento claro e racional do computador entre as aplicações e usuários, mesmo sob a mais intensa demanda;
- Caracteristicas multi-usuário que permite várias pessoas utilizarem um sistema *FreeBSD* de forma simultânea, para uma variedade de coisas. Isto implica, por exemplo, que os periféricos do sistema, como impressoras, serão apropriadamente compartilhadas entre todos os usuários no sistema ou na rede, e que limites individuais possam ser definidos para usuários e grupos de usuários, protegendo recursos críticos do sistema evitando sobrecarga;
- Forte rede TCP/IP com suporte a diversos padrões. Isto significa que uma estação *FreeBSD* pode interagir facilmente com outros sistemas, da mesma forma que pode agir como um servidor corporativo, oferecendo funções vitais como NFS (acesso remoto à arquivos), serviços de correio eletrônico, WWW, FTP, roteamento e firewall (segurança);

---

[1] *UNIX-Like* é um sistema similar ao *UNIX*, não estando necessariamente de acordo com o *Single UNIX Specification*.

- Proteção de memória garante que aplicações ou usuários não interferirão entre si, ou seja, a falha de uma aplicação não afetará outras de forma alguma;
- Compatibilidade binária com quaisquer programas compilados para Linux;
- Milhares de aplicações prontas para imediata utilização (*ready-to-run*) estão disponíveis a partir da coleção de ***ports*** e ***packages*** do *FreeBSD*;
- Memória virtual paginada por demanda e uma concepção eficiente (*Merged VM/buffer cache*) que satisfaz a necessidade de recursos de aplicações;
- Suporte *SMP* (*Symmetric Multi-Processing*) para máquinas com múltiplas CPUs;
- Conjunto completo de ferramentas de desenvolvimento em linguagem *C*, *C++*, *Fortran*, e *Perl*. Muitas linguagens adicionais para pesquisa e desenvolvimento avançado também estão disponíveis na coleção de *ports* e *packages*.

A versão 5.0-*RELEASE*, foi anunciada em 19 de Janeiro de 2003. Segundo FreeBSD (2013), esta versão foi o resultado culminante de aproximadamente três anos de trabalho e colocou o *FreeBSD* no caminho do suporte avançado a multiprocessamento simétrico, suporte avançado a aplicações *multithread*[2] e apresentou ao público suporte às plataformas UltraSPARC (*Sun Microsystems*) e ia64 (*Intel*). Esta versão foi seguida pela 5.1 em Junho de 2003. Além de um número muito grande de novas funcionalidades, as versões 5.X do *FreeBSD* contém ainda uma série de trechos em desenvolvimento em todas as arquiteturas de sistemas relacionadas. Por tal razão, as versões 5.X são consideradas versões de “Nova Tecnologia”, enquanto a série 4.X atua como versões de “Produção”.

Atualmente o *FreeBSD* se encontra na versão 10.0 com algumas novidades e melhorias a seguir:

[2] Multithreading é a capacidade que o sistema operacional possui de executar, dentro de um processo, várias threads simultaneamente sem que uma interfira na outra.

- Novo instalador, tornando o processo de instalação mais intuitivo;
- O sistema de arquivos ZFS foi atualizado, bem como os drivers ATA/SATA, que agora suportam AHCI[3];
- Melhorias no protocolo TCP/IP;
- Arquitetura PowerPC incorporada ao sistema com suporte ao *PlayStation* 3, da Sony®.

2.1.2. O Kernel do sistema

De acordo com Lucas (2002), o *Kernel* representa a interface entre hardware e software. Além disso, o *Kernel* permite gravar dados em unidades de disco e rede, lidando com as operações de memória e processador. Ele traduz, por exemplo, um arquivo mp3 a um fluxo de dados em números binários para que a placa de som possa entender. Com isso, o *Kernel* se torna responsável por fornecer interfaces para programas que necessitam de acesso aos periféricos (*hardware*) do computador.

Segundo Lucas (2002), o *Kernel* do *FreeBSD* pode ser ajustado dinamicamente ou alterado em tempo real, melhorando aspectos como o desempenho do sistema conforme a necessidade. Ao mesmo tempo, algumas partes do *Kernel* não podem ser alteradas durante sua execução, e alguns recursos do *Kernel* exigem extensa reconfiguração. Além disso, é possível reduzir o tamanho do *Kernel* removendo componentes desnecessários e aumentando ainda mais o desempenho e a segurança do sistema.

O *FreeBSD* fornece duas maneiras de se configurar o Kernel: usando a **sysctl** ou o através do carregamento de boot – *o* ***loader.conf***.

2.1.3. sysctl

De acordo com Lucas (2002), o programa sysctl permite buscar, observar e, em alguns casos, mudar valores no *Kernel*. Com o sysctl, é possível fazer

[3] Advanced Host Controller Interface (AHCI) é a tecnologia presente nos chipsets Intel/AMD mais modernos. Oferece suporte a NCQ que permite que as unidades SATA aceitem mais de um comando por vez e os reorganizem para máxima eficiência. Além disso, oferece suporte à troca quente de dispositivos, e suporte a rotações escalonadas entre várias unidades no momento da inicialização.

modificações no núcleo do sistema em tempo real sem a necessidade de recompilar o *Kernel* configurando as opções estaticamente. Entretanto, é recomendado que se use o sysctl com cautela, evitando mudar valores que podem prejudicar parte ou todo o sistema operacional (FREEBSD, 2013).

Segundo *FreeBSD* (2013), para listar todas as variáveis sysctl utiliza-se o seguinte comando:

sysctl –a

Para configurar um valor em uma variável em particular, usa-se a sintaxe *variável=valor*, por exemplo:

sysctl kern.maxfiles=5000

Segundo Lucas (2002), a organização das sysctls são classificadas em árvore chamada de Gestão da Base de Informações (MIB), que abrange diversas categorias tais como *net*, *vm*, *kern* etc. Cada uma dessas categorias são subdividas, por exemplo, a categoria *net* abrange todas as sysctl's de rede e são divididas em categorias como IP, TCP, UDP e ICMP.

Cada MIB tem um valor que representa uma configuração ou característica utilizada pelo *Kernel*. Ao alterar um valor, muda-se a forma como o *Kernel* funciona. Por exemplo, algumas sysctls controlam a quantidade de memória que é usada para cada conexão de rede. Se o desempenho da rede é precário, é possível aumentar o desempenho do sistema reservados para conexões de rede.

Algumas das MIBs sysctl estão listadas na tabela 1:

Sysctl	Função
Kern	Funções relacionadas ao kernel
Vm	Funções relacionadas à memória virtual do sistema
Vfs	Funções relacionadas ao sistema de arquivos
Net	Funções relacionadas à rede
debug	Informações de depuração
Hw	Informações sobre hardware
Machdep	Variáveis que dependem da plataforma do processador (por exemplo, Alpha, i386, etc).

Tabela 1: Algumas MIBs da árvore systcl
Fonte: Adaptado de LUCAS (2002)

Segundo Lucas (2002), cada valor sysctl pode ser uma *string*, inteiro, binário ou opaco. Ou seja, *strings* são texto de formato livre e comprimento arbitrário; inteiros são números inteiros ordinários; binários são 0 (desligado) e 1 (ligado); e opacos estão em código de máquina e apenas programas especializados podem interpretá-los.

Além disso, sysctls podem ser ajustadas para inicializar no carregamento do sistema operacional. Isto é feito editando o arquivo *sysctl.conf* em */etc/sysctl.conf* indicando o nome da systcl seguido de um valor específico. Por exemplo:

vfs.usermount=1

De acordo com Lucas (2002), esta sysctl permite aos usuários montarem sistemas de arquivos mudando o valor binário de 0 para 1.

2.1.4. Configuração do kernel com *loader.conf*

De acordo com Lucas (2002), o *loader.conf* é um arquivo de configuração onde se torna possível acrescentar ou retirar módulos do *Kernel* para serem

carregados durante o processo de boot antes da inicialização do sistema. Este arquivo de configuração tem duas funções principais: carregar módulos de *Kernel* e oferecer dicas para drivers de dispositivo. Segundo Lucas (2002), essas dicas de driver geralmente são MIBs *sysctl* que só podem ser ajustadas no boot, e não em tempo real.

O *loader.conf* se torna bastante útil por dois motivos: tornar possível carregar o *Kernel* mais "enxuto" subindo somente os módulos necessários para o uso, e mudar algumas configurações no *Kernel* que não podem ser feitas com o sistema operacional já carregado utilizando o comando sysctl. Lucas (2002) apresenta um exemplo de uma sysctl que ativa ou desativa a gravação em cache de um disco rígido IDE[4]:

hw.ata.wc=0

No exemplo acima, a gravação em cache está apontando para 0 (desligada). Para habilitar, basta alterar o valor 0 para 1 (ligada). Além disso, esta sysctl só pode ser alterada no boot do sistema ou incluir uma linha contendo essa sysctl no arquivo *bootloader.conf* (LUCAS, 2002).

Segundo Lucas (2002), essa característica de poder carregar ou descarregar porções (módulos) de *Kernel*, economiza memória e aprimora a flexibilidade do sistema operacional.

2.2. Proxy

Segundo Nakamura e Geus (2007), os proxies são sistemas que atuam como um *gateway* entre duas redes, permitindo ou negando as requisições dos usuários internos de acordo com a política de segurança definida. Além disso, o proxy realiza uma filtragem mais apurada dos pacotes da rede, por atuar na camada de aplicação do modelo *Organization Standardization / Open Systems Interconnection* (ISO/OSI).

O esquema de funcionamento típico dos proxies faz com que o cliente deva, primeiramente, se conectar ao proxy, que libera acesso após a autenticação. Após esta etapa, o cliente envia sua requisição ao proxy, que a retransmite ao servidor

[4] Unidades IDE de gravação em cache possuem um chip onboard pequeno, que grava os dados que precisam ser passados para o disco. Ou seja, primeiramente a informação é armazenada no cache da unidade para depois ser gravada definitivamente no disco.

web, por exemplo. A resposta do servidor web também passa pelo proxy, que funciona como uma porta entre o cliente e o servidor. De acordo com Nakamura e Geus (2007), duas conexões são iniciadas em cada requisição: uma do cliente até o proxy, e outra do proxy ao servidor web. Isto protege o cliente e o servidor por meio do controle de requisição de serviços, proibindo determinados eventos no nível de aplicação.

Alguns proxies também podem realizar a função de cache, ou seja, toda requisição do cliente permitida pelo proxy armazena em disco os objetos das páginas web recentemente visitadas. Isto faz com que se obtenha desempenho e economia no consumo de banda larga pois, caso o cliente volte a visitar a mesma página requisitada anteriormente, o proxy irá pesquisar primeiramente no seu cache e devolverá ao cliente os objetos da página ao invés de requisitar todo o conteúdo novamente na internet.

2.2.1. Squid Cache

Segundo Pacheco (2007), o *Squid* é um servidor proxy *cache* de alto desempenho que suporta os protocolos HTTP, HTTPS, FTP, TLS e SSL, reduz o consumo de banda e melhora os tempos de resposta de páginas solicitadas que estão em *cache*. O *Squid* possui grande controle de ACLs (Listas de controle de acesso) permitindo ou negando acesso à determinadas páginas web, autenticando usuários cadastrados, bloqueando downloads de extensões de arquivos, além de ser muito flexível tendo em vista diversas opções de customização. Funciona em plataformas Linux, Unix e Windows e é licenciado sob o GNU GPL (Licença Pública Geral – software livre).

Segundo Zanoni (2007), a configuração do *Squid* fica gravada no arquivo ***squid.conf***. A seguir algumas opções que definem o funcionamento básico do proxy *Squid:*

- **http_port n**: Utilizada para definir em quais portas (n) o *Squid* espera por conexões HTTP. A porta padrão é 3128, mas é possível especificar uma outra qualquer, dependendo da necessidade.
- **cache_dir Tipo diretório Mbytes Nível-1 Nível-2:** Esta opção serve para definir em quais diretórios serão armazenados os objetos de cada página web. Em **“Tipo”** especifica o sistema de armazenamento a ser

utilizado. Atualmente o tipo que pode ser utilizado com segurança é **ufs**. **“Diretório”** especifica o nome do diretório onde há o arquivo que mantém os metadados dos objetos armazenados no disco. Este arquivo é utilizado para recriar o *cache* durante a inicialização do *Squid*. **“*Mbytes*”** especifica a quantidade de espaço em disco que deverá ser utilizada sob este diretório. **Nível-1** e **Nível-2** especificam o número de diretórios de primeiro e segundo nível, respectivamente, a serem criados, definindo na opção **Diretório**. Os valores padrão são 16 e 256, respectivamente. É possível ter vários diretórios para cache, inclusive em discos distintos.

- **cache_efective_user usuário, cache_efective_group grupo:** Estas opções servem para informar ao *Squid* com qual usuário e de grupo, respectivamente, ele deve ser executado.
- **cache_mem Mbytes:** Especifica em *Mbytes* o quanto de memória ram será utilizada para armazenar temporariamente os metadados de páginas web. Útil, pois o acesso às páginas web melhoram consideravelmente o desempenho.
- **visible_hostname nome_da_máquina:** Esta opção define o nome do computador que aparece em mensagens de erro de bloqueio de páginas ao usuário e em outras informações compartilhadas entre servidores cache.

De acordo com Zanoni (2007), o controle de acesso do *Squid* tem recursos suficientes para definir com precisão quais tipos de serviços podem ser acessados por quais máquinas e em quais horários. As regras da lista de controle de acesso (*Access Control List* ou simplismente ACLs) tem uma sintaxe bastante simples, e são incluídas no arquivo *squid.conf*. Cada ACL está associada a uma **regra de acesso** permitindo (*allow*) ou negando (*deny*), seguido de uma lista de nomes de elementos ACL:

acl proibir_cracker url_regex cracker

http_access deny proibir_cracker

De acordo com o exemplo acima, a primeira linha indica uma ACL criada para proibir a palavra “*cracker*”, ou seja, a segunda coluna “proibir_cracker” indica o nome

da ACL. O tipo de ACL utilizada foi a ***url_regex***, este tipo compara em toda URL baseando-se em expressões regulares, no caso do exemplo, a palavra "*cracker*". A segunda linha indica a regra bloqueando a ACL criada, ou seja, o ***http_access*** é responsável por permitir ou negar clientes http (*browsers*) acessarem a porta http.

2.2.2. SARG

De acordo com Morimoto (2008), o SARG é um interpretador de logs para o *Squid*. Sempre que executado, ele cria um conjunto de páginas web, divididas por dia, com uma lista de todas as páginas que foram acessadas e a partir de que máquina da rede veio cada acesso. Caso o *Squid* seja configurado para exigir autenticação, ele organiza os acessos com base nos *logins* dos usuários. Caso contrário, ele mostra os endereços IP das máquinas.

2.3. Gateway

De acordo com Morimoto (2005), um gateway pode ser um dispositivo de rede ou um computador que possui duas ou mais interfaces de rede que tem a função de interligar duas redes distintas, além de poder compartilhar a conexão com a Internet entre várias estações.

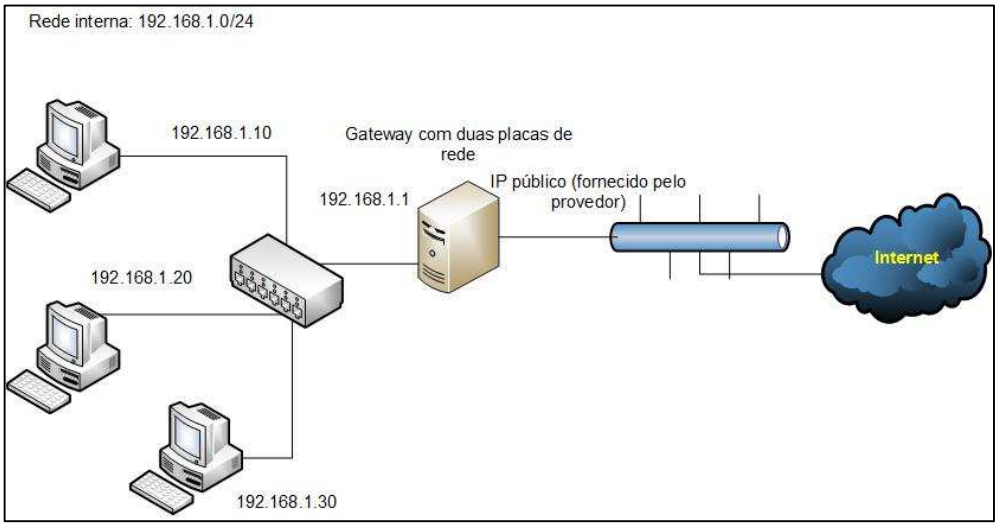

Figura 1: Exemplo de atuação de um Gateway
Fonte: Adaptado de MORIMOTO (2005)

De acordo com a Figura 1, a estação enviará ao gateway qualquer requisição de endereço que não faça parte da rede local. Uma rede com 3 computadores configurados com os endereços 192.168.1.10, 192.168.1.20 e 192.168.1.30, qualquer endereço fora do escopo 192.168.1.x será enviado ao *gateway*, que se

encarregará de acessá-lo na outra rede, ou na Internet e entregar o resultado à estação requisitante.

2.4. *Firewall*

De acordo com Nakamura e Geus (2007), um *firewall* é um ponto entre duas ou mais redes, que pode ser um componente ou um conjunto de componentes, por onde passa todo o tráfego da rede, tornando possível que o controle, a autenticação e os registros sejam realizados. A seguir, Nakamura e Geus (2007) exemplificam essa definição de firewall na Figura 2:

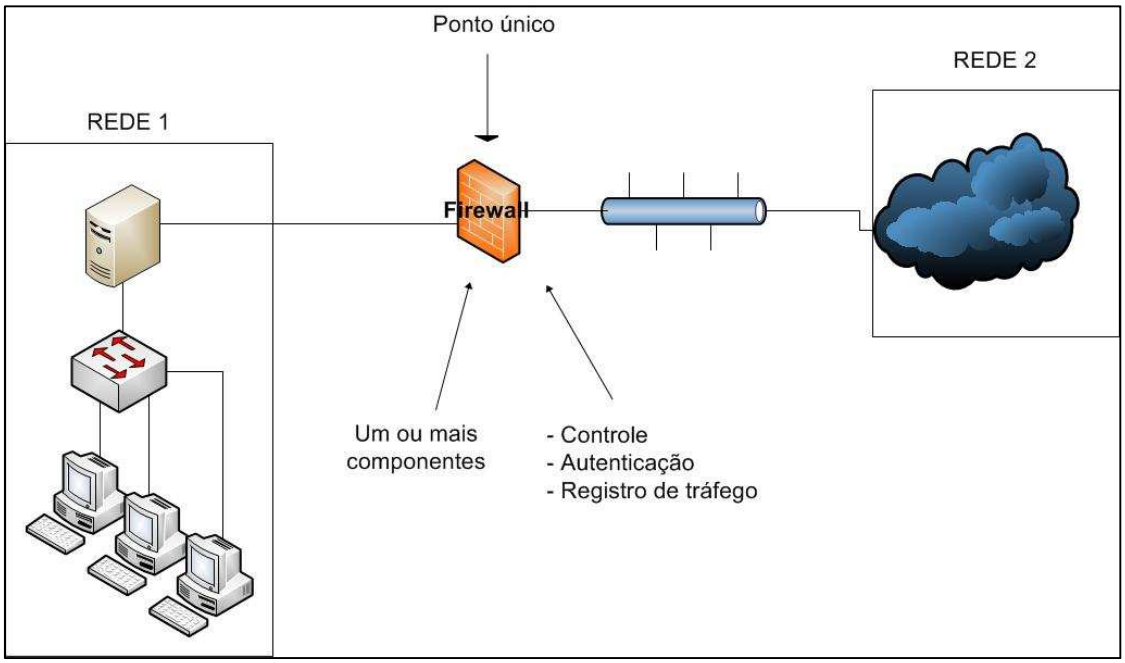

Figura 2: Definição de Firewall
Fonte: Adaptado de NAKAMURA e GEUS (2007)

2.4.1. Filtro de pacotes

De acordo com Nakamura e Geus (2007), o filtro de pacotes funciona na camada de rede e de transporte da pilha do protocolo TCP/IP, de modo que realiza as decisões de filtragem com base nas informações do cabeçalho dos pacotes tais como: endereço de origem e destino, a porta de origem e destino, e a direção das conexões.

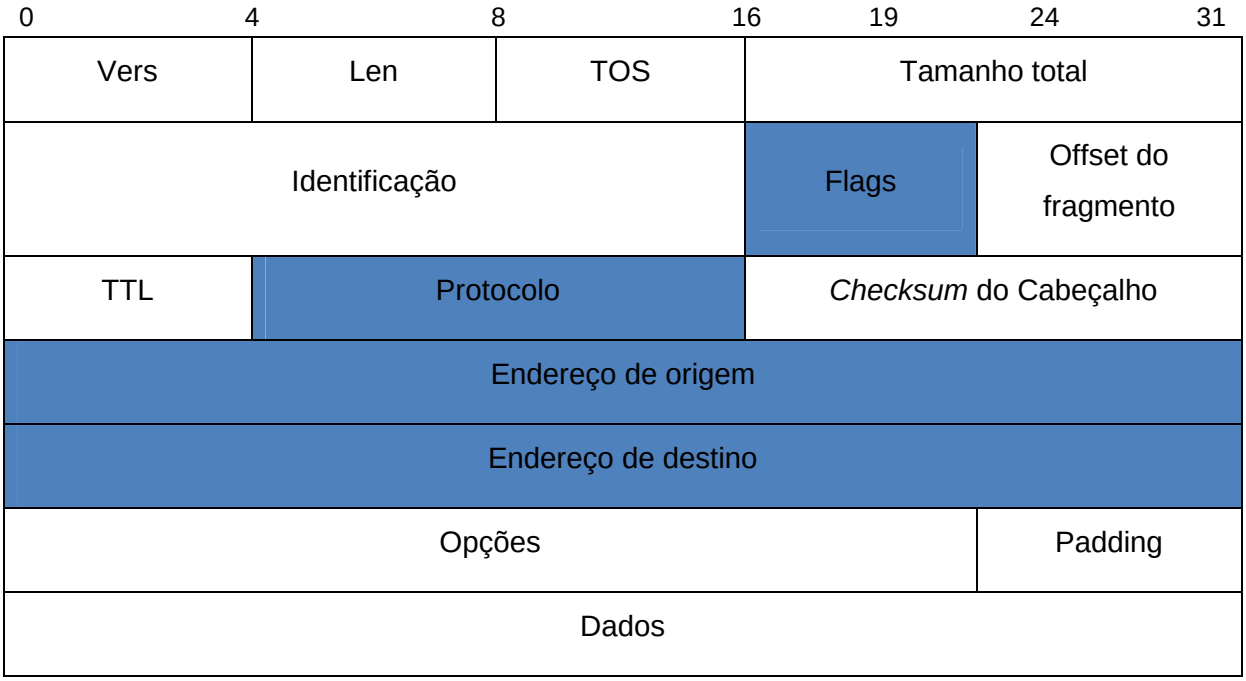

Figura 3: Campos do cabeçalho IP utilizados pelo firewall
Fonte: Adaptado de NAKAMURA e GEUS (2007)

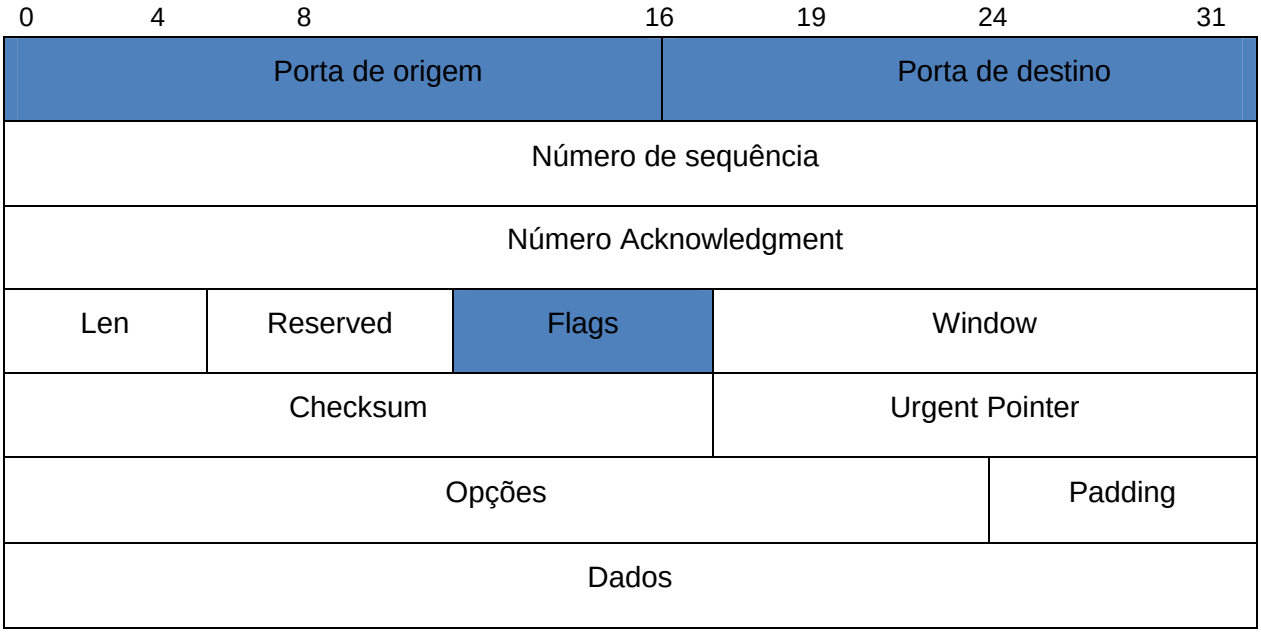

Figura 4: Campos do cabeçalho TCP utilizados pelo firewall
Fonte: Adaptado de NAKAMURA e GEUS (2007)

De acordo com as Figuras 3 e 4, é possível observar que os campos destacados são utilizados pelo *firewall*. Além disso, o sentido das conexões é verificado com base nas *flags* SYN, SYN-ACK e ACK do 3 – *Way Handshake* do protocolo TCP (*Protocolo de Controle de Transmissão*). Segundo Kurose (2005), o

TCP é orientado a conexões pois, antes que um host passe a enviar dados a outro, os dois necessitam primeiramente realizar um “aperto de mãos”, ou seja, devem enviar alguns segmentos um ao outro para estabelecer a transferência de dados. Este procedimento de estabelecimento de conexão é chamado de **apresentação de três vias** (*3-way handshake*). Basicamente, uma conexão TCP é estabelecida da seguinte maneira:

1. O cliente envia um pacote com *flag* SYN (início de conexão) ativa;
2. O servidor responde com um pacote com as *flags* SYN e ACK (reconhecimento);
3. O cliente responde com um pacote contendo a *flag* ACK.

Já as conexões dos protocolos UDP[5] e ICMP[6] feitas pelo firewall são um pouco diferentes da realizada nas conexões TCP. Com relação ao protocolo UDP, Nakamura e Geus (2007) ressalta que não é possível filtrar os pacotes com base no sentido das conexões, pois o mesmo não é orientado a conexões como o TCP (Figura 5). Já com protocolo ICMP, a filtragem é feita tomando como base nos tipos e códigos das mensagens conforme a Figura 6.

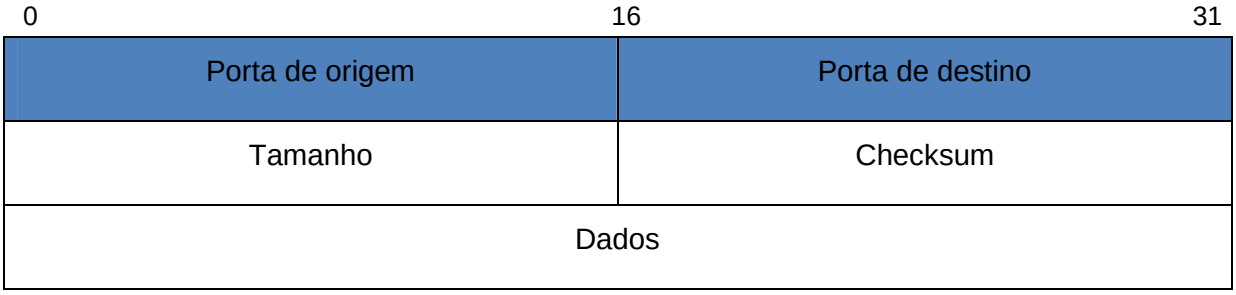

Figura 5: Campos do cabeçalho UDP utilizados pelo firewall
Fonte: Adaptado de NAKAMURA e GEUS

[5] O **User Datagram Protocol** (UDP) é um protocolo simples da camada de transporte. Ele permite que a aplicação escreva um datagrama encapsulado num pacote IPv4 ou IPv6, e então enviado ao destino. Mas não há qualquer tipo de garantia que o pacote irá chegar ao destino ou não.

[6] **ICMP**: é um utilitário de diagnóstico e de relatório de erros e é considerada uma parte necessária de qualquer implementação de IP. Noções básicas sobre o ICMP e saber o que pode gerar um tipo específico de ICMP são úteis no diagnóstico de problemas de rede.

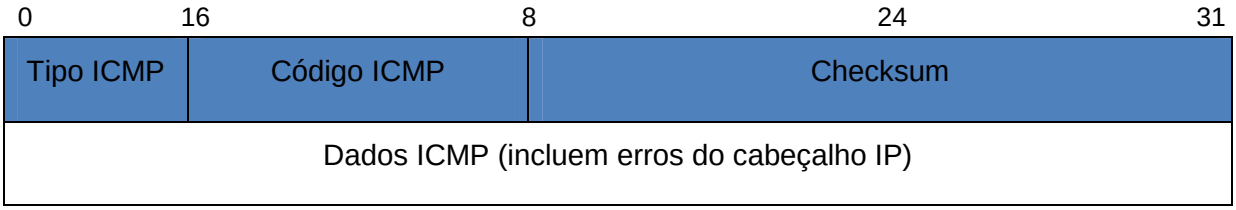

Figura 6: Campos do cabeçalho ICMP usados pelo firewall
Fonte: Adaptado de NAKAMURA e GEUS

Com isso, as regras dos filtros de pacotes são definidas com base nos endereços IP ou com os serviços (portas TCP/UDP relacionadas) permitidos ou negados, além de serem estáticas, de modo que esse tipo de firewall é conhecido como ***Static Packet Filtering***. Para os pacotes ICMP, a filtragem é realizada por código e por tipo de mensagem de controle ou erro.

Vantagens do filtro de pacotes:

- Alto desempenho da rede devido a sua simplicidade;
- É uma solução barata e flexível;
- Bom para o gerenciamento de tráfego;
- É transparente ao usuário;

Desvantagens:

- Permite conexão direta para os hosts internos de clientes externos;
- Complicado de gerenciar em ambientes complexos;
- É vulnerável a ataques de IP *spoofing* (falsificação de endereços IP privados), a menos que seja configurado para que isso seja evitado;
- Não oferece autenticação ao usuário;
- Dificuldade de filtrar serviços que utilizam portas dinâmicas;
- Deixa ‘brechas’ permanentes abertas no perímetro da rede.

Segundo Nakamura e Geus (2007), essas ‘brechas’ ocorrem porque as conexões que possuem regras específicas de permissão passam livremente pelo firewall e são estáticas. Com isso, abrem-se possibilidades de ataques que podem ser minimizados utilizando o filtro de pacotes baseado em estados. Nakamura e Geus (2007) exemplifica isto em um cenário onde um atacante faz uso de um cavalo de Troia utilizando softwares específicos e instala na máquina da vítima, que permite acesso remoto para obtenção de senhas digitadas e de visualização de telas.

Abaixo segue um exemplo de um conjunto de regras para permitir que usuários internos acessem a internet:

Regra	End. de origem: Porta de origem	End. de destino: Porta de destino	Ação tomada
1	IP da rede interna: porta alta	Qualquer endereço: 80 (HTTP)	Permitir
2	Qualquer endereço: 80 (HTTP)	IP da rede interna: porta alta	Permitir
3	Qualquer endereço: qualquer porta	Qualquer endereço: qualquer porta	Negar

Tabela 2: Regras de filtragem de um filtro de pacotes
Fonte: Adaptado de NAKAMURA e GEUS (2007)

Na Tabela 2, a primeira regra permite que usuários da rede interna iniciem a requisição de uma página web. Uma porta alta aleatória e acima de 1024 é usada pelo cliente para começar a requisição na porta 80 de um determinado servidor web. Caso a conexão tenha sucesso, a resposta da requisição é enviada para o cliente, passando pela regra 2 do firewall. A última regra (3) tem a função de negar qualquer outra tentativa de conexão e é recomendado que seja usada explicitamente.

Segundo Nakamura e Geus (2007), este tipo de firewall pode deixar com algumas falhas de segurança. Voltando ao cenário do atacante utilizando um cavalo de Troia, o canal aberto pela regra 2 da tabela acima pode ser explorado. Nesse caso, o atacante inicia uma conexão na porta 80 (HTTP) tendo como destino a porta alta aberta no usuário. Assim, as regras do filtro de pacotes nem sempre são capazes de proteger adequadamente todos os sistemas.

2.4.2. Filtro de pacotes baseado em estados

Segundo Nakamura e Geus (2007), os filtros de pacotes dinâmicos ou filtro de pacotes baseado em estados (*stateful packet filter*), tomam as decisões de filtragem com base em dois elementos:

1) As informações dos cabeçalhos dos pacotes de dados, assim como no filtro de pacotes;
2) Possui uma tabela de estados no *kernel* do sistema, que guarda os estados de todas as conexões.

Este firewall trabalha analisando somente o primeiro pacote de cada conexão, de acordo com as regras de filtragem definidas. A tabela de conexões que possui informações sobre o estado das mesmas ganha uma entrada a partir do momento que o pacote inicial é aceito. Após isso, os demais pacotes serão filtrados utilizando as informações da tabela de estados das conexões.

Tanto o filtro de pacotes quanto o filtro de pacotes baseado em estados trabalham na camada de rede do protocolo TCP, obtendo um bom desempenho. A diferença entre os dois é que o filtro de pacotes *stateful* monitora o estado das conexões a todo o momento, permitindo que a ação do firewall seja definida de acordo com o estado das conexões anteriores mantidas em sua tabela.

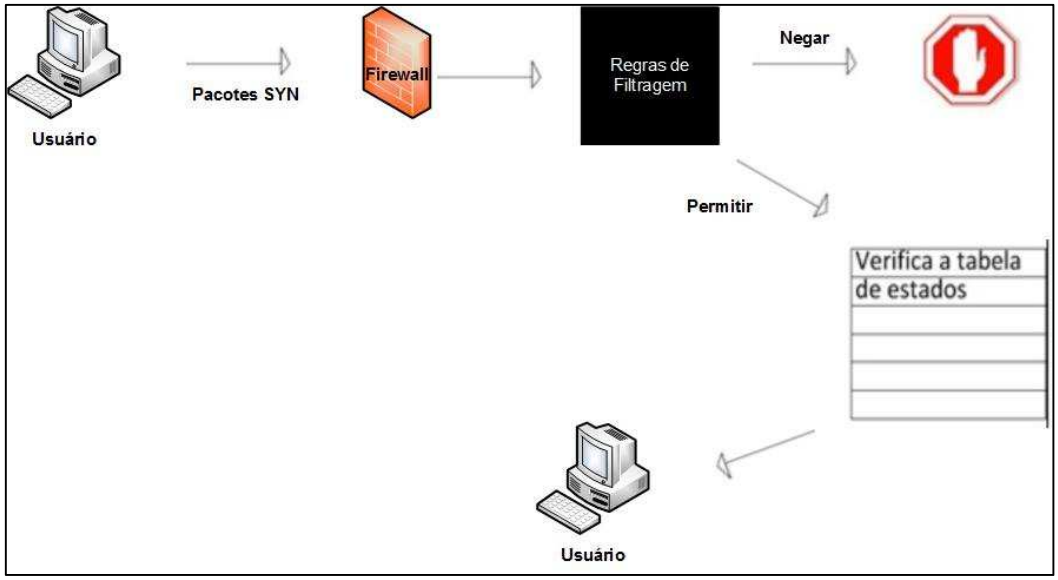

Figura 7: Filtro de pacotes baseado em estados operando na chegada de pacotes SYN
Fonte: Adaptado de NAKAMURA e GEUS (2007)

Conforme a Figura 7, um cliente inicia uma conexão TCP usando o pacote SYN, ele é comparado com as regras do firewall na ordem sequencial numérica da tabela de regras. Se o pacote passar por todas as regras existentes sem ser aceito, ele será descartado. Caso o pacote seja aceito, a sessão é inserida na tabela de estados do firewall, que está na memória do kernel. Para os demais pacotes, se a sessão estiver na tabela e o pacote fizer parte da mesma, ele será aceito. Porém, se

os pacotes não fizerem parte de nenhuma sessão presente na tabela de estados, eles serão descartados. A Figura 8 exemplifica isto:

Figura 8: Filtro de pacotes baseado em estados trabalhando na chegada dos demais pacotes
Fonte: Adaptado de NAKAMURA e GEUS (2007)

De acordo com Nakamura e Geus (2007), é justamente essa característica que faz com que este tipo de filtro de pacotes seja uma solução mais elegante na proteção de ataques. Com esse firewall, o conjunto de regras pode levar em conta somente os inícios das conexões, que usam a flag de início de conexão SYN.

Regra	End. de origem: porta de origem	End. destino: porta de destino	Ação
1	IP da rede interna: porta alta	Qualquer endereço: 80(HTTP)	Permitir
2	Qualquer endereço: qualquer porta	Qualquer endereço: qualquer porta	Negar

Tabela 3: Regras de filtragem do filtro de pacotes baseado em estados
Fonte: Adaptado de NAKAMURA e GEUS (2007)

De acordo com a Tabela 3, o conjunto de regras fica menos extenso e, consequentemente, mais fácil de administrar, minimizando as possibilidades de erros. Além disso, a regra 1 é usada para verificar se uma conexão pode ser iniciada, e a resposta a essa requisição é permitida com a verificação dos pacotes de acordo com a tabela de estados do firewall. Caso um atacante tente acessar alguma porta alta aberta ele não terá sucesso, pois a regra que permite essa conexão não existe.

Vantagens do filtro de pacotes baseado em estado:

- Aberturas apenas temporárias no perímetro da rede;

- Baixo overhead/alto desempenho da rede;
- Aceita quase todos os serviços;

Desvantagens

- Permite a conexão direta entre hosts internos a partir de redes externas;
- Não oferece autenticação do usuário.

2.4.3. NAT (Network Addess Translation)

Segundo Nakamura e Geus (2007), o NAT não foi desenvolvido para ser usado como um componente de segurança, mas sim para resolver o problema de escassez de endereços IP em redes de grande porte. Devido a isto, o NAT é responsável pela conversão de endereços IP privados (segundo a *Request For Comments*, RFC 1918) para endereços IP públicos e roteáveis, quando a rede externa é acessada. Com isso, vários usuários de uma mesma rede interna podem acessar a rede externa utilizando somente um endereço público fornecido pelo provedor.

Sob o ponto de vista da segurança, Nakamura e Geus (2007) afirmam que o NAT pode mascarar os endereços dos equipamentos da rede interna, e consequentemente sua topologia, dificultando os eventuais ataques externos.

2.5. Conceito de ataque de força bruta

Segundo o CERT.BR (2013), um ataque *brute force* ou força bruta consiste em descobrir, por tentativa e erro, um nome de usuário e senha, e assim, ter privilégios em executar processos e acessar sites, computadores e serviços. Qualquer computador, equipamento de rede ou serviço que esteja acessível pela Internet e que possua um nome de usuário e senha pode ser alvo deste tipo de ataque.

De acordo com o CERT.BR (2013), mesmo que um atacante não consiga descobrir a senha de uma determinada conta, o usuário poderá ter problemas ao acessá-la caso ela tenha sofrido um ataque de força bruta, pois muitos sistemas bloqueiam as contas quando várias tentativas de acesso sem sucesso são feitas. Apesar dos ataques de força bruta poderem ser realizados manualmente, eles são

realizados com o uso de ferramentas automatizadas que permitem tornar o ataque bem mais efetivo.

Segundo o CERT.BR (2013), as tentativas de adivinhação são baseadas em:

- Dicionários de diferentes idiomas;
- Listas de palavras comumente usadas, como nomes de times de futebol, celebridades do cinema;
- Substituições óbvias de caracteres, como trocar “a” por “@”;
- Sequências numéricas e de teclado, como “123456”, “qwert”;
- Informações pessoais, de conhecimento prévio do atacante ou coletadas na Internet em redes sociais como, nome, sobrenome, datas e números de documentos.

2.6. Trabalhos Relacionados

Com a popularização da Internet e a inclusão digital, as empresas atualmente passaram a utilizar com muito mais frequência os recursos via web. Não se limitando somente a este fato, novas ferramentas empresariais foram disponibilizadas para facilitar diversas atividades administrativas das organizações, tais como: Internet *Banking*, ferramentas para Recursos Humanos, Vídeo Conferência, análises do mercado cambial etc. Com o uso cada vez mais frequente da rede pública, fez com que surgisse uma excelente oportunidade para determinados atacantes/*hackers* habilidosos realizarem fraudes explorando vulnerabilidades em redes computacionais. Tais vulnerabilidades, podem incluir falhas de segurança em alguns protocolos onde a comunicação é total ou parcialmente aberta para a rede externa, práticas de engenharia social – onde o intuito é ludibriar o usuário leigo a fornecer senhas de acesso – ataques de negação de serviço em servidores, falsificação de endereço IP privado, falsificação de páginas web de instituições bancárias, etc.

Sob um olhar mais apurado diante dos atuais problemas de segurança da rede, alguns pesquisadores concentraram seus esforços para apresentar alguns recursos disponíveis com o intuito de evitar ataques vindos da rede externa. Tratando –se de um gateway onde sua localização lógica na rede se concentra no perímetro mais externo em um conceito de defesa em camadas, o principal recurso utilizado foi um dispositivo de *firewall*.

Rocha Junior (2010) apresentou um estudo sobre as características, conceitos e tipos de *firewall,* apresentando ainda alguns modelos de *firewall* corporativos e uma abordagem teórica e técnica sobre a implementação correta utilizando um sistema operacional Linux e *firewall iptables* nativo em praticamente todas as distribuições GNU Linux.

Em seu estudo de caso, Rocha Junior (2010) utilizou um cenário organizacional fictício que necessita proteger a rede interna e servidores. Dentre os serviços da rede estão: servidores de e-mail, banco de dados, web e DNS. A solução proposta pelo autor consistiu em acrescentar um servidor de firewall com três interfaces de rede: a primeira interface foi interligar os servidores críticos da rede onde estes estão em uma zona desmilitarizada (DMZ[7]) com outra classe de endereço IP privado. A segunda interface foi interligar a rede interna da organização, e a terceira interface para comunicação com a Internet.

Além disso, Rocha Junior (2010) mudou a política padrão do *iptables* para *DROP* (Isto significa que o *firewall* com uma política fechada seja necessário liberar somente os serviços utilizados no meio corporativo). A grande vantagem é a criação de regras mais “enxutas”, viabilizando o gerenciamento mais claro do *firewall*, agregando menos tempo de configuração, além de proporcionar muito mais segurança.

Furin e Machado Junior (2011), apresentaram as funcionalidades do *IPFIREWALL* (ipfw) - que é o *firewall* nativo do *FreeBSD*. Além disso, o principal objetivo foi realizar um comparativo entre as funcionalidades do *iptables* com as do ipfw. Segundo Furin e Machado Junior (2011), o IPFW conseguiu realizar todas as mesmas funcionalidades do *iptables* com eficiência, além de obter uma sintaxe mais legível das regras.

Barros (2011) apresentou um estudo de caso real em um órgão público, visando melhorias na segurança em um *gateway* de Internet (perímetro). O sistema operacional utilizado foi o FreeBSD com o *firewall* ipfw estático no *kernel*, além do acréscimo de um proxy *Squid* e o gerenciador gráfico de logs do proxy, o SARG.

[7] DMZ é uma rede que fica entre a rede interna – que deve ser protegida – e a rede externa. Essa segmentação faz com que, caso algum equipamento dessa rede desmilitarizada seja comprometido por alguma falha de segurança ou invasão, a rede interna continue intacta e segura.

Além disso, foram acrescentados recursos de controle de banda (*dummynet*), ajustes na pilha do protocolo TCP/IP e alterações das diretivas de alguns dispositivos de segurança do sistema.

Barros (2011) obteve resultados satisfatórios com relação ao consumo de banda larga, redução na ocorrência de vírus e ataques e melhorias no desempenho da navegação web. Entretanto, houve uma grande resistência de muitos usuários por tais mudanças, devido a alteração da política de segurança da informação da organização. Barros (2011) exemplifica isso com o caso de um usuário que acessava um programa de chat livremente dentro da rede, e que, com as mudanças realizadas, este tipo de serviço foi totalmente bloqueado por questões óbvias de segurança.

3. DISPOSITIVOS NATIVOS DE SEGURANÇA

O *FreeBSD* possui diversos dispositivos nativos de segurança que garantem o *grade level* mínimo CC/EAL4 (EAL5 está em processo de validação pelo DARPA em conjunto com o CL de *Cambridge*) de certificação de segurança (disponível em: http://www.freebsdbrasil.com.br/home.php?area=2&conteudo=5). Isto faz do *FreeBSD* um sistema operacional confiável, principalmente para prover serviços de Internet.

Neste tópico serão abordados alguns dos principais dispositivos nativos de segurança do sistema operacional *FreeBSD* para atuar como um *gateway* seguro.

3.1. O arquivo ttys

Segundo FreeBSD (2013), o *FreeBSD* possui múltiplos consoles virtuais que permitem executar vários programas de uma vez. Cada console tem seu próprio canal de saída, e o sistema operacional redireciona as entradas do console virtual do monitor, e também a saída padrão quando se alterna de um para o próximo console virtual. Além disso, combinações especiais de teclas foram reservadas pelo *FreeBSD* para controlar a alternância de consoles (pode usar Alt-F1, Alt-F2, até Alt-F8 para alternar entre diferentes consoles virtuais no *FreeBSD*).

Conforme estiver alternando entre um console e outro, o *FreeBSD* se responsabiliza por salvar e restaurar o conteúdo de cada tela. O resultado é uma “ilusão” de múltiplas telas e teclados virtuais disponíveis, de forma que torna possível digitar qualquer comando para o *FreeBSD* executar. Os programas que são iniciados em um console virtual não param de ser executados quando o console não está visível. Eles continuam a ser processados quando acontece alternância de um console virtual para outro (FREEBSD, 2013).

De acordo com FreeBSD (2013), O arquivo /etc/ttys lista todas as portas do sistema *FreeBSD* em que deseja permitir *logins*. A configuração padrão iniciará oito consoles virtuais. Contudo, esta configuração não é imutável e pode ser facilmente customizada de forma a possibilitar que o sistema seja iniciado com mais ou menos consoles virtuais. Cada linha descomentada neste arquivo (linhas que não começam com o caracter #) contém ajustes para um único terminal ou console virtual. A versão

padrão deste arquivo oferece com o FreeBSD nove consoles virtuais, e habilita oito deles:

```
# name getty           type    status     comments
#
console         none            unknown off secure
ttyv0   "/usr/libexec/getty Pc"  cons25 on secure
# Virtual terminals
ttyv1   "/usr/libexec/getty Pc"  cons25 on secure
ttyv2   "/usr/libexec/getty Pc"  cons25 on secure
ttyv3   "/usr/libexec/getty Pc"  cons25 on secure
ttyv4   "/usr/libexec/getty Pc"  cons25 on secure
ttyv5   "/usr/libexec/getty Pc"  cons25 on secure
ttyv6   "/usr/libexec/getty Pc"  cons25 on secure
ttyv7   "/usr/libexec/getty Pc"  cons25 on secure
ttyv8   "/usr/X11R6/bin/xdm -nodaemon"  xterm   off secure
```

A primeira linha do arquivo indica o console a ser utilizado quando em modo monousuário (modo de segurança). Já em modo multiusuário, o sistema utiliza terminais virtuais definidos de 0 a 7, sendo que o terminal 8 pode ser utilizado como terminal gráfico caso um servidor de ambiente gráfico esteja instalado no sistema operacional.

Dentre as opções no arquivo ttys, a que mais interessa no que diz respeito à segurança, é a última coluna (*secure*). Segundo FreeBSD (2013), se o modo está configurado como *secure*, isto significa que o servidor está alocado fisicamente em local seguro, onde apenas administradores e analistas em TI possuem acesso na sala. Entretanto, se determinado terminal estiver configurado como *insecure*, significa que o servidor pode estar em um ambiente físico inseguro.

No caso do console mono usuário, quando muda-se o parâmetro para *insecure*, faz com que toda vez que entrar em modo de segurança exija a senha de

root (super usuário). Fazendo isto, acrescenta um nível à mais de segurança fazendo com que somente pessoas autorizadas que possuem a senha de root, tenha acesso ao modo de segurança do sistema. Porém, caso ocorra a perda ou o simples esquecimento da senha de root, será impossível de recuperá-la, mesmo no modo monousuário (modo de segurança), pois o mesmo também irá exigir a senha de root para obter acesso (FREEBSD, 2013).

Outra função do parâmetro *insecure* é garantir a auditoria do servidor quando este é gerenciado por mais de um administrador. Ou seja, quando é configurado em um ou mais terminais virtuais no modo *insecure*, obrigatoriamente faz com que o administrador primeiramente faça logon com sua conta para depois, fazendo uso do comando ***su***, logar com a conta de super usuário. Segundo FreeBSD (2013), esta pequena mudança na configuração, garante que cada administrador com acesso ao sistema seja monitorado através dos logs, caso algum administrador faça algo de errado no servidor, como desconfigurar um serviço ou até deletar algum arquivo vital para o sistema.

Abaixo segue um exemplo de log no arquivo */var/log/messages* forçando determinado administrador a fazer o logon primeiramente com sua conta para somente depois logar como root:

Sep 19 19:22:40 wiki su: kratos to root on /dev/ttyv1

3.2. Kernel Securelevel

Segundo Lucas (2002), o *FreeBSD* possui níveis de segurança, que são configurações que mudam o comportamento básico do sistema para desaprovar determinadas ações. Ou seja, dependo do nível que é aplicado, o *Kernel* ligeiramente muda seu comportamento bloqueando determinadas ações como, não permitir a exclusão de arquivos vitais para o sistema, mudanças nas regras de firewall ou até mesmo impedir a gravação no disco rígido. Isto faz com que o atacante se frustre, pois de acordo com o nível de segurança aplicado, acaba por engessar suas ações maliciosas, evitando outros problemas para o servidor.

Segundo Lucas (2002), o *FreeBSD* oferece 5 níveis de segurança: -1, 0, 1, 2 e 3, sendo que o -1 é o nível mais baixo e o 3 é considerado alto:

- **Nível -1:** É o modo padrão, sinônimo de *Kernel securelevel* desligado. Não provê nenhuma segurança adicional;
- **Nível 0:** Este modo é utilizado somente no processo de boot. Ele garante que as permissões de arquivos sejam respeitadas;
- **Nível1:** a partir deste nível, as características de segurança começam a ser tornarem mais robustas:
    - As *chflags* podem ser definidas, mas não podem ser removidas;
    - Módulos de *Kernel* não podem ser carregados ou descarregados;
    - Programas não podem escrever diretamente na memória via /dev/mem (interface para memória física) e */dev/kmen* (interface para memória virtual). Além disso, o servidor gráfico X não funciona mais a partir deste ponto;
    - Nenhum dispositivo de armazenamento pode ser aberto para escrita direta a não ser por processos do sistema.
- **Nível 2:** Este nível de segurança incorpora todas as características descritas no Nível 1 e mais duas:
    - Discos não podem ser abertos para escrita direta, apenas pelo comando *mount*. Neste nível torna-se impossível a formatação de discos utilizando o comando *newfs* e aplicações que fazem acesso direto ao disco;
    - O relógio do sistema não pode ser alterado para mais de um segundo;
- **Nível 3:** Neste modo, todas as regras definidas nos modos anteriores estão ativas e acrescenta mais uma regra: regras de firewall não podem ser alteradas.

Para tornar esta operação permanente é necessário aumentar o nível de segurança direto no */etc/rc.conf* ou em */etc/sysctl.conf*:

/etc/rc.conf

- *kern_securelevel_enable="YES"*
- *kern_securelevel="2"*

/etc/sysctl.conf

- *kern.securelevel=2*

De acordo com Lucas (2002), é necessário utilizar com cautela o carregamento no boot, pois uma vez que os níveis de segurança sejam elevados antes do término do carregamento do *rc.conf*, a negação de privilégios pode atrapalhar o funcionamento de alguns programas. Além disso, a única maneira de poder desativar as *flags* e baixar o nível de segurança, é entrando em modo monousuário.

3.3. Marcas de arquivo – *chflags*

Todos os sistemas operacionais baseados em UNIX possuem permissões dando direitos de leitura, escrita e execução em arquivos e diretórios para o dono do arquivo, grupo ou todos os usuários. Segundo Lucas (2002), o *FreeBSD* estende este esquema de permissões para aumentar a segurança do sistema com o *file flags* (marcas de arquivo) utilizando o comando ***chflags.***

Sua sintaxe é:

chflags [opções] <flags> <arquivo>

A Tabela 4 apresenta as marcas de arquivo relacionados à segurança e suas funcionalidades:

NOME	FUNÇÃO	USUÁRIO HABILITADO
sappnd, sappend	*System-level append-only*. Quando utilizada esta opção, o arquivo não pode ser aberto para edição, apenas concatenado.	Super Usuário
Schg	*System-level immutable*. Com esta opção setada, o arquivo não pode ser editado, renomeado, apagado, ter suas permissões alteradas, etc.	Super usuário
sunlnk, sunlink	*System undeletable*. Usando esta opção, o arquivo não pode ser apagado sob hipótese alguma. Apenas poderá ser editado.	Super usuário
uappnd, uappend	*User append-only*. Tem o mesmo comportamento que sappnd, porém é apenas setada pelo dono do arquivo.	Dono do arquivo
uchg, uchange, uimmutable	*User immutable*. Tem as mesmas funções do schg, porém o usuário root pode apagar o arquivo.	Dono do arquivo
uunlnk, uunlink	User undeletable. Possui as mesmas funções do sunlnk, porém o usuário root pode anular esta *flag.*	Dono do arquivo

Tabela 4: Marcas de arquivos relacionados à segurança
Fonte: Adaptado de LUCAS (2002)

Para visualizar as *chflags* definidas basta executar o seguinte comando:

#ls –lo <arquivo>

*-rw-r-r-- 1 mwlucas mwlucas **uchg** 0 May 11 19:51 arquivo*

Cada marca de arquivo pode ser removida colocando-se o prefixo “no” em cada uma delas:

#chflags nouchg <arquivo>

Também podem ser removidas todas as *chflags* de um arquivo usando o seguinte comando:

#chflags 0 <arquivo>

3.4. IPFIREWALL

De acordo com Shandruk (2008), o IPFW (interface de comando do *IPFIREWALL*) é o *firewall* nativo mais popular do *FreeBSD* para implementar a filtragem de pacotes IP e controle de tráfego em uma rede de computadores. Além disso, o *IPFIREWALL* atua monitorando todas as conexões da rede, e a partir da versão 4.x do *FreeBSD*, o *IPFIREWALL* também faz a filtragem de pacotes baseado no estado (*stateful*) das conexões, o que o torna mais confiável. Este comportamento é sempre transparente para os usuários, até que um evento aguardado seja bloqueado.

3.4.1. O acionamento

Shandruk (2008) apresenta duas formas de acionar o IPFW: a primeira sugere adicionar algumas opções apropriadas no arquivo de configuração do *Kernel*, e compilar um novo *Kernel* para o sistema. A segunda, sugere utilizar o comando ***kldload*** para carregar dinamicamente no *Kernel* o módulo do IPFIREWALL. Segundo Shandruk (2008), os dois modos para acionar o IPFW funcionam bem, porém a compilação estática no *Kernel* proporciona, com pouca diferença, um melhor desempenho. Além disso, permite ainda que se adicione opções mais detalhadas de configuração no arquivo do *Kernel*, como por exemplo, a limitação de logs.

Abaixo o exemplo para carregar o IPFW de forma dinâmica:

kldload ipfw

Abaixo o exemplo para carregar o IPFW de forma estática no *Kernel*:

options IPFIREWALL

Todavia, para utilizar o IPFW de forma estática eficientemente é necessário acrescentar mais algumas opções no arquivo de configuração do novo *Kernel* que será compilado. Tomassoni (2012), aplica as opções necessárias objetivando o uso do *FreeBSD* como um *Gateway*. Abaixo segue as opções:

options IPFIREWALL_FORWARD

options IPFIREWALL_VERBOSE

options IPFIREEALL_VERBOSE_LIMIT=1000

options IPDIVERT

- *IPFIREWALL_FORWARD*: habilita o repasse de pacotes entre as interfaces de rede interna e externa;
- *IPFIREWALL_VERBOSE*: habilita o log do *firewall* no *syslog*[8];
- *IPFIREWALL_VERBOSE_LIMIT*=1000: limita o *log* em MB;
- *IPDIVERT*: habilita o uso do *socket divert* por padrão na porta 8668 para utilização do NAT.

3.4.2. O arquivo /etc/rc.firewall

De acordo com Shandruk (2008), sempre que a opção ***firewall_enable="YES"*** é adicionada no arquivo de inicialização *rc.conf* e o sistema é iniciado, o /etc/rc.firewall também é lido e executado, e os seguintes comandos são sempre adicionados ao IPFW:

${fwcmd} add 100 pass all from any to any via lo0

${fwcmd} add 200 deny all from any to any to 127.0.0.1/8

A variável *${fwcmd}* é definida anteriormente do *rc.firewall*, implicando quando o IPFW executará as regras de forma silenciosa (opção –q) ou não. Segundo

[8] Syslog é um daemon que fornece a ambientes Unix a opção comum de gravação de registros (logs) de programas. Estes registros são muito importantes para posteriores análises de problemas, como ocorrências de segurança do sistema.

Shandruk (2008), a primeira regra permite todo o tráfego proveniente da interface de *loopback*[9] (a lo0), e a segunda bloqueia todo o tráfego direcionado à rede *localhost* (127.0.0.0). A primeira regra é necessária para comunicação inter processos locais, e a segunda evita que qualquer pacote externo adentre o endereço de host local, que é o endereço de *loopback*, protegendo então o tráfego local. Numa política fechada de firewall, estas regras são necessárias para evitar problemas na inicialização de alguns serviços do sistema operacional.

3.4.3. Carregando o conjunto de regras

Segundo Shandruk (2008), existem duas formas de se carregar um conjunto de regras do firewall: a primeira, é usar uma das definições de firewall já existentes no arquivo padrão *rc.firewall*. A segunda, é criar um conjunto próprio de regras. Os autores do firewall nativo do *FreeBSD* recomendam o segundo caso, pois a customização das regras se adaptam da melhor maneira possível ao ambiente em questão, satisfazendo as necessidades reais do nível de segurança que se deseja atingir.

Para criar um conjunto próprio de regras, é necessário especificar um arquivo no *rc.conf* utilizando a seguinte opção:

firewall_type="/etc/firewall/regras.sh"

Feito isso, se torna possível customizar as regras no caminho do arquivo especificado, e sempre que o sistema for iniciado, estas regras serão carregadas no *IPFIREWALL*.

3.4.4. Comandos básicos do IPFIREWALL

Segundo Shandruk (2008), a sintaxe das regras são simples. Qualquer regra poderá ser adicionada pelo console com o comando **ipfw** (sbin/ipfw). A seguir, alguns exemplos de como listar as regras do *firewall*:

#ipfw list

---

[9] A interface de loopback é um tipo especial de interface que permite realizar conexões do próprio computador. Todos os computadores que utilizam o protocolo TCP/IP usam esta interface por várias razões, por exemplo, testar programas de rede sem interferir com ninguém da rede. Por convenção, o endereço IP utilizado é o 127.0.0.1.

#ipfw –t list

#ipfw –a list

#ipfw show

O primeiro exemplo é a forma mais simples de listar regras ativas. O segundo, mostra a data e a hora da última vez em que um pacote coincidiu com uma regra. Já os dois últimos exemplos vão apresentar informações mais detalhadas das regras, como por exemplo, o número de bytes que trafegaram por cada regra ativa.

A partir de agora, serão apresentados inicialmente alguns exemplos de regras *stateless*, ou seja, que não monitora o estado das conexões TCP:

add 100 allow all from any to any

Segundo Shandruk (2008), este é o exemplo mais simples de uma regra. Ou seja, “*all*” (todos) os pacotes vindos de “*any*” (qualquer) lugar para “*any*” qualquer destino são permitidos (*allow*). No *IPFIREWALL*, na maioria das vezes sempre que um pacote coincide com uma regra, o firewall termina de examinar as outras regras na lista para o pacote em questão. Isto significa que, a ordem como as regras são processadas no firewall são do tipo “*first match wins*”, onde a primeira constatação do pacote evita que as outras sejam lidas.

De forma geral, a sintaxe mais simples para o IPFW é:

<comando> [<número da regra>] <ação> <protocolo> from <origem> to <destino>

Segundo Shandruk (2008), os <comandos> mais importantes são "*add*" e "*delete*". O <número da regra> varia de 0 até 65535, e indica a ordem que serão processadas no *firewall*, portanto a última regra sempre define a política padrão do firewall no *Kernel*. A "<ação>" na sintaxe do IPFW pode ser uma das seguintes:

- “*allow*” | “*pass*” | “*permit*” | “*accept*” – sempre que um pacote combinar com essa ação, será permitido seu roteamento pelo firewall;
- “*deny*” | “*drop*” – sempre que um pacote combinar com essa ação, são silenciosamente bloqueados pelo firewall;
- “*count*” – todos os pacotes que combinarem com essa ação, o IPFIREWALL incrementa a contagem de pacotes. Exemplo:

add 1212 count all from 200.230.50.0/24 to 192.168.1.0/24

Esta regra conta quantos pacotes, dentre todos, estariam sendo enviados da rede 200.230.50.0/24 (origem) pra rede 192.158.1.0/24 (destino).

O <protocolo> na sintaxe básica de uso do IPFW, é o protocolo de comunicação que aquela regra combine. Definições de protocolos do tipo "*ip*" ou "*all*" são especificações gerais de todos os protocolos. Entretanto, os protocolos mais comuns utilizados são "*icmp*", "*udp*" e "*tcp*", mas a relação de protocolos com os quais o *IPFIREWALL* trabalha é enorme.

O endereço de destino e de origem de um pacote podem ser um nome, um endereço IP, um endereço de rede com máscara de rede e, ainda podem definir uma ou várias portas (se o protocolo for *TCP* ou *UDP*). De acordo com Shandruk (2008), usar nomes ou endereços IP é indiferente, basta atentar ao fato que a resolução de nomes via DNS pode mudar sem o conhecimento prévio.

add 100 deny all from 192.168.1.78 to any

A regra acima nega toda conexão da máquina 192.168.1.78 para qualquer outra estação. Sempre que um pacote coincidir com uma regra do firewall, o processamento pra aquele pacote termina, e o pacote é permitido ou negado, de acordo com a ação definida. Além disso, esse é um exemplo muito simples de um filtro baseado em estações (*host-based filtering*), que filtra de acordo com o destino ou origem do pacote (SHANDRUK, 2008).

Um *firewall* por filtragem de redes funciona de forma similar, distinguindo-se apenas a notação de redes, onde é necessário definir a máscara de subrede (*netmask*) ou ainda o *bitmask*:

add 2000 allow all from 192.168.0.0/16 to any

add 2100 deny all from any to 10.0.0.0:255.0.0.0

A primeira regra permite todo o tráfego de pacotes vindo da rede cujo conjunto de endereços IP começa em 192.168.0.0. A regra usa uma máscara (*bitmask*) pra indicar a abrangência do endereçamento da rede. Segundo Shandruk (2008), a máscara em bits especifica quantos bits do endereço de rede (192.168.0.0) devem permanecer o mesmo pra cada pacote. Nesse caso, os primeiros 16 bits dos 32 bits do endereço vão continuar os mesmos. Como os primeiros 16 bits são os primeiros

dois octetos do endereçamento da rede, então todos os endereços cuja origem seja a indicada nos dois primeiros octetos (ou nos 16 bits iniciais), nesse caso a rede 192.168, serão permitidos por essa regra.

A segunda regra tem uma função similar, utilizando as máscaras de rede. A máscara de rede (*netmask*) indica quantos bits do endereço de rede deve ser monitorado pelo regra do *firewall*. No exemplo acima, a regra tem a máscara de rede 255.0.0.0. Segundo Shandruk (2008), o primeiro octeto dessa regra é definido como “bits altos”, o que indica pro *firewall* que apenas os pacotes cujos primeiros 8 bits do endereço da rede devem ser filtrados. Como os primeiros 8 bits da rede fazem parte do endereço decimal igual a 10, então todos os pacotes cujo endereço de destino seja 10 no primeiro octeto (ou seja, toda a faixa de 10.0.0.0 até 10.255.255.255) vão combinar com essa regra, e então serão bloqueados.

Por fim, também é possível filtrar as conexões baseando-se nas portas de serviço de cada protocolo. Isso possibilita que bloqueie ou libere acesso a um determinado serviço específico ao invés de controlar o tráfego de todas as estações (SHANDRUK, 2008). A definição das portas em uma regra no IPFW é feita após a declaração do endereço de origem ou do endereço de destino, definindo a porta após o endereço. Uma faixa de portas pode ser especificada comumente com um hífen ou separadas por vírgula. Não se pode definir "*all*" como protocolo na regra que especifica portas, pois nem todos os protocolos trabalham com portas (SHANDRUK, 2008).

add 1000 allow tcp from any to 172.16.0.5 25

add 1100 allow tcp from any to 172.16.0.5 1021-1023

add 1200 allow tcp from any to 172.16.0.5 21,22,23

Na primeira regra (1000), todos os pacotes TCP cujo destino é a porta 25 da estação 172.16.0.5 são permitidos pelo *firewall*. Na segunda regra todas as conexões TCP cujo destino seja qualquer porta da faixa 1021 até 1023 da estação 172.16.0.5 também são permitidas. Na terceira regra, todos os pacotes TCP destinados às portas 21, 22 ou 23 na estação 172.16.0.5 é permitida pelo *IPFIREWALL*.

3.4.5. Sintaxe avançada das regras

Embora a sintaxe básica apresentada anteriormente seja o bastante pra cobrir a maioria dos cenários e necessidades simples de um *firewall*, ele se mostra solenemente curto pra muitas situações mais complexas. Então, Shandruk (2008) apresenta uma pequena expansão da sintaxe do IPFW:

<comando> [<número da regra>] <ação> <proto> from <origem> to <destino> [<interface-spec>] [<opções>]

Segundo Shandruk (2008), uma das mais importantes funcionalidades do IPFW é o controle de fluxo e de interface, ou seja, a possibilidade de criar regras que verifiquem os pacotes de acordo com uma interface em especial (aplicável quando se tem um sistema com múltiplas interfaces de rede). Essas regras podem identificar de onde esses pacotes estão vindo, e em que direção estão indo. Quando se deseja filtrar pacotes que estão apenas entrando ou saindo pela rede, as opções "*in*" e "*out*" podem ser utilizadas com precisão. Ambas opções correspondem ao campo "*interface-spec*" no modelo de sintaxe dado anteriormente, e normalmente, são definidas próximas ao final de cada regra, antecedendo qualquer possível opção extra. Dessa forma, quando se filtra todos os pacotes vindos de qualquer lugar e indo pra qualquer lugar, poderia definir da seguinte forma:

add 1000 allow all from any to any in

Para filtrar pacotes indo através de uma interface em particular, pode ser usada a opção literal "via", seguida do nome da interface. De acordo com Shandruk (2008), se estiver usando uma placa de rede 3Com 3c59x PCI, sua interface será "xl0". Para filtrar qualquer pacote, vindos especialmente dessa interface, com origem e destinos quaisquer, a seguinte sintaxe seria o suficiente:

add 1100 allow all from any to any in via xl0

Ou talvez, se tiver um sistema com múltiplas interfaces, e quiser filtrar todos os pacotes vindos de qualquer lugar e indo pra qualquer lugar, mas que esteja ao menos saindo de alguma interface, pode fazer da seguinte forma:

add 1200 allow all from any to any out via any

Segundo Shandruk (2008), pacotes ICMP, TCP e IP são apresentados em vários formatos distintos. Esses tipos de pacotes são definidos por várias *flags*

(opções de identificação). É possível filtrar cada um desses tipos usando uma das seguintes opções do *IPFIREWALL* ao final de cada regra:

***icmptypes* (tipos de ICMP)**

Essa opção filtra o pacote ICMP do tipo definido, e inverte a opção se uma '!' (exclamação) for definida antes do tipo, ou seja, todos os pacotes que não forem desse tipo serão combinados. Segundo Shandruk (2008), existe atualmente 15 tipos diferentes de pacotes ICMP que podem ser filtrados; cada um é definido por um número. É possível ainda especificar faixas de *'icmptypes'* utilizando hífen ou separando-os por vírgula.

De acordo com Shandruk (2008), usar filtros de pacotes ICMP é muito útil, especialmente pra controlar o comando *ping*[10], ou seja, é possível permitir que qualquer cliente dentro da rede ‘pingue’ qualquer estação pra fora da rede, enquanto bloqueia qualquer estação externa de ‘pingar’ no *gateway*, ou qualquer outro cliente interno.

tcpflags, setup e established

"*tcpflags <flag>*" - Essa opção filtra qualquer pacote TCP cujo cabeçalho contenha alguma das *flags* (opções) identificadas. Uma definição inversa também pode ser utilizada se colocar uma '!' (exclamação) antes da <*flag*>, dessa forma filtrando todos os pacotes TCP que não possuam a *flag* em questão.

Segundo Shandruk (2008), a *flag* do tipo **SYN** é a mais interessante, visto que ela é enviada quando uma conexão TCP é iniciada. Existe inclusive uma opção separada do IPFW dedicada para definir pacotes TCP cujo cabeçalho tenha a *flag* SYN ajustada. Essa opção se chama "***setup***". Qualquer regra contendo essa opção irá filtrar todos os pacotes TCP cujo cabeçalho indique a *flag* SYN ajustada.

add deny tcp from any to any in tcpflags syn

OU:

add deny tcp from any to any in setup

---

[10] Ping ou latência como podemos chamar, é um utilitário que usa o protocolo ICMP para testar a conectividade entre equipamentos.

Em cada uma dessas regras, a mesma ação é tomada: todos os pacotes TCP SYN vindos de qualquer (*any*) origem para qualquer (*any*) destino serão negados.

"*established*" – Esta opção identifica quando uma conexão já está estabelecida. Pela importância de facilitar o controle de conexões TCP, Shandruk (2008) afirma que as opções "*established*" e "*setup*" foram disponibilizadas dessa forma oferecendo facilidade na criação de regras. Utilizando estas opções é possível ter inicialmente um controle de conexões TCP mais simples. Além disso, essa é a base de implementações "*stateful*" (filtro baseado no estado das conexões) de um *firewall*.

3.4.6. Configurações *Stateful* avançadas

Segundo Shandruk (2008), as configurações de firewall *stateful* usando apenas *setup* e *established* são muito limitadas. Exatamente por permitir controle de conexões *stateful* apenas sobre o protocolo TCP, essa filtragem é a mais simples existente. Desde a versão 4.0 do *FreeBSD*, o *IPFIREWALL* adotou características *stateful* baseada em ótimos argumentos; agora pode-se controlar conexões TCP, UDP e ICMP de forma *stateful*, além de outros tipos de pacotes, usando **regras dinâmicas**.

As regras dinâmicas, como seu próprio nome sugere, são regras dinamicamente criadas para conexões independentes. Cada regra dinâmica depois de um certo período de tempo sem serem utilizadas são descarregadas. O tempo que uma conexão TCP leva para ser terminada pode ser controlada por inúmeras variáveis, ou seja, de certa forma um conjunto de regras monitora não somente o início de uma conexão, mas também quando essa conexão é terminada, e ajusta suas ações de acordo com a configuração utilizada.

De acordo com Shandruk (2008), uma opção e um comando são utilizados pra controlar esse comportamento *stateful* avançado:

- *keep-state*: quando um pacote é combinado com uma regra que tenha essa opção, então uma regra dinâmica é iniciada;
- *check-state*: esse comando define que a verificação das regras pelo firewall irá primeiro checar as regras dinâmicas criadas no início de uma conexão TCP. Caso não exista uma regra com esse comando em

> todo o conjunto de regras do IPFW, então as regras dinâmicas serão verificadas no momento em que a opção “*keep-state*” for definida. Se uma regra com esse comando combinar com um pacote, a verificação das regras é terminada, caso contrário, a verificação continua.

A seguir, um exemplo de regra dinâmica:

add 1000 check-state

add 2000 allow tcp from any to any 22 in setup keep-state

De acordo com o exemplo acima, a primeira regra faz com que o *firewall* verifique as regras dinâmicas. Se o pacote não pertence a nenhuma conexão já estabelecida, então a verificação continua na regra 2000, onde, se o pacote for do tipo TCP SYN entrando pela porta 22, então a função "*keep-state*" ordena a criação de uma regra dinâmica, que será sempre verificada na constatação da regra 1000.

Com essa nova abordagem, utilizando a opção "*keep-state*" e o comando "*check-state*" proporciona algumas vantagens sobre as outras configurações de firewall. Antes, a opção "*established*" permitia a ocorrência de qualquer pacote TCP vindo de uma conexão previamente estabelecida, ainda que esse pacote fosse ‘*spoofado’* e não fosse um pacote legítimo dessa conexão TCP. De acordo com Shandruk (2008), a expressão "*spoof*" define um tipo de pacote que traz consigo informações de origem manipulada, ou seja, o pacote não legítimo se faz passar por um pacote que na verdade não é ele. É uma técnica não muito simples e que pode ser evitada de várias formas, uma delas é a verificação pelo *firewall*. Cada regra dinâmica é criada para uma conexão específica entre dois hosts e suas respectivas portas, ou seja, um pacote TCP spoofado poderia manipular o endereço de destino e de origem, mas não manipularia a porta onde a conexão foi efetivada, e onde a conexão TCP legítima está sendo mantida. Dessa forma a regra 1000 (*check-state*) falha, não permitindo o roteamento do pacote, e posteriormente a regra seguinte (2000) também falharia, a não ser que o pacote fosse do tipo TCP SYN, e dessa forma o pacote é negado pra regra final da política padrão fechada do *IPFIREWALL*.

4. ESTUDO DE CASO

O referido estudo objetiva o leitor a uma compreensão mais clara dos fundamentos teóricos apresentados anteriormente através de experiências realizadas no decorrer do desenvolvimento deste trabalho. Estes experimentos são de suma importância para mostrar a confiabilidade e eficiência do sistema operacional *FreeBSD* atuando como um Gateway de perímetro, além do uso de outras ferramentas que colaboram para a efetiva segurança.

Neste capítulo serão apresentados testes de segurança utilizando o arquivo ttys, as *chflags*, o nível de segurança do *Kernel*, além do uso do proxy para segurança interna da rede bem como o uso do SARG para visualização gráfica dos logs de acesso do proxy *Squid*. Além disso, serão apresentadas as regras do *IPFIREWALL* e suas funções utilizadas para proteção de ameaças externas. Por fim, também será apresentado ao leitor um teste de invasão utilizando o conceito de força bruta ao protocolo SSH, bem como a prevenção deste ataque utilizando o aplicativo *Fail2ban* (disponível no *FreeBSD* para a instalação) em conjunto com o *IPFIREWALL*.

4.1. Hardwares e softwares utilizados

Os hardwares utilizados para o experimento foram:

- Notebook *Dell*® *Inspiron* 14R;
    - Processador *Intel*® Core i5-3337U 1.80GHz;
    - Memória *Kingston*® DDR3 6GB;
    - HD *Seagate*® 1TB;
    - Placa de vídeo *Nvidia*® *Geforce* GT730M 2GB 128bits.
- Roteador *Cisco*® *Linksys* WRT160N 300Mbps;
- Banda larga 10Mbps (valor nominal);

Softwares utilizados:

- *Oracle*® *VirtualBox* versão 4.2.16;
- *Kali Linux* versão 1.0.7;
- *Microsoft*® Windows 8 Pro;
- *FreeBSD*® versão 9.2;

- *Squid Cache* versão 3.4.1;
- *Apache* versão 2.2.25;
- *Fail2ban;*
- *Sarg* (relatórios de acesso do proxy).

4.2. Características da rede

Todo cenário montado foi realizado em máquinas virtuais (*VM – Virtual Machine*) utilizando o *VirtualBox*. A Figura 9 e a Tabela 5 apresentam o diagrama e os IPs respectivamente:

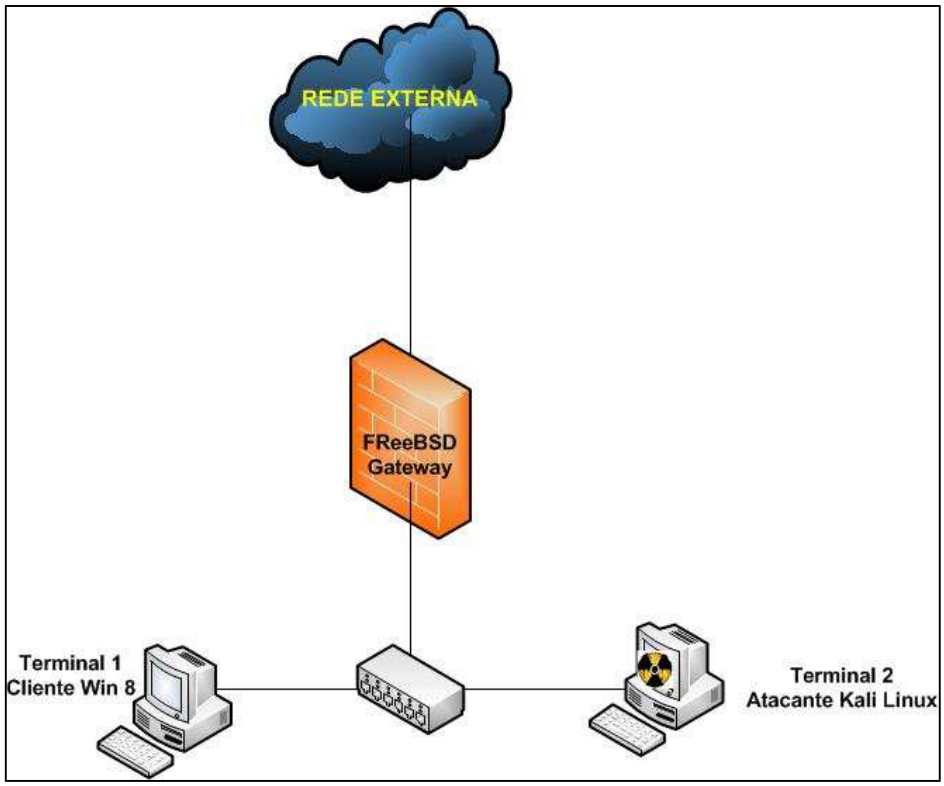

Figura 9: Diagrama lógico da rede no VirtualBox
Fonte: Próprio autor (2014)

TIPO	HOST	IP
REDE EXTERNA	FreeBSD	201.62.123.75 (IP real) / 10.0.3.15 (IP virtual da VM)
REDE INTERNA	Terminal 1	192.168.1.210/24
	Terminal 2	192.168.1.220/24
	FreeBSD	192.168.1.230/24

Tabela 5: Endereço IP de cada host
Fonte: Próprio autor (2014)

4.3. O arquivo de inicialização

A Figura 10 mostra a configuração do arquivo /etc/rc.conf para carregar os módulos utilizados, bem como as interfaces de rede.

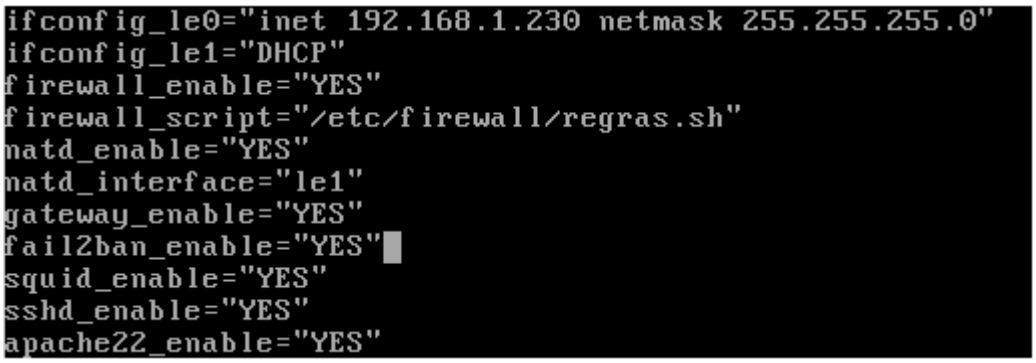

Figura 10: Configuração do rc.conf
Fonte: Próprio autor (2014)

4.4. Segurança aplicada no arquivo ttys

Neste arquivo foram aplicados, tanto no console monousuário quanto no multiusuário, o parâmetro “*insecure*”. Isto significa que, o acesso no console monousuário será exigida a senha de root, e no console multiusuário será proibido o acesso direto à conta root, obrigando o administrador a realizar o login em uma conta previamente cadastrada e sem privilégios antes de se tornar super usuário (*root*) no sistema. As figuras 11, 12 e 13 mostram os efeitos dessa mudança:

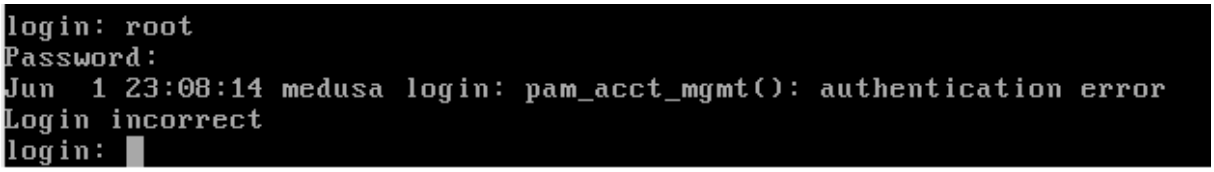

Figura 11: Falha no login de root do console multiusuário
Fonte: Próprio autor (2014)

Figura 12: Acesso root após exigência de login em uma conta sem privilégios
Fonte: Próprio autor (2014)

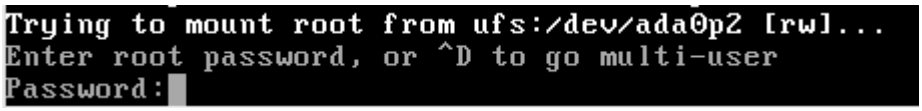

Figura 13: Exigência da senha de root no console monousuário
Fonte: Próprio autor (2014)

4.4. Aplicação das marcas de arquivo (*chflags*)

A *chflag* **schg** no qual fornece a imutabilidade, foi aplicada nos arquivos considerados importantes para o sistema, como os binários, arquivo de configuração do proxy, arquivo com a lista de regras do firewall e o arquivo de inicialização do sistema. Isto evita a remoção acidental ou intencional por algum usuário sem autorização e/ou por invasões externas. A seguir, a Figura 14 mostra a aplicação de uma *chflag*. Já as figuras 15 e 16 mostram as consequências disso:

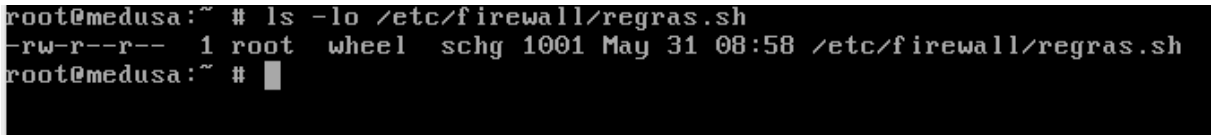

Figura 14: Aplicação da chflag schg no arquivo do ipfw
Fonte: Próprio autor (2014)

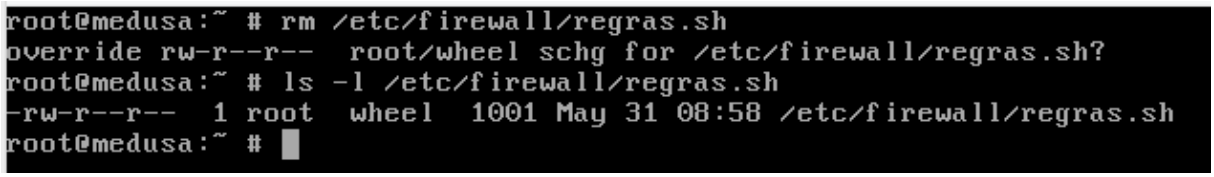

Figura 15: Falha ao deletar o arquivo
Fonte: Próprio autor (2014)

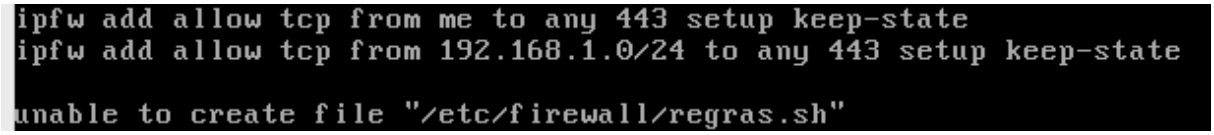

Figura 16: Falha ao editar o arquivo
Fonte: Próprio autor (2014)

De acordo com HANDBOOK (2013), a única forma de excluir está *chflag* é iniciando o sistema em modo monousuário.

Além disso, esta *chflag* se torna muito útil caso ocorra algum ataque externo. Pois, mesmo que um atacante ganhe o acesso privilegiado ao sistema, ele não conseguirá alterar e nem apagar os principais arquivos do sistema.

4.4. Alterando o nível de segurança do *Kernel*

A alteração do nível de segurança do Kernel fazendo uso da sysctl "*kern.securelevel*" foi utilizada neste experimento apenas para fins de teste. Em um servidor corporativo, o administrador necessita usar esta configuração com cautela, pois acaba "engessando" demais o sistema tornando-o impossível de trabalhar, além de possíveis frustrações dos usuários da empresa. Todavia, esta sysctl se torna uma grande aliada em planos de contingência, como por exemplo, uma possível ameaça tentando invadir o sistema. Ou seja, basta apenas uma configuração e automaticamente o sistema toma medidas de segurança bloqueando diversos serviços e arquivos.

O nível 2 na sysctl *kern.securelevel* foi utilizado neste experimento, como visto no capítulo anterior. A seguir, as figuras 17, 18 e 19 apresentam alguns resultados após a mudança do referido nível de segurança do sistema:

Figura 17: Falha na montagem automática do sistema de arquivos durante o boot
Fonte: Próprio autor (2014)

Figura 18: Falha no carregamento de módulos do Kernel
Fonte: Próprio autor (2014)

Figura 19: chflags podem ser criadas, mas jamais removidas
Fonte: Próprio autor (2014)

4.5. Uso do proxy *Squid*

O *Squid* foi configurado no modo de autenticação padrão utilizando o *ncsa_auth* do *Apache*. De modo geral, cada usuário precisará se autenticar no navegador web através do proxy, antes de utilizar a *Internet*. Com isso, cada usuário cadastrado terá no arquivo de configuração do proxy, uma lista de *sites* proibidos e permitidos, dependendo do privilégio de cada um. Controlando o acesso *web*, acrescenta-se mais um nível de segurança no experimento. Pois, toda requisição *web* dos usuários serão controladas obrigatoriamente pelo proxy antes de sair para a rede externa. Segundo Nakamura e Geus (2007), os proxies atuam na camada de aplicação, o que oferece mais controle sobre a interação entre o cliente e o servidor externo.

Além disso, o *Squid* foi configurado para realizar cache de páginas *web* (visto na seção 2.2). Isto ajuda a obter economia de banda larga e agilidade nas respostas de requisições.

As figuras 20 e 21, mostram trechos importantes do arquivo de configuração do *Squid*, o *squid.conf*:

```
auth_param basic program /usr/local/libexec/squid/ncsa_auth /usr/contas/usuarios
auth_param basic children 5
auth_param basic credentialsttl 1 hour
auth_param basic realm Digite seu Login e Senha
auth_param basic casesensitive off
```

Figura 20: Trecho responsável pela autenticação
Fonte: Próprio autor (2014)

- ***auth_param basic program***: indica o caminho do módulo ncsa_auth que irá fazer a autenticação e o caminho do arquivo onde ficarão as senhas dos usuários;
- ***auth_param basic children 5***: número de processos filho para autenticação. O valor 5 é o padrão na maioria dos casos. Apenas muda-se este valor quando diversos computadores enviam requisições de autenticação;
- ***auth_param basic credentialsttl 1 hour***: define o tempo de logon no proxy;
- ***auth_param basic realm***: define o texto que irá aparecer na caixa de login;
- ***auth_param basic casesensitive***: ativa ou desativa a verificação de letras maiúsculas e minúsculas.

```
cache_mem 200 MB
maximum_object_size_in_memory 32 KB
maximum_object_size 1024 MB
minimum_object_size 0 KB
cache_swap_low 90
cache_swap_high 95
cache_dir ufs /var/squid/cache 2000 32 512
access_log /var/squid/logs/access.log
```

Figura 21: Trecho responsável pelo cache de páginas web
Fonte: Próprio autor (2014)

- ***cache_mem***: define quanto de memória ram será utilizado para o cache;
- ***maximum_object_size_in_memory***: define o tamanho máximo de objetos na memória ram;
- ***maximum_object_size***: define o tamanho máximo de objetos no disco;
- ***minimum_object_size***: tamanho mínimo dos objetos em cache;
- ***cache_swap_low 90/cache_swap_high 95***: a partir do momento em que o *cache* atingir 95%, serão eliminados objetos mais antigos até que a porcentagem retorne abaixo de 90%;
- ***cache_dir ufs /var/squid/cache 2000 32 512***: define o diretório e o número de diretórios onde os objetos de cache estarão. O número 2000 representa o espaço em disco em MB que o *cache* do *Squid* poderá ocupar. O 32 representa o número de diretórios de cache. O número 512 representa a quantidade de subdiretórios de *cache*;
- ***access_log***: define o caminho dos *logs* de acesso dos usuários.

```
root@judite:/var/squid/cache # ls
00              07              0E              15              1C
01              08              0F              16              1D
02              09              10              17              1E
03              0A              11              18              1F
04              0B              12              19              swap.state
05              0C              13              1A
06              0D              14              1B
```

Figura 22: Cache sendo realizado
Fonte: Próprio autor (2014)

4.5.1. ACLs (Listas de Controle de Acesso) definidas para o bloqueio de acordo com a conta do usuário

De acordo com Gonçalves (2005), as ACL's são regras de controle de acesso inseridas no arquivo de configuração do *Squid* para atender às necessidades de

autenticação de usuários, bloqueio e permissão de *sites*, definição do uso da *web* por horários, definição regras de acesso por setor ou usuário etc.

As ACL's definidas neste experimento contempla o bloqueio e liberação de determinados sites de acordo com o usuário:

```
acl lib_users proxy_auth "/usr/local/etc/squid/filtros/usuarios_livres"
acl control_ti proxy_auth "/usr/local/etc/squid/filtros/ti"
acl control_adm proxy_auth "/usr/local/etc/squid/filtros/adm"
acl bloqsite_ti dstdomain -i "/usr/local/etc/squid/filtros/bloq_ti"
acl bloqsite_adm dstdomain -i "/usr/local/etc/squid/filtros/bloq_adm"
```

Figura 23: Trecho das ACLs definidas
Fonte: Próprio autor (2014)

De acordo com a Figura 23, a primeira ACL estão cadastrados os usuários com acesso total à *Internet*. A Segunda e terceira ACL, estão configurados os usuários com acesso mais controlado.

As duas últimas ACL's estão cadastrados os domínios que serão bloqueados. Detalhe ao parâmetro "-i" onde desativa a verificação de maiúsculas e minúsculas de cada palavra no arquivo.

Entretanto, para funcionar definitivamente as ACL's, é necessário acrescentar o comando ***http_access*** para então liberar ou bloquear. A seguir, a sintaxe do http_access e um pequeno trecho do *squid.conf* na Figura 24:

http_access <ação> nome_da_acl

```
http_access allow lib_users
http_access deny bloqsite_ti control_ti
http_access deny bloqsite_adm control_adm
```

Figura 24: Trecho do comando http_access
Fonte: Próprio autor (2014)

Na Figura 25 segue o teste realizado com o usuário fictício George do departamento de TI, onde ele não tem acesso apenas ao *site* facebook.com:

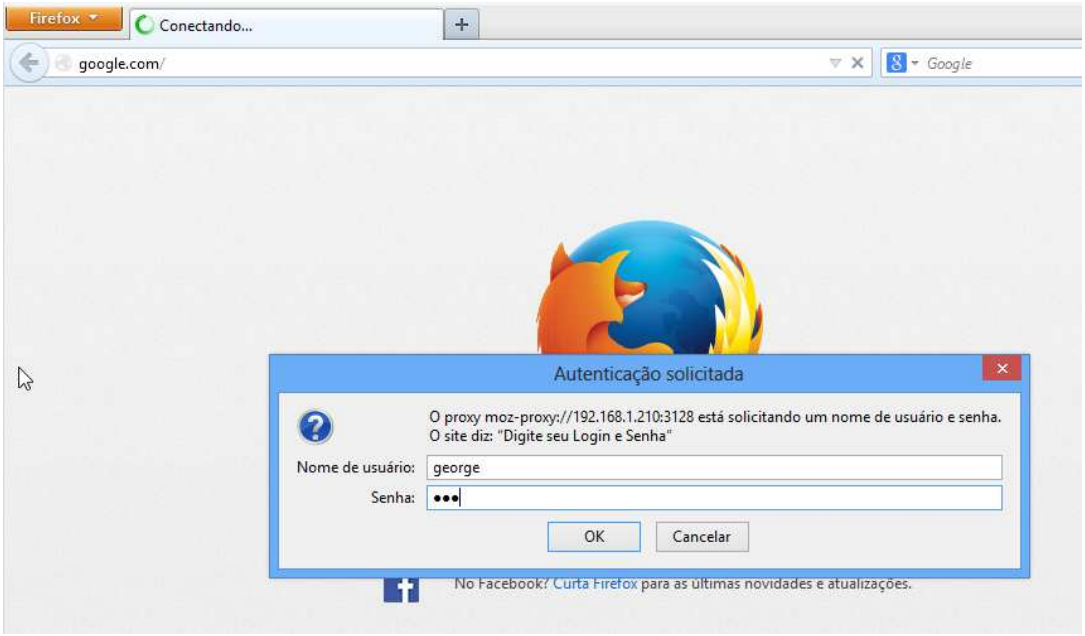

Figura 25:Autenticação do usuário fictício
Fonte: Próprio autor (2014)

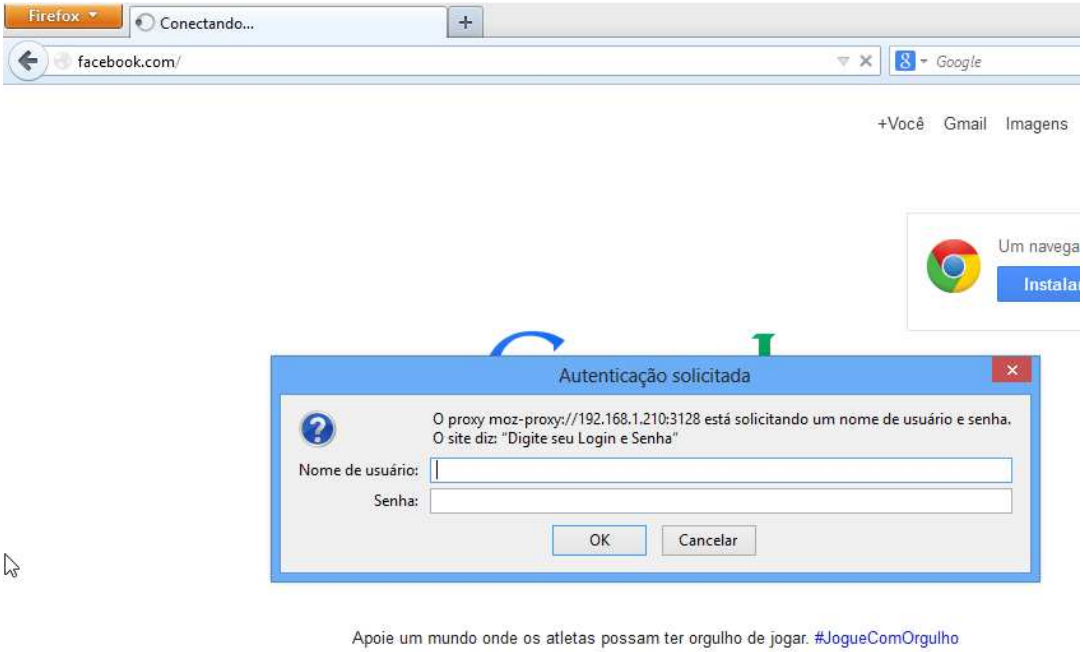

Figura 26: Erro ao tentar acessar o Facebook
Fonte: Próprio autor (2014)

O resultado na Figura 26 é que toda vez que o usuário tentar acessar o site facebook.com, abrirá a caixa de autenticação obrigatória. Porém, quando o usuário realizar o *login* novamente, mesmo assim não conseguirá acessar e abrirá novamente a caixa de autenticação até que seja feito o *login* com um nome de usuário e senha válidos para o acesso.

Figura 27: Usuário com privilégios para acessar o facebook.com
Fonte: Próprio autor (2014)

A Figura 27 mostra o acesso ao *Facebook* após ter utilizado um usuário com privilégios de acesso definidos no arquivo *squid.conf*,

4.6. Uso do SARG para *logs* gráficos do *Squid*

O SARG foi utilizado para visualizar graficamente através do navegador, os *logs* de acesso *web* de cada usuário. Abaixo segue os parâmetros utilizados no arquivo de configuração do SARG:

- ***Access_log /var/squid/logs/access.log***: define o local onde estão os logs do Squid;
- ***Temporary_dir /usr/temp***: define o diretório temporário onde o SARG vai gerar o relatório (não é o diretório de saída do relatório para ser visualizado no browser);
- ***Output_dir /usr/local/www/apache22/data/sarg****:* define o diretório de saída do relatório para ser visualizado no browser.

Além disso, foi configurado o acesso restrito ao SARG onde somente usuários cadastrados poderão ter acesso aos logs. Isso foi feito através do arquivo de configuração do *Apache*, o *httpd.conf*.

Abaixo segue o comando para gerar o relatório manualmente no SARG:

/usr/local/bin/sarg –f /usr/local/etc/sarg/sarg.conf

Após gerar o relatório, basta digitar no navegador o ip/sarg para visualizar os logs. As figuras 28 e 29 exemplificam isso:

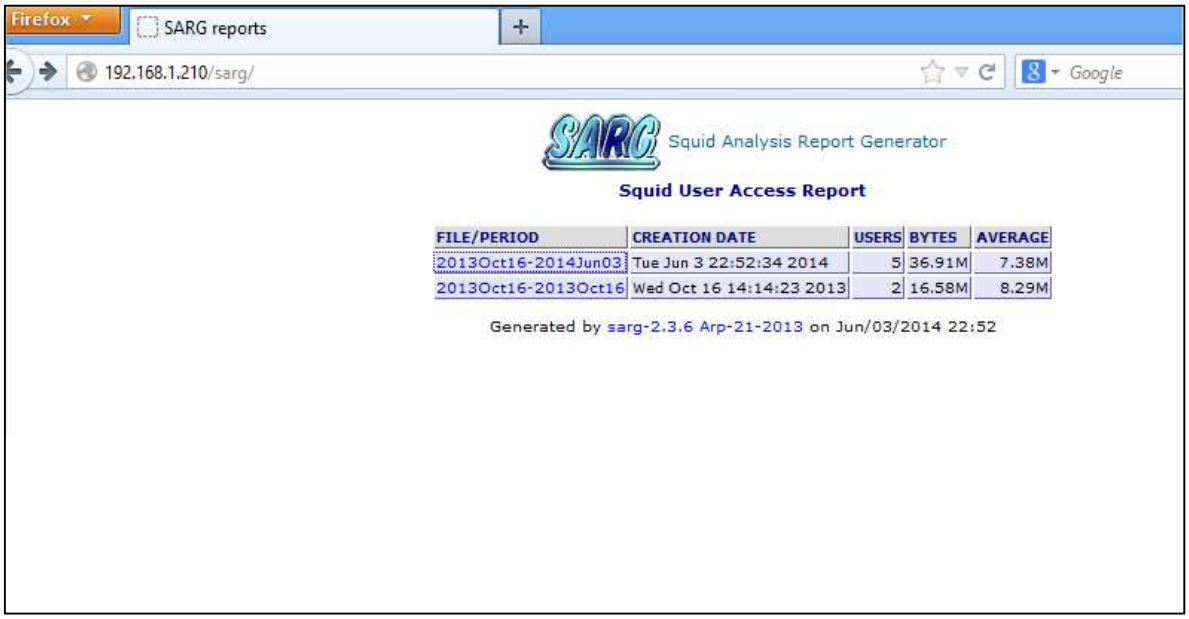

Figura 28: Página principal do SARG
Fonte: Próprio autor (2014)

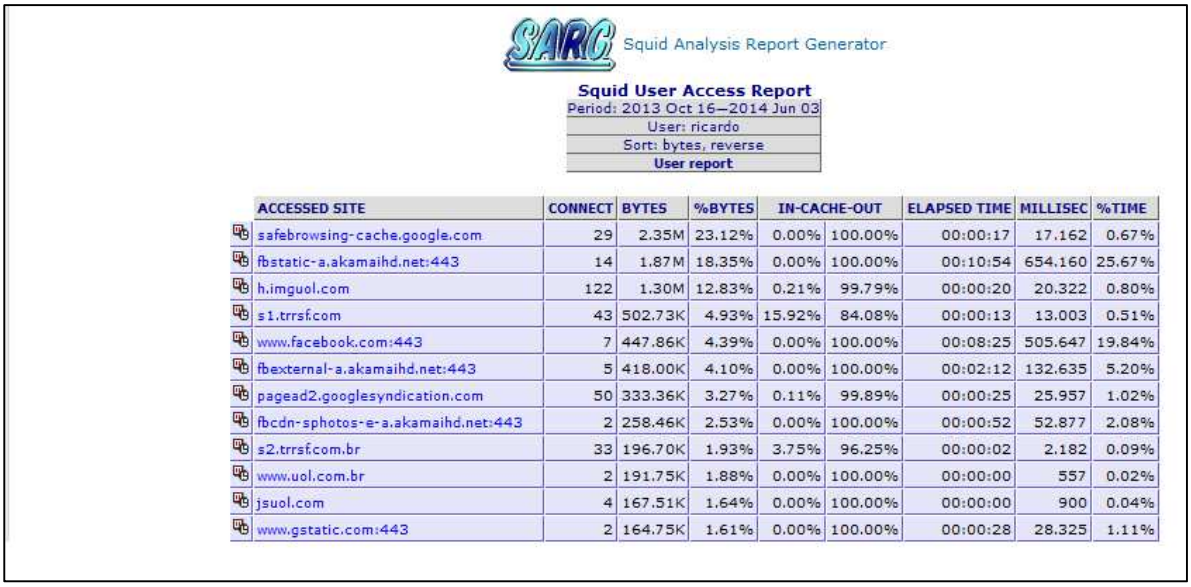

Figura 29: Lista de acessos do usuário Ricardo
Fonte: Próprio autor (2014)

4.7. Regras do *IPFIREWALL* utilizadas

Por padrão, a política do IPFW é fechada, ou seja, sua última regra bloqueia tudo e todos na rede. Isto se torna conveniente, pois o administrador apenas

precisará liberar os serviços necessários na rede corporativa, além do conjunto de regras ser menor.

Abaixo segue o conjunto de regras utilizado neste experimento:

00100 fwd 127.0.0.1,3128 tcp from 192.168.1.0/24 to any dst-port 80

00200 fwd 127.0.0.1,3128 tcp from 192.168.1.0/24 to any dst-port 443

#utilizado para forçar as requisições web do usuário para o proxy

00300 divert 8668 ip from any to any via rl0

#realizando o NAT na interface externa

00400 allow tcp from 192.168.1.0/24 to me dst-port 3128

00500 allow tcp from me 3128 to 192.168.1.0/24

#liberando a porta de acesso do Squid, que por padrão é a 3128

00600 allow udp from 192.168.1.0/24 to any dst-port 53

00700 allow udp from any 53 to 192.168.1.0/24

00800 allow udp from me to any dst-port 53

00900 allow udp from any 53 to me

#liberando o protocolo DNS

01200 check-state

01250 allow tcp from any to me dst-port 22 setup keep-state

01300 allow tcp from me to any dst-port 80 setup keep-state

01400 allow tcp from me to any dst-port 443 setup keep-state

#regras stateful liberando acesso http, https e liberando o protocolo ssh no servidor

01500 allow icmp from me to me

#liberando o ping em localhost

01600 allow icmp from any to any out via rl0 icmptypes 8

01700 allow icmp from any to any in via nfe0 icmptypes 8

01800 allow icmp from any to any in via rl0 icmptypes 0

01900 allow icmp from any to any out via nfe0 icmptypes 0

#liberando ICMP de dentro pra fora, e permitindo somente a resposta

65535 deny ip from any to any

#política fechada do firewall

4.8. *Pentest* realizado ao protocolo SSH

O sistema operacional utilizado para realizar o ataque de força bruta, foi o *Kali Linux*. Este sistema é uma avançada distribuição *Linux* especializada em testes de intrusão e auditoria de segurança. Ela é uma reconstrução completa do *Backtrack Linux*, que incorpora totalmente os padrões de desenvolvimento do *Debian*. Uma infraestrutura completamente nova foi montada, todas as ferramentas foram revistas e empacotadas. Além disso, contém mais de 300 ferramentas de testes de intrusão, onde algumas que não funcionavam foram eliminadas e outras trocadas por outras ferramentas com funcionalidades semelhantes.

Dentre estas ferramentas, a utilizada para realizar força bruta foi o *Hydra*. Entretanto, o *Hydra* depende de um “dicionário” de senhas e combinações para realizar o teste. Para isso, a ferramenta *Crunch* supre as necessidades do teste:

Figura 30: Comando para gerar o dicionário
Fonte: Próprio autor (2014)

Analisando a Figura 30, o *Crunch* irá criar um dicionário contendo no mínimo 3 e no máximo 4 letras, adicionando todas as combinações possíveis no arquivo "*password.txt*". Vale lembrar que, para fins de teste, foi utilizado uma senha root fraca no FreeBSD propositalmente.

Figura 31: Comando do Hydra
Fonte: Próprio autor (2014)

O comando da Figura 31 tenta acesso ao protocolo SSH do *FreeBSD* fazendo uso da conta de super usuário e utilizando o dicionário criado anteriormente.

Figura 32: Ataque realizado com sucesso
Fonte: Próprio autor (2014)

A Figura 32 mostra um ataque que obteve sucesso.

4.8.1. Problema encontrado

Após o sucesso do ataque, foi acrescentada a seguinte regra no IPFW para tentar bloquear:

allow tcp from any to me 22 setup limit src-addr 8

Na teoria, esta regra limitaria em apenas 8 conexões simultâneas o acesso SSH e consequentemente falharia o ataque. Porém, o comportamento do *Hydra* encerrava uma conexão e automaticamente iniciava outra, permitindo ainda a continuação do processo do ataque que, por fim, obteve sucesso pela segunda vez.

4.8.2. O Fail2ban como solução encontrada

O *Fail2ban* é um agente que monitora os *logs* e verifica a quantidade de tentativas de conexão sem sucesso, bloqueando o IP suspeito no *firewall* após determinado número de insucessos. Além disso, o *Fail2ban* consegue monitorar diversos protocolos (inclusive o SSH) em uma ação proativa.

O arquivo ***jail.conf*** (em */usr/local/etc/fail2ban/*) são definidas as regras de bloqueio, quantos minutos os IPs ficam banidos e quantas tentativas de acesso

podem ser feitas. Abaixo segue o trecho da configuração utilizada para o SSH e as descrições:

[ssh-ipfw]

#nome do filtro de ativação

port = ssh

#filtra a porta do ssh

enable = true

filter = bsd-sshd

#nome do filtro

action = ipfw.conf

#direciona para o arquivo de ações

logpath = /var/log/auth.log

#indica o arquivo de log de acesso do SSH

bantime = 3600

#tempo em segundos que o IP fica bloqueado

maxretry = 3

#número máximo de tentativas de conexão

Após isso, também é necessário editar o arquivo *ipfw.conf* (em */usr/local/etc/fail2ban/action.d*) onde estarão as ações que serão feitas em caso de força bruta no ssh. Abaixo segue o trecho do arquivo:

actionban = ipfw table 1 add <ip>

actionunban= ipfw table 1 delete <ip>

Na linha “actionban” é inserido os IPs do atacante utilizando a tabela 1 no IPFW contendo uma lista de IPs bloqueados por tentativas de insucessos de conexões SSH. A última linha é utilizada para “desbanir” um IP.

Além disso, também foi necessário adicionar uma regra no IPFIREWALL bloqueando a lista de IPs acrescentados pelo *Fail2ban:*

00910 deny ip from table(1) to me 22

Por fim, foi adicionada a seguinte linha no arquivo */etc/rc.conf* para subir o serviço do *Fail2ban* a cada inicialização do *FreeBSD:*

fail2ban_enable=”YES”

A Figura 33 mostra o ataque de força bruta ao SSH sem sucesso após a inicialização do *Fail2ban:*

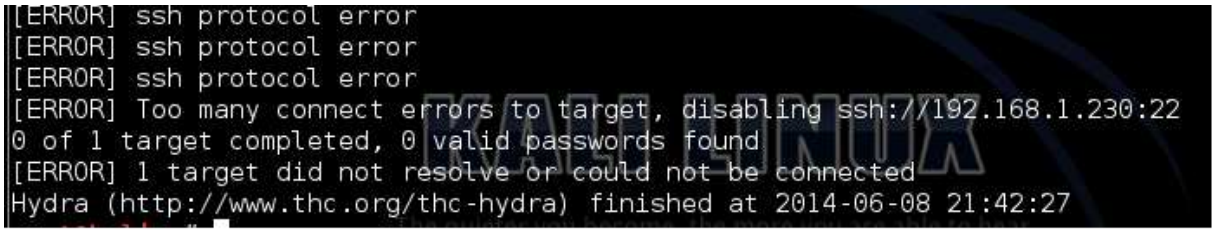

Figura 33: Ataque sem sucesso utilizando o Hydra
Fonte: Próprio autor (2014)

A Figura 34 apresenta o IP do atacante adicionado pelo Fail2ban no IPFW:

Figura 34: Regra adicionada pelo Fail2ban
Fonte: Próprio autor (2014)

Abaixo segue o log do *Fail2ban* em /var/log/fail2ban.log após banir o IP:

2014-06-02 18:30:14,833 fail2ban.actions[4]: WARNING [ssh] Ban 192.168.1.20

5. CONCLUSÃO

A partir dos testes realizados neste trabalho, foi possível observar que o *FreeBSD* mostrou-se eficiente e confiável para atuar como um gateway. A utilização dos dispositivos nativos de segurança vinculados ao proxy *Squid* e *Fail2ban*, tornam o *FreeBSD* uma excelente opção em qualquer ambiente corporativo, onde o foco predominante é a proteção, tanto da rede interna quanto da externa. Além disso, o uso do Fail2ban foi um fator determinante para o sucesso do teste de penetração ao protocolo SSH, pois mesmo utilizando a alternativa de apenas limitar a quantidade de conexões no *IPFIREWALL*, o *Hydra* iniciava outra conexão continuando o processo de ataque.

Na fundamentação teórica foram apresentados ao leitor de forma clara e direta conceitos importantes sobre o *FreeBSD*, *proxy*, *gateway* e *firewall*. Também foram apresentados conceitos de força bruta, além de alguns dispositivos de segurança nativos do *FreeBSD* utilizados para o desenvolvimento do estudo prático.

Por meio deste estudo foi possível concluir também que o *IPFIREWALL* nativo do *FreeBSD* possui características tão eficientes quanto qualquer outro firewall disponível no mercado, além de obter uma configuração mais amigável e não consumir tantos recursos de *hardware*.

Além disso, tal projeto garantiu ao estudante uma visão mais ampla sobre as reais necessidades de segurança digital em um ambiente corporativo, além de estudar a fundo o sistema operacional *FreeBSD* explorando suas vantagens em resultados palpáveis.

Por fim, sendo esse o projeto de graduação, foram verificados duas possibilidades de continuação do estudo como: a inserção do *Fail2ban* em conjunto com o IPFW para prevenção de ataques no *Apache* e o uso do aplicativo *Dansguardian* como filtro de conteúdo *web* dinâmico.

Referências Bibliográficas

BARROS, A. R. **Segurança de Perímetro com FreeBSD.** Lavras, 2011. Disponível em: <http://www.ginux.ufla.br/files/mono-RicherTiago_0.pdf>. Acesso em: 20 Mar 2014.

CERT.BR. **Ataques na Internet.** 2013. Disponível em: <http://cartilha.cert.br/ataques>. Acesso em: 08 Fev 2014.

FAIL2BAN. **The Main Page.** Disponível em: <http://www.fail2ban.org/wiki/index.php/Main_Page> Acesso em: 08 Maio 2014

FREEBSD Documentation Project**. The FreeBSD Documentation Project.** 2013. Disponível em: <http://www.freebsd.org/doc/handbook>

FREEBSD Brasil. **Por que escolher tecnologia FreeBSD?**. Disponível em: <http://www.freebsdbrasil.com.br/home.php?area=2&conteudo=5> Acesso em: 20 Fev 2013.

FUG-BR. **Grupo Brasileiro de Usuários de FreeBSD.** Disponível em: <http://www.fug.com.br>

GONÇALVES, M. D. **Compartilhe a Internet usando FreeBSD + Squid.** Rio de Janeiro: Brasport, 2005. 152p.

KUROSE, F. J.; ROSS, W. K. **Redes de Computadores e a Internet**: Uma abordagem top – down. 3. ed. São Paulo. Pearson Addison Wesley, 2010. 592p.

LUCAS, M. **Absolute BSD:** The ultimate guide to FreeBSD. [S. l.]. [s.n], 2002. 585p. ISBN 1-886411-74-3

MACHADO, M. D.; FURIN, F. A. M. **Análise de usabilidade da ferramenta IPFIREWALL para Firewall.** Revista de Iniciação científica da Libertas. São Sebastião do Paraíso, v.1, n.1, p. 100-113, Dez 2011.

MORIMOTO, E. C. **Gateway.** 2005. Disponível em: <http://www.hardware.com.br/termos/gateway>. Acesso em: 15 Nov 2013.

MORIMOTO, E. C. **Usando o SARG para monitorar o acesso.** 2008. Disponível em: <http://www.hardware.com.br/livros/servidores-linux/usando-sarg-para-monitorar-acesso.html>. Acesso em: 03 Mar 2014

NAKAMURA, T. E.; GEUS, L. P. **Segurança de Redes em Ambientes Cooperativos.** 1ª Edição. São Paulo: Novatec, 2007. 483p.

PACHECO, R. **Squid.** 2007. Disponível em: <http://wiki.ubuntu-br.org/Squid>. Acesso em: 07 Fev 2014

ROCHA, F. V. **Estudo e implementação de Firewall em ambientes corporativos**. João Pessoa. FATEC, 2010.

SHANDRUK, M. W. **IPFW-HOWTO**. 2008. Disponível em: <http://freebsdhowto.com/Ipfw-HOWTO.txt>

TOMASSONI, R. **Gateway com FreeBSD:** Compartilhe a Internet com FreeBSD. Disponível em: <http://www.devmedia.com.br/websys.4/webreader.asp?cat=62&revista=inframagaziii_1#a-3509>

ZANONI, G. **Servidor Proxy (Squid).** 2007. Disponível em: <http://imasters.com.br/artigo/6220/linux/servidor-proxy-squid/>. Acesso em: 01 Jul 2014.