Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação

Detecção e Tratamento de Intrusões em Plataformas Baseadas no XEN

Rafael Antonioli

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Prof. Dr. Luiz Gustavo Leão Fernandes

Porto Alegre
2008

Dados Internacionais de Catalogação na Publicação (CIP)

A635d Antonioli, Rafael.
Detecção e tratamento de intrusões em plataformas baseadas no XEN / Rafael Antonioli. – Porto Alegre, 2008.
77 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Luiz Gustavo Leão Fernandes

1. Informática. 2. Xen. 3. Virtualização. 4. Multiprocessamento (Sistemas Operacionais). 4. Segurança – Computação. I. Título.

CDD 004.35

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "**Detecção e Tratamento de Intrusões em Plataformas Baseadas no XEN**", apresentada por Rafael Antonioli, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 28/03/08 pela Comissão Examinadora:

Prof. Dr. Luiz Gustavo Leão Fernandes – Orientador — PPGCC/PUCRS

Prof. Dr. César Augusto Fonticielha De Rose – — PPGCC/PUCRS

Prof. Dr. Celso Maciel da Costa – — FACIN/PUCRS

Homologada em 02/09/08, conforme Ata No. 18/08 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@inf.pucrs.br
www.pucrs.br/facin/pos

Para alguém especial.

Agradecimentos

Primeiramente, gostaria de agradecer ao professor Luiz Gustavo Leão Fernandes por proporcionar a chance de tornar esse sonho possível, contribuindo com incontáveis conselhos, dicas e oportunidades. Agradeço muito ter sido escolhido seu aluno e orientando, mesmo sem termos mantido nenhum contato prévio, sem dúvida nenhuma posso afirmar que tirei sorte grande na seleção do mestrado.

Agradeço aos colegas dos grupos de pesquisa CPTS/CPSE/CPPH/ex-CPAD pela receptividade quando cheguei em Porto Alegre. E principalmente agradeço aos amigos que conheci no decorrer do mestrado, evitarei citar nomes pra não deixar alguém de fora, eles sabem quem são: a galera que participava dos futebóis semanais na PUC e no Rosário, galerinha das festas nos finais de semana na CB, da vizinhança do Ed. Ada que faziam festas únicas cada vez reservando o salão em nome de algum morador diferente (o zelador sempre proibia de fazer outras festas, por isso eram sempre únicas), sem esquecer também do gurizedo que estava sempre presente na cancha apoiando o glorioso tricolor nas boas e nas ruins.

Agradeço à Lidiane por entrar na minha vida no decorrer de 2007, seu apoio e carinho foi a verdadeira motivação para este trabalho chegar até o final. Agradeço a sua compreensão, que apesar do pouco tempo que estamos juntos, sempre lutou para que o nosso carinho e adoração um pelo outro não terminasse, mesmo morando longe nesses últimos meses.

Agradeço à minha família pelo apoio nos momentos difíceis ao longo desses dois últimos anos, meus pais por terem me colocado no mundo, oferecendo estudo acima de tudo e deixando de lado muitas coisas que eram importantes para eles, sempre objetivando que seus filhos tivessem um estudo qualificado, o qual não tiveram a oportunidade quando crianças. Agradeço principalmente a Lerides, minha Mãe, com "M" maiúsculo, que me ajudou em tudo nessa vida desde que me conheço por gente e principalmente ajudou a pagar os últimos 6 meses de mestrado que não foram nada fáceis.

Resumo

A virtualização de servidores aparece como uma solução para várias demandas atuais dos sistemas computacionais: taxa de ociosidade das máquinas , alto consumo de energia, ocupação de espaço físico e dificuldade para gerenciamento de muitos sistemas operacionais em um mesmo *datacenter*. Entre as alternativas de virtualização disponíveis, o monitor de máquina virtual Xen é uma das opções mais consolidadas e que possui melhor desempenho dentre as demais alternativas existentes. Para proporcionar sistemas virtualizados seguros, um aliado eficaz são os sistemas detectores de intrusão que trabalham realizando monitoração no tráfego da rede.

Este trabalho apresenta uma abordagem para detecção de intrusão em máquinas virtuais baseadas no monitor de máquina virtual Xen, introduzindo uma ferramenta para detectar e bloquear intrusos que estiverem tentando obter acesso indevido ao sistema. A ferramenta elaborada recebeu o nome de XenGuardian e trabalha realizando comunicação entre as máquinas virtuais (domU) com a máquina hóspede (dom0). Na ocorrência de tentativas de acesso não autorizadas, a máquina hóspede realiza o tratamento da ocorrência bloqueando o usuário. Para validar a solução, *exploits* foram utilizados, desferindo ataques contra sistemas de detecção de intrusos e auferindo medições de desempenho através do *benchmark NetPerf*.

Palavras-chave: Sistemas detectores de intrusos, Monitor de Máquina Virtual, Xen.

Abstract

Servers virtualization is a technology which became a solution to several demands of today's computational systems: machines idles cycles, high energy consumption, physical space occupation issues and the difficult to management different operating systems in a same computational environment. Among the existing virtualization platforms, Xen appears as an interesting choice presenting the best performance in comparison with the remaining alternatives. Virtualized systems very often require security and one way to provide safe virtualized systems is through the use of intrusion detection systems which are able to monitor the network data traffic.

This work presents an approach to detect intrusion in Xen virtual machines, introducing a tool to monitor and block malicious or unwanted access to the system. The tool was named XenGuardian and it works using direct communication between the virtual machine (DomU) and the host machine (Dom0) to indicate when a suspicious action is detected. If unauthorized access attempts are identified, the host machine deal with the situation blocking the user responsible for the attack. In order to validate XenGuardian efficiency, exploits were used to perform attacks against the intrusion detection system. Performance measures were obtained through the use of the NetPerf benchmark.

Keywords: Intrusion Detection Systems, Virtual Machine Monitor, Xen.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

IDS	*Intrusion Detection System*	14
VMM	*Virtual Machine Monitor*	14
UML	*User Mode Linux*	15
JIT	*Just-in-time Compilation*	18
IVT	*Intel Virtualization Technology*	24
VMX	*Virtual Machine Extensions*	24
VMM	*Virtual Machine Monitor*	24
AMD-V	*AMD Virtualization*	25
VMCB	*Virtual Memory Control Block*	26
DEV	*Device Exclusion Vector*	26
DMA	*Direct Memory Access*	26
IOMMU	*Input/Output Memory Management Unit*	26
FSB	*Front Side Bus*	26
GPL	*General Public Licence*	27
DHCP	*Dynamic Host Configuration Protocol*	29
CVM	*Cooperative Virtual Machine*	31
XCS	*Xen Control Server*	34
MMU	*Memory Management Unit*	36
VBD	*Virtual Block Devices*	36
VRF	*Virtual Firewall Router*	37
VIF	*Virtual Interface*	37
SSH	*Secure Shell*	39
NIDS	*Network Intrusion Detection Systems*	41
IGMP	*Internet Group Management Protocol*	43
DOS	*Denial of Service*	43
UDP	*User Datagram Protocol*	54
TCP	*Transmission Control Protocol*	54
ICMP	*Internet Control Message Protocol*	65

Sumário

1 Introdução

A virtualização de servidores vem em benefício a diversos fatores que preocupam instituições nos dias de hoje, dentre os quais podemos citar: a taxa de ociosidade de máquinas em *data centers*, ocupação de espaço físico e grande consumo de energia. Esse último tende a causar impactos ambientais preocupantes em um mundo cada vez mais focado em sustentabilidade e alternativas para diminuir a degradação do meio ambiente . Muitos servidores trabalham perto do seu limite de capacidade nos horários de pico, entretanto, são utilizados em média somente 10% a 18% de sua capacidade computacional [3].

Devido ao grande crescimento da quantidade de computadores interconectados ao redor do mundo, o tema sistemas detectores de intrusão vem recebendo fortes incentivos para pesquisas, além de grandes investimentos dentro da área da segurança de redes de computadores. Computadores interligados representam uma oportunidade aos invasores, que tentam a cada dia criar novos métodos de ataques para derrubar sistemas computacionais cada vez mais robustos e caros. Este cenário introduz a necessidade de técnicas mais complexas e elaboradas para detectar e prevenir a ação destes invasores. A grande vantagem de se utilizar Monitores de Máquina Virtual (*Virtual Machine Monitor* - VMM) para implementar sistemas detectores de intrusão é que cada sistema operacional virtualizado possui um espaço de execução isolado [47].

Sistemas Detectores de Intrusão (*Intrusion Detection System* - IDS) são um aliado no reforço da segurança dos sistemas computacionais, onde não basta somente a utilização de *firewalls*, antivírus, análises de *logs* e rígidas políticas de segurança, faz-se necessária uma camada extra de proteção. A detecção de uma intrusão, quando é realizada pela monitoração do tráfego da rede, permite a correção posterior de eventuais problemas de segurança, ou ainda, torna possível a atuação do monitor para bloquear o invasor e interromper um ataque, caso este já tenha sido iniciado.

Neste trabalho é apresentado o *XenGuardian*, um detector de intrusos elaborado para utilização em ambientes virtualizados através do Xen. A principal preocupação ao elaborar uma ferramenta deste tipo é proporcionar o menor *overhead* possível para o sistema, sem comprometer a qualidade da ferramenta, que deve ser capaz de *sniffar* pacotes de rede em tempo real e dar um tratamento adequado a cada um deles. Esse tratamento pode ser a aceitação do pacote, permitindo que ele trafegue na rede, como pode também a rejeição, estado no qual o pacote é descartado, não sendo permitido o seu tráfego pela rede das máquinas virtuais.

1.1 Motivação

Máquinas virtuais podem ser utilizadas para aperfeiçoar a segurança de um sistema de computadores contra ataques a seus serviços. A chamada consolidação de servidores faz uso freqüente de máquinas virtuais, ao invés de usar vários computadores fisicamente separados, utiliza-se apenas um, no qual várias máquinas virtuais isoladas executam diferentes sistemas operacionais convidados, cada qual com suas aplicações e serviços próprios.

A escolha do monitor de máquina virtual Xen baseia-se em pesquisas que apontam custos de virtualização [7] inferiores a 3% para a execução de sistemas operacionais convidados. Além disso, o Xen é um forte candidato para se tornar líder no mercado de virtualização [44] devido ao seu código fonte ser aberto e receber apoio de grandes empresas da área de tecnologia.

Combinar máquinas virtuais com sistemas detectores de intrusos é uma excelente opção para administradores que desejam reforçar a segurança de suas máquinas. Existem trabalhos relacionados com contribuições importantes na área, dos quais podemos citar o trabalho de Laureano [35] onde é apresentada uma modificação na máquina virtual *User-Mode Linux* (UML) para coletar dados detalhados do sistema operacional convidado, como as chamadas de sistema emitidas pelos processos, permitindo identificar anomalias nos processos em execução, consistindo em um sistema de detecção de intrusão baseado em *host*.

Em Kurniadi [6], é apresentada uma proposta de detector e monitoramento de intrusos, baseada em coletas de informações sobre ataques em *honeypots*. O autor deste artigo faz descobertas interessantes, afirmando que o *overhead* no desempenho do sistema é independente de quais eventos estão sendo monitorados, porém depende diretamente da quantidade destes eventos.

O avanço de pesquisas na área de segurança, mesclando detectores de intrusão com sistemas monitores de máquinas virtuais é a motivação para o desenvolvimento da solução apresentada neste trabalho, utilizando diversos conceitos de áreas variadas em prol do incremento da segurança em máquinas interconectadas à rede.

1.2 Objetivos

O objetivo deste trabalho é fornecer um meio de simplificar o controle e a segurança de máquinas virtuais, realizando tarefas automatizadas de análise de tráfego através da comunicação direta entre a máquina hóspede *dom0* [1] e as máquinas hospedeiras *domU* [2]. Uma máquina virtual baseada no Xen tem somente um *dom0*, mas pode ter quantos forem os domU possíveis,

[1] Nomenclatura utilizada no Xen para referenciar o domínio hóspede, responsável por invocar as máquinas virtuais.

[2] Nomenclatura utilizada no Xen para referenciar a máquina hospedeira, a qual é invocada a partir do *dom0*. A sigla *domU* vem de *domain unprivileged*.

de acordo com os recursos disponíveis no sistema (memória e espaço livre no disco rígido).

Essa tarefa é alcançada com o uso de sistemas detectores de intrusão, os quais adicionam uma camada de abstração que tem por finalidade analisar cada pacote que possui origem na rede externa e destino uma máquina *dom0* ou *domU* (pacotes que tem por origem a rede interna *dom0/domU* e destino a rede externa não são analisados).

Para realizar a comunicação entre as máquinas *dom0* e *domU* é utilizada a comunicação via *Berkeley Sockets* [32] em conjunto com a biblioteca *libvirt* [38], a qual permite extrair informações do *hypervisor* do Xen, facilitando a gerência das máquinas virtuais e permitindo criar/destruir domínios de acordo com a necessidade, coletar informações do uso de memória e disco de cada sistema virtualizado e outras informações relevantes para auxiliar no gerenciamento do ambiente virtualizado.

A ferramenta desenvolvida recebeu o nome *XenGuardian* e consiste em um aplicativo baseado no modelo de programação cliente-servidor, onde um servidor e vários clientes executam em *background* trocando informações sobre os pacotes recebidos na rede. O *XenGuardian* é totalmente transparente para o usuário, que não precisará de conhecimentos avançados de segurança de redes e mesmo assim estará seguro contra a exposição de invasores mal intencionados.

A implementação do *XenGuardian* pode ocorrer em qualquer sistema que possui o monitor de máquina virtual Xen e a biblioteca *libvirt*, apesar dos testes serem realizados no sistema operacional Linux/Debian Etch, o funcionamento da ferramenta se dará em qualquer sistema operacional baseado no Linux, desde que possuam instalados as ferramentas pré-requisitas citadas anteriormente.

1.3 Organização do texto

Este documento está organizado da seguinte forma. No Capítulo 2 são apresentados alguns conceitos e tendências na área de virtualização, diferenciando as várias tecnologias em uso atualmente no mercado de virtualização, além de apresentar exemplos dos principais monitores de máquinas virtuais. No Capítulo 3 é descrito em detalhes o monitor de máquina virtual Xen, considerado um produto diferencial e bastante sólido no mercado de virtualização. No Capítulo 4 é apresentado o *Snort*, sistema de detecção de intrusos baseado em rede, além da apresentação de sistemas de *honeypots*. Em seguida, o Capítulo 5 detalha a maneira como o sistema proposto comunica-se com as máquinas virtuais do Xen, como são extraídas as informações do *hypervisor* e explanada a arquitetura da solução apresentada. No Capítulo 6 é exposta a metodologia de testes utilizada, a qual funciona através da exploração das máquinas virtuais com *exploits* e medições de desempenho com o *Netperf*, enquanto a conclusão e alguns possíveis desdobramentos do trabalho são apresentados no Capítulo 7.

2 Conceitos e tendências em virtualização

A área de virtualização está chamando a atenção de grandes empresas da computação, entre as quais se encontram HP, IBM, AMD, Intel, entre outras. Estas empresas estão concentrando esforços na busca de soluções de virtualização e aperfeiçoamento de técnicas já existentes. Idéias e paradigmas antigos da computação [39] vem sendo utilizados para melhorar o desempenho das aplicações tradicionais, para que elas executem em ambientes virtualizados.

Esse capítulo mostra onde as máquinas virtuais podem ser úteis para o avanço da computação, diferencia as técnicas de emulação e virtualização, além de apresentar os novos processadores da AMD e Intel que oferecem instruções especiais para o processador se comunicar diretamente com as máquinas virtuais, sem necessidade de tratar essas rotinas por meio de *software*.

2.1 Aplicabilidade

Um meio de ver as diferentes abstrações das máquinas virtuais é imaginar como se fossem fatias na pilha de *hardware/software*. Um sistema de computador é composto de camadas, começando com o *hardware* na camada mais baixo nível, incluindo camadas de sistema operacional e programas aplicativos que executam sobre o sistema operacional. O *software* de virtualização abstrai as máquinas virtuais, inserindo camadas em vários lugares do sistema [39]. Três exemplos destas camadas de virtualização incluem virtualização no nível de *hardware*, virtualização no nível do sistema operacional e linguagem de alto nível de máquinas virtuais.

Um monitor de máquina virtual possibilita a criação de várias máquinas virtuais com seu próprio sistema operacional, seja ele Linux, Windows ou Mac OS. É como se tivesse mais de uma máquina, porém é apenas um *hardware* físico, onde cada máquina virtual funciona como um microcomputador completo, contendo processador, memória, disco, vídeo, som, todos virtualizados (às vezes emulado) pelo monitor de máquina virtual. Uma máquina virtual pode ainda usar, de forma simultânea com o *dom0*, as unidades de disquete e CD-ROM.

Do ponto de vista do *dom0*, cada máquina virtual é um arquivo. Com isso, pode-se transportar uma máquina virtual de um micro para outro sem problemas, comportamento esse que no Xen recebe o nome de *live migration*. Outro ponto interessante é a possibilidade de ligação entre o sistema hospedeiro e todas as máquinas virtuais, como se estivessem numa rede tradicional sendo cada qual com seu endereço IP. É possível ter um servidor com o sistema ope-

racional Linux e várias máquinas virtuais em execução, todas interligados na mesma rede, ou ainda fazendo partes de redes diferentes, de acordo com a necessidade. Essa topologia de rede é muito útil para testes de funcionalidade de recursos de um servidor (DHCP, proxy e outros) ou para testes de rede entre sistemas operacionais diferentes (Linux e Windows, por exemplo).

Pode haver virtualização em vários níveis dentro de um sistema computacional, em [47] alguns deles são descritos:

Virtualização no nível do *hardware*: a camada de virtualização está sobre o *hardware* que exporta a abstração de máquina virtual. A máquina virtual se parece com o *hardware* e todo o *software* escrito para ele executará na máquina virtual. Esta é a definição de máquina virtual original dos anos sessenta, incluindo tecnologias mais velhas como os IBM *mainframes* VM/370, e também a tecnologia de virtualização VMware em máquina baseadas em x86.

Virtualização no nível do sistema operacional: neste caso, a camada de virtualização está entre o sistema operacional e os programas aplicativos que executam no sistema operacional. A máquina virtual executa um conjunto de aplicações que são escritas para o sistema operacional particular que é virtualizado. *FreeBSD Jails*[1] é um exemplo dessa virtualização.

Linguagem de alto nível de máquinas virtuais: em linguagem de alto nível para máquinas virtuais, a camada de virtualização é como um programa aplicativo rodando em cima de um sistema operacional. A camada exporta uma abstração da máquina virtual que pode executar programas escritos e compilados para a definição particular da máquina abstrata. Qualquer programa escrito na linguagem de alto nível e compilado para esta máquina virtual executará ele. *Smalltalk* e Java são dois exemplos deste tipo de máquina virtual.

Máquina virtual é uma parte de um *software* computacional que permite isolar a aplicação que o usuário está trabalhando. Isso funciona pelo fato de versões de uma máquina virtual serem escritas para várias plataformas, ao invés de produzir versões separadas da aplicação para cada computador ou sistema que se deseja operar. A aplicação é executada no computador, utilizando um interpretador ou executando através de *Just In Time Compilation* (JIT).

O termo máquina virtual é utilizado também para referenciar o ambiente criado por um emulador, quando o *software* é utilizado para emular um sistema operacional para o usuário final. Isto é feito para permitir que aplicações escritas para um sistema operacional possam ser executadas em uma máquina que esteja rodando um sistema operacional diferente, ou para

[1] *FreeBSD Jails* é um recurso nativo do FreeBSD que permite ao administrador particionar o sistema em mini-sistemas independentes, os quais recebem o nome de *Jails*. Cada mini-sistema possui sua própria base de usuários e endereço IP, possuindo também um excelente isolamento entre eles.

fornecer execução *sandbox*[2], oferecendo um grande nível de isolação entre processos do que é obtido quando se está executando múltiplos processos em uma mesma instância de um sistema operacional.

Alguns cenários onde se faz uso de máquinas virtuais são:

Consolidação de servidores: permite centralizar múltiplos servidores em um único *host* físico com desempenho e isolação de falhas provendo uma máquina virtual limitada. Esta técnica permite aumentar a eficiência no ambiente operacional de TI, reduz os custos de manutenção e facilita o gerenciamento dos recursos computacionais.

Independência de *hardware*: permite que aplicações e sistemas operacionais executem em sistemas heterogêneos com máquinas de características diferentes, sem a necessidade de realizar alterações no arquivo de imagem da máquina virtual.

Configurações de múltiplos sistemas operacionais: permite executar múltiplos sistemas operacionais simultaneamente, tornando-se muito útil para realizar o desenvolvimento e testes de novas aplicações.

Desenvolvimento do *kernel*: testes e modificações podem ser feitos no *kernel* de uma máquina virtual, sem a necessidade de uma máquina separada para testes, trazendo ganhos de tempo para desenvolvedores.

Computação em *cluster*: gerência de granularidade de uma máquina virtual, oferecendo mais flexibilidade do que gerenciando separadamente cada *host* físico. Oferece melhor controle e isolamento do que soluções de imagem de um único sistema, particularmente utilizando a *live migration* para balanceamento de carga.

Suporte de *hardware* para sistemas operacionais customizados: permite o desenvolvimento de novos sistemas operacionais beneficiando-se do suporte ao *hardware* variado de existentes sistemas operacionais, tais como o Linux.

2.1.1 Vantagens

Embora existam diferentes níveis de virtualização, a maior parte delas converge para um conjunto comum de atributos, em [47] algumas vantagens são elencadas.

Compatibilidade de *software*: a máquina virtual provê uma abstração de forma que todo o *software* escrito para ela será executado. Uma máquina virtual a nível de *hardware* executará *software*, sistemas operacionais e aplicações escritas para o seu *hardware*. Semelhantemente, uma máquina virtual operando no nível do sistema operacional executará

[2]Sandbox é um mecanismo de segurança para executar programas de forma segura, *applets* e animações em Flash existentes em muitos sites são um exemplo deste mecanismo.

aplicações para aquele sistema operacional em particular, enquanto que uma máquina virtual de alto nível executará programas escritos na linguagem de alto nível. A abstração de máquina virtual pode mascarar diferenças no *hardware* e *software* nas camadas abaixo da máquina virtual. Um exemplo é o slogan do Java *write once, run anywhere* (escreva uma vez, rode em qualquer lugar).

Isolamento: a abstração isola o *software* que executa na máquina virtual de outras máquinas virtuais e reais. Este isolamento provê que pode haver falhas ou ataques de *hackers* dentro da máquina virtual mas que não irão afetar outras partes do sistema. Além de isolamento de dados, a camada de virtualização pode executar isolamento de desempenho, de forma que recursos consumidos por uma máquina virtual não prejudiquem o desempenho de outras máquinas virtuais.

Encapsulamento: permite separar o mecanismo de funcionamento de sua interface. Esta camada é usada para manipular e controlar a execução do *software* na máquina virtual. Esta abstração também pode ser utilizada para prover um ambiente de execução melhor, por exemplo, máquinas virtuais para linguagens de alto nível aceitam verificações em tempo de execução que podem reduzir erros de programação. Isto inclui *type-safe*, *memory-safe* e administração de memória *garbage-collection*. Esta camada provê um ambiente de execução melhor para o programador.

Desempenho: acrescentar uma camada de *software* a um sistema pode acarretar em *overhead*, afetando o desempenho do *software* que executa na máquina virtual. Os benefícios de sistemas de máquinas virtuais bem sucedidos superam o *overhead* que eles introduzem.

2.1.2 Desvantagens

As máquinas virtuais não trazem somente maravilhas para o ambiente computacional, segundo [11], além de necessitar equipamentos mais robustos e de última geração, existem outras dificuldades que surgem junto com a virtualização, como:

Processador não pode ser virtualizado: Com exceção das novas tecnologias de virtualização da Intel e AMD, conhecidas respectivamente por Intel VT e AMD-V, que são vistas com maiores detalhes na seção 2.3 deste mesmo capítulo. A arquitetura dos processadores 32 bits da Intel não permite naturalmente o uso da virtualização pelo fato de não ter sido concebida para executar múltiplos sistemas operacionais ao mesmo tempo.

Diversidade de dispositivos de hardware: A arquitetura aberta de um computador permite que fabricantes desenvolvam equipamentos com as mais variadas características e mais variado *hardware*. Faz-se necessário um esforço de programação muito grande para que

o monitor de máquina virtual seja capaz de reconhecer a vasta variedade de dispositivos de *hardware* com *drivers* diferentes.

Custo de execução dos processos: Dependendo do *hardware* que se está utilizando, o monitor de máquina virtual pode haver um *overhead* gerado pela camada de virtualização, que chega a até 35% [33]. Ambientes com suporte a virtualização tendem a apresentar um *overhead* muito menor pelo fato de possuírem instruções específicas para interagir com as máquinas virtuais, desonerando assim o *software* monitor de máquina virtual.

Os desenvolvedores do Xen garantem que o *overhead* gerado por ele gira em torno de 3% [7], desenvolvedores do VMWare e de outras soluções de virtualização discordam dos dados e disponibilizam resultados diferenciados, onde mostram que seu produto obtém melhor desempenho [54]. Por último a equipe responsável pelo Xen publicou novos resultados [57], afirmando que a versão comercial do Xen, chamada de XenEnterprise, na sua versão 3.2 possui um desempenho igual ou superior se comparado ao VMware ESX Server 3.0.1

2.2 Emulação, virtualização e *porting*

É muito comum no mundo das máquinas virtuais se deparar com nomenclaturas confusas. Alguns sistemas são chamados de emuladores, outros dizem empregar técnicas de virtualização, para-virtualização, *dynamic translation* e *porting*. Essa seção procura explicar melhor esses conceitos.

2.2.1 Emulação

Emulador é um *software* com a função essencial de transcrever instruções de um processador alvo para o processador no qual ele está sendo rodando. Muitas pessoas utilizam emuladores para rodar jogos de *video games* antigos, que naturalmente possuem poucos requisitos de memória e poder de processamento, perto do que os poderosos computadores de mesa oferecem atualmente.

Em um sistema emulado é praticamente impossível atingir 100% da velocidade de um sistema original. Além de requerer grande otimização, também se faz necessário possuir um processador muito mais poderoso e capaz de processar mais instruções por segundo que o sistema original [24].

Outra desvantagem da emulação é a compatibilidade, pois o sistema original a ser emulado precisa ser simples e muito bem documentado. Dependendo da ocasião, alguns periféricos e características do sistema original podem ser impossíveis de emular.

Um exemplo de emulador para sistema operacional é o Bochs [37], o qual possui uma boa documentação, é escrito na linguagem C++ e pode ser portável para várias plataformas. O Virtual PC [13] da Microsoft também se encaixa nessa categoria, sendo possível emular um sistema operacional Mac OS, dentro de uma janela do Windows XP.

2.2.2 Virtualização

Virtualização [17] é o processo de apresentar um grupo ou subgrupo lógico de recursos computacionais, de modo que possam ser acessados de várias maneiras, oferecendo benefícios sobre a configuração original. Nesta nova visão virtual os recursos não são limitados pela implementação, localização geográfica ou configuração física dos recursos.

Um tipo popular de virtualização, atualmente muito referenciado na mídia, é a virtualização de *hardware*, utilizada para executar mais de um sistema operacional ao mesmo tempo através de *nanokernels* [3] ou abstração das camadas do *hardware*, como por exemplo o Xen.

Uma máquina virtual é um ambiente que aparece como um sistema operacional convidado ao *hardware*, mas é simulado em um ambiente de *software* contido dentro do sistema operacional hospedeiro. A simulação deve ser robusta o bastante para que os dispositivos do *hardware*, possam operar dentro do sistema operacional convidado.

Alguns tipos de virtualização existente são:

Virtualização Completa: a máquina virtual simula o *hardware* completo, permitindo um sistema operacional não modificado usar uma CPU completamente diferente. Se encaixam nessa categoria o Bochs e o Virtual PC, na versão PowerPC.

Para-Virtualização: a máquina virtual não simula o hardware, porém oferece uma API especial que requer modificação no *kernel* nativo do sistema operacional, a exemplo do Xen.

Virtualização: a máquina virtual somente simula partes do *hardware* para permitir que um sistema operacional não-modificado seja executado de forma isolada, porém o sistema operacional convidado deve ser designado para o mesmo tipo de CPU, como por exemplo o VMware e a versão x86 do Virtual PC.

2.2.3 *Porting*

Porting é a capacidade que um *software* tem de ser compilado ou executado em diferentes arquiteturas de sistemas computacionais, sejam diferenças na arquitetura de *hardware* ou de sistema operacional [37]. O *porting* ocorre através da modificação no código fonte original,

[3] Pequeno *kernel* utilizado normalmente para tratar processamento de interrupções geradas pelo *hardware*.

permitindo a leitura e interpretação em outra linguagem, se um *software* não for portável para outra plataforma, a única maneira de sua execução será através da emulação. Geralmente um *software* que foi portado de uma linguagem para outra possui melhor desempenho se comparado com um software que foi executado através de técnicas de emulação.

O termo não se aplica ao processo de adaptar um *software* de maneira que possa executar em um computador com menos memória, porém com a mesma CPU e sistema operacional, ou a reescrita para uma linguagem diferente. Isto recebe o nome de conversão de linguagem ou tradução.

2.2.4 *Dynamic Translation*

Dynamic Translation também é conhecido como *Just-in-time Compilation* (JIT) e consiste em uma técnica para aumentar o desempenho de execução de um sistema. Antes que seja executado um código interpretado, ele é convertido para o código nativo da CPU onde será executado. Esse código nativo recebe o nome de *bytecode*.

Bytecode não é o código de máquina para um computador em particular, podendo ser portável entre arquiteturas de computadores diferentes. Esse código é então interpretado ou executado em uma máquina virtual.

A *Dynamic Translation* analisa uma curta seqüência de código, tipicamente na ordem de um único bloco, traduzindo e armazenando os resultados. Exemplos de sistemas que utilizam essa técnica são as linguagens de programação Smalltalk, Perl, GNU CLISP e algumas versões mais antigas de Java.

A *Dynamic binary translation* [30] se diferencia de técnicas de emulação por eliminar o método principal de emulação, um laço de leitura-decodificação-execução, onde o sistema perde em desempenho. Pagando por isso um grande *overhead* durante o tempo de tradução. Este *overhead* é compensado pois as traduções da seqüência de códigos são executadas em múltiplos tempos.

O QEMU funciona como um tradutor dinâmico [9]. Quando encontra um pedaço de código, o mesmo é convertido para o conjunto de instruções da máquina *host*, posteriormente executando-o.

2.3 Novas tecnologias de virtualização

Com os grandes avanços da virtualização tanto na área acadêmica como nas áreas comerciais, a partir de 2005 fabricantes de processadores não mediram esforços para lançar componentes que saíssem de fábrica já preparados para a grande demanda de produtos prontos para virtualização que estariam surgindo.

Primeiro foi a vez da Intel, seguida pela AMD, ambas foram responsáveis por inserir no mercado processadores com instruções adaptadas para trabalhar com monitores de máquinas virtuais e assim, realizar tarefas com grande desempenho. A seguir veremos como funcionam essas duas tecnologias.

2.3.1 Intel VT

Em 2005 foram lançados, pela Intel, os primeiros processadores Pentium 4 com tecnologia de virtualização. Também conhecida como *Intel Virtualization Technology* (IVT), essa tecnologia detinha, durante a fase de projeto, o nome preliminar de *Vanderpool*. Esta habilita que um processador funcione como se existissem vários processadores trabalhando em paralelo, de modo a permitir que vários sistemas operacionais sejam executados ao mesmo tempo em uma mesma máquina.

A vantagem de implementar uma tecnologia de virtualização dentro de um processador, é permitir que novas instruções controlem exclusivamente a virtualização. Com essas instruções o monitor de máquina virtual pode ser mais simples, obtendo um maior desempenho se comparado a soluções baseadas exclusivamente em *software*.

Processadores com tecnologia de virtualização possuem um conjunto de instruções extra chamado de *Virtual Machine Extensions* [52] ou VMX, trazendo 10 novas instruções específicas de virtualização para o processador: VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUCH, VMRESUME, VMXOFF e VMXON.

Existem dois modos de execução dentro da virtualização: *root* e não-*root*. Normalmente, apenas o *software* de controle da virtualização, chamado *Virtual Machine Monitor* (VMM), roda no modo *root*, enquanto que os sistemas operacionais trabalhando no topo das máquinas virtuais rodam no modo não-*root*. Os programas do usuário são executados no topo das máquinas virtuais.

Para entrar no modo de virtualização, o programa deve executar a instrução VMXON e então chamar o *software* VMM. Em seguida, este *software* pode entrar em cada máquina virtual usando a instrução VMLAUNCH, e sair delas usando a instrução VMRESUME. Se o VMM quiser parar todas as máquinas virtuais e sair do modo de virtualização, ele executa a instrução VMXOFF.

Cada convidado mostrado na Figura 1 pode ser um sistema operacional diferente, rodando o seu próprio *software* e provavelmente rodando dentro dele vários programas.

Para toda essa tecnologia funcionar é necessário *chipset* e placa-mãe adequados para o reconhecimento das novas instruções, além de um monitor de máquina virtual que faça o uso dessas novas chamadas de sistemas. Desempenho e funcionalidade dependerão das configurações de *hardware* e *software*.

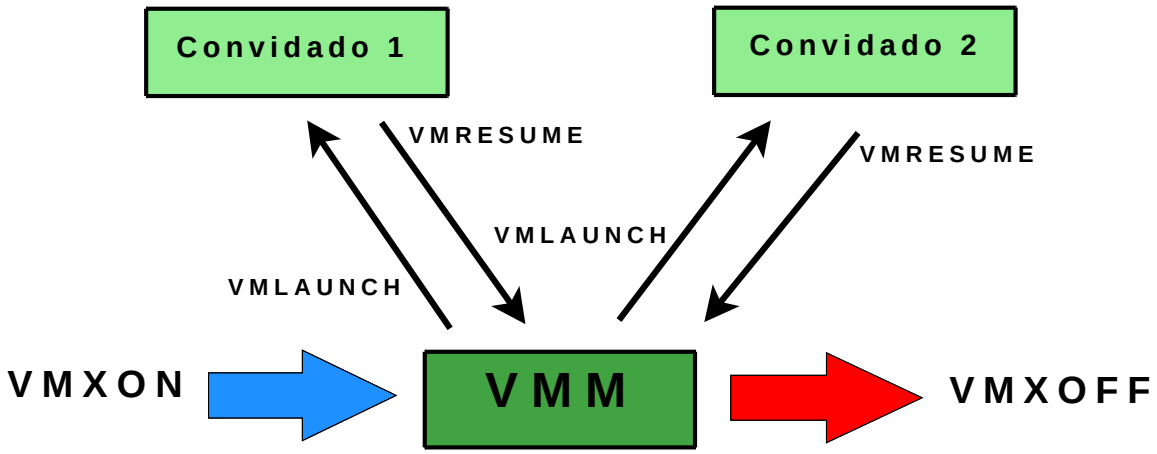

Figura 1 – Operação da tecnologia de virtualização da Intel [52].

2.3.2 AMD-V

A tecnologia de virtualização da AMD é conhecida por *AMD Virtualization* (AMD-V). O projeto original recebia o nome *AMD Pacifica*, no entanto sua nomenclatura foi alterada para lançamento no mercado. A tecnologia chegou aos processadores em 2006, um pouco mais tarde que a tecnologia *Vanderpool* da Intel, oficialmente chamada Intel *Virtualization Technology* (Intel-VT).

O projeto do *AMD Pacifica* é baseado na plataforma 64 *bits*. O objetivo é estender a tecnologia AMD64 utilizando uma arquitetura chamada *Direct Connect* para aumentar a virtualização de *desktops* e servidores, introduzindo um modelo com novas características no processador e controlador de memória [5]. O projeto procura estender o modelo tradicional de virtualização, que utiliza somente *software*, para uma nova abordagem utilizando chamadas diretamente ao *hardware*. Essas novas funcionalidades podem reduzir a complexidade dos *softwares* de virtualização, aumentar a segurança de novas soluções e manter a compatibilidade existente de virtualização.

Como em aperfeiçoamentos anteriores de arquiteturas, a AMD procura manter a compatibilidade do AMD-V com aplicações já escritas para x86 e AMD64, não necessitando alterações em *software*.

São características da arquitetura *AMD Pacifica*, segundo McDowell [40]:

- Novo modo do processador: ***Guest Mode***;
- Nova estrutura de dados: ***Virtual Machine Control Block*** **(VCMB)**;
- Nova instrução: **VMRUN**;
- Novo modo de memória: ***Real Mode with Paging***;
- Proteção de acessos externos através do ***Device Exclusion Vectors*** **(DEV)**;

- Aumento de desempenho habilitando para-virtualização através da técnica de ***Selective Interception***;
- Suporte para **SKINIT** (*secure kernel init*);
- Melhorias no gerenciamento de memória, com as técnicas ***Tagged TLB*** e ***Nested Page Table Support***.

Na arquitetura AMD-V a virtualização é baseada na instrução VMRUN. Ela pode ser executada pelo sistema operacional hospedeiro, fazendo uma chamada para executar o sistema operacional convidado. Todo o estado da CPU para o sistema operacional convidado fica localizado na estrutura de dados *Virtual Memory Control Block (VMCB)*.

Para aumentar a velocidade de um computador e diminuir a carga na CPU, computadores x86 usam um mecanismo chamado *Direct Memory Access* (DMA) que permite que certos dispositivos de *hardware* acessem a memória para leitura e escrita, independentemente da CPU.

Tanto a tecnologia AMD-V quanto a Intel VT utilizam uma nova técnica de acesso a dispositivos de entrada e saída, a AMD chama a tecnologia de *Input/Output Memory Management Unit* (IOMMU) [20], enquanto a Intel chama de *Intel Virtualization for Directed I/O* (VT-d). As duas técnicas funcionam basicamente da mesma maneira e permitem melhorar o desempenho e segurança na hora de alocar memória e acessar dispositivos. A Figura 2 ilustra o funcionamento da IOMMU.

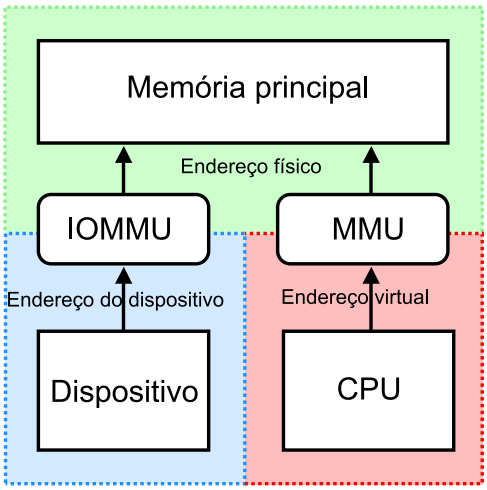

Figura 2 – Diagrama da tecnologia IOMMU [20].

A técnica de conexão direta apresenta um controlador de memória integrado à pastilha de silício, evitando o uso de barramentos frontais (FSB). Os processadores, controlador de memória e subsistema de I/O ficam conectados diretamente com a CPU realizando uma comunicação em alta velocidade.

Um dispositivo que age diretamente sobre a memória pode ser perigoso pois se um programa executando em um sistema operacional convidado faz com que um *drive* de disquete carregue

um arquivo de 100 *kilobytes* na memória, esses *kilobytes* podem ser vistos de maneira diferente pelo programa, pelo sistema operacional convidado e pelo VMM. Também é possível que um único sistema operacional convidado imagine que é dono exclusivo de um *drive* de disco e outro sistema operacional também tenha essa mesma idéia.

Uma característica importante do AMD-V é chamada *Device Exclusion Vector* (DEV). Esta funcionalidade, no entanto, não resolverá todos os problemas citados no parágrafo anterior, porém dará ao programador do VMM as ferramentas necessárias para auxiliar no desenvolvimento de um código fonte mais limpo e confiável do *software* de virtualização. O DEV é um pedaço da memória que diz a um dispositivo se é permitido acessar uma página de memória ou não.

2.4 Exemplos de Máquinas Virtuais

Esta seção apresenta as principais máquinas virtuais disponíveis no mercado, tanto sob a licença da *General Public Licence* (GPL), como também soluções comerciais.

Algumas soluções são aplicáveis somente a sistemas Linux, enquanto outras executam em ambientes Microsoft Windows ou ainda Mac OS. Nesta seção não será feita nenhuma comparação de desempenho entre máquinas virtuais, apenas uma demonstração das suas principais características.

2.4.1 Virtual PC

O Virtual PC [13] é um *software* de emulação para Apple Mac OS X e permite também virtualização para sistemas operacionais baseados no Microsoft Windows.

Escrito originalmente pela Connectix, empresa que foi adquirida pela Microsoft, que optou por dar continuidade ao projeto. Com o Virtual PC é possível ter um Windows XP e realizar testes com um servidor Windows 2003 e um servidor SQL, sem a necessidade de formatar a máquina e instalar cada sistema separadamente.

A Microsoft possui dois produtos de virtualização hoje no mercado: o Virtual PC para sistema operacional cliente, como por exemplo o Windows XP; e o Virtual Server 2005, que é um *software* mais robusto e sua instalação deve ser realizada somente em sistemas operacionais servidores, tais quais o Windows Server 2003.

Existe uma versão do Virtual PC para computadores da linha Macintosh da Apple, que possuem processadores Power PC. Através de técnicas de emulação é possível utilizar um sistema baseado na arquitetura x86 para rodar o Mac OS, incluindo todos os componentes básicos do *hardware*. Utilizando um *hardware* emulado cada máquina virtual trabalha como se fosse um computador físico separado, sendo possível alternar facilmente do Mac OS para o Windows com um simples clique do mouse.

2.4.2 User-Mode Linux

O *User-Mode Linux*(UML) [18] é um *kernel* do Linux modificado especialmente para rodar como um processo normal, dentro de um sistema Linux nativo. O UML não virtualiza um ambiente inteiro, apenas as chamadas de sistema e alguns dispositivos acessados freqüentemente pelos processos. Os processos que rodam no UML não podem lidar diretamente com o *hardware*, somente recebem autorização para requisitar serviços ao *kernel*, o que torna relativamente fácil virtualizar um ambiente para eles.

Como esse tipo de virtualização não envolve emular CPU nem *hardware*, a velocidade chega perto da nativa. O consumo de memória de cada sessão virtual é proporcional ao requisitado pelos processos rodados ali dentro, diferente de algumas máquinas virtuais que alocam toda a memória RAM virtual logo no início da emulação. A velocidade e praticidade do UML permite o uso do mesmo como ferramenta de segurança, isolando-se um *software* servidor de um Linux virtual, torna-se muito difícil um invasor tomar conta do sistema, mesmo que ele consiga permissões de *root* dentro dele.

2.4.3 VMware

VMware é mais uma opção de monitor de máquina virtual [54]. Trata-se de um dos primeiros emuladores de sistemas operacionais, apesar de ser um *software* comercial, existem no mercado versões gratuitas, porém com quantidade de recursos reduzida. É um produto de boa qualidade competindo diretamente com o XenEnterprise [57], a versão paga do Xen. Pelo fato do VMware não emular a CPU, o código do sistema operacional convidado executa diretamente sobre a CPU real, em velocidade próxima a nativa.

Inicialmente só existiam versões para Linux e Windows. Recentemente a VMware Inc. está desenvolvendo uma versão para o Mac OS X, para ser executada exclusivamente em *Macintosh* com processadores Intel. Essa versão recebeu o nome de *VMware Fusion* e atualmente encontra-se na fase *beta* com poucos recursos e muitas falhas.

No início de novembro de 2007 esses eram os principais produtos oferecidos pela VMware Inc. [53]:

VMware Player: versão gratuita de um player de máquinas virtuais. Permite que máquinas virtuais criadas previamente possam ser distribuídas e colocadas em produção ou testes. Esta versão não permite a criação de máquinas virtuais;

VMware ACE: ACE vem de *Assured Computing Envinroment for Enterprise*, um ambiente de máquina virtual baseado em uma variação de VMware Player, que permite disponibilizar PC's virtuais para terceiros, tais quais prestadores de serviço, consultores e parceiros, com

isolamento controlado e gerenciável da rede corporativa. Oferece controle de expiração do sistema, configurações avançadas de segurança, rede e sistema em geral, incluindo controle da interface do usuário;

VMware Workstation: Atualmente na versão 6.0, é voltada para virtualização de *desktops*, sendo que uma licença pode ser adquirida por U$ 189,00 para cada máquina *host*. É geralmente utilizada para testes preliminares de máquinas virtuais ou para uso em sistemas não críticos. Oferece suporte para 19 versões do Windows e 26 versões do Linux. O suporte ao *hardware* é bastante amplo, oferecendo até 8 GB de RAM por máquina virtual, suporte a USB 2.0, dispositivos *wireless* e opções avançadas de rede, com servidor DHCP incorporado e suporte para até 10 *switches* virtuais;

VMware Server: antigamente esta versão era conhecida como **VMWare GSX Server**. Trata-se de uma versão aprimorada do ambiente de virtualização que roda somente em sistemas operacionais servidores (Windows 2000/2003 Server) e Linux. Essa versão inicialmente era paga, porém com a grande competitividade com outras máquinas virtuais gratuitas, tais quais o Xen, a versão foi disponibilizada gratuitamente. Pelo fato de ser uma versão gratuita, algumas tecnologias estão em caráter experimental, como por exemplo o suporte ao Virtual SMP, com dois processadores e tecnologia de virtualização da Intel;

VMware ESX Server: é o produto mais robusto de virtualização da VMware Inc. Atualmente está na versão 3 e diferencia-se dos demais produtos da empresa. Enquanto as soluções anteriores funcionam por meio de um *software*, esta versão baseia-se em um sistema operacional próprio, constituído de um *kernel* Linux modificado, otimizado para esta função. Esta solução é indicada para ambientes sofisticados, críticos e consolidação de servidores em larga escala.

2.4.4 Bochs

De acordo com Lawton [37] e [36], Bochs é um emulador de máquina virtual que possui código aberto, escrito em C++, sendo portável para processadores x86 e x86-64. Aceita emulação de processador (incluindo modo protegido), memória, discos, vídeo, rede, BIOS e periféricos comuns de PC's.

Diferente de virtualizadores e para-virtualizadores, o Bochs apenas emula o *hardware*, permitindo executar muitos sistemas operacionais convidados, entre eles: MS-DOS, várias versões do Windows, PPC, Alpha, Sun, BSDs e Linux. Também são diversos os sistemas operacionais *hosts* que podem ser executados, incluindo Windows, Linux, Mac OS X, o video game da Microsoft Xbox e o console de video game PSP da Sony.

Este emulador é utilizado principalmente para desenvolvimento de sistemas operacionais. Quando um sistema operacional entra em *crash*, essa falha não afeta o sistema operacional

hóspede e o sistema operacional emulado pode ser depurado. Também é utilizado para executar outros sistemas operacionais convidados dentro do sistema operacional hóspede, sendo que algumas pessoas utilizam esta funcionalidade para executar jogos antigos de computador dentro de computadores não compatíveis de forma nativa.

O Bochs pode emular o *hardware* necessário para o sistema operacional convidado, incluindo discos rígidos e *drives* de CD e disquete. Discos e imagens ISO podem ser inseridas enquanto o sistema está sendo executado, porém o desempenho do sistema é muito lento pelo fato de ser emulado. Não é uma opção para aplicações mais pois perde em desempenho se comparado ao VMware e ao Xen.

2.4.5 Plex86

Plex86 [51] é um projeto para criar uma máquina virtual para a arquitetura x86 que executa o sistema operacional Linux. No passado houve um pouco de confusão sobre a meta principal do Plex86, sendo conhecido como uma versão mais básica do Bochs. Possui muitos atributos em comum com o Bochs, exceto que enquanto o este tenta emular o sistema por inteiro, consumindo consideravelmente mais recursos do sistema.

Este utilitário emula partes do sistema absolutamente necessárias para executar um sistema operacional Linux. Como tal, emulará somente um processador baseado em Intel 386+ e só executará no sistema operacional Linux, porque neste sistema operacional os programadores tem acesso mais fácil aos dispositivos de *hardware*.

O *software* do Plex86 está atualmente na etapa alpha. Seu site foi transferido para o Sourceforge em meados de 2003, antes disso era hospedado em outro lugar. A atividade sobre este projeto é incerta, sendo o último artigo de notícias da página oficial do projeto publicado em 19 de dezembro de 2003, o qual estava pedindo doações para ajudar a manter o projeto vivo.

2.4.6 QEMU

QEMU [9] é um *software* gratuito escrito por Fabrice Bellard que implementa um emulador de processador rápido, permitindo ao usuário executar um sistema operacional dentro de outro. É semelhante a projetos como o Bochs, VMware Workstation e PearPC, mas tem várias características que faltam nestes projetos, incluindo maior velocidade em x86 e suporte para múltiplas arquiteturas (em progresso). Utilizando *dynamic translation* consegue alcançar uma velocidade razoável, sendo fácil portar novas CPUs *hosts*. QEMU tem dois modos de operação:

***User mode emulation*:** QEMU pode lançar processos Linux compilados para uma ou outra CPU. Chamadas de sistema do Linux são convertidas na representação dos dados de 32/64

bits. Permite facilmente realizar *cross-compilation*[4] e *cross-debugging* [5];

***System mode emulation*:** QEMU emula um sistema completo, incluindo um processador e vários periféricos. Permite facilmente fazer teste e depuração de códigos do sistema. E também pode ser utilizado para prover *hosts* virtuais de vários PC's em um único servidor.

O QEMU pode também ser executado sem um *kernel* hospedeiro, fornecendo um desempenho aceitável. Para emulação de sistema suporta desde processadores x86/x86-64, até PC's antigos sem barramento PCI, que utilizam *slot* ISA. Emula também PowerPC, ARM, MIPS e CPU's Sparc32/64. Suporta SMP, permitindo a utilização de até 255 CPU's.

Durante o ano de 2007 foi anunciada a disponibilidade sob a licença GPL, do KQEMU, também conhecido como *The QEMU Accelerator*, um driver para uso juntamente com o *kernel* do Linux, que melhora o desempenho do QEMU na ordem de 50% a 100% [10], o que o coloca em um nível de desempenho similar ao VMware. Com essa técnica de aceleração, ao invés de interpretar as chamadas das máquinas hospedeiras, o QEMU as repassa diretamente para a máquina hóspede.

2.4.7 Cooperative Linux

Também conhecido por coLinux [14], é um *software* que permite que o *Microsoft Windows* coopere com o *kernel* do Linux para executar ambos em paralelo dentro de uma mesma máquina. Utiliza o conceito de *Cooperative Virtual Machine* (CVM) , em contraste com máquinas virtuais tradicionais e compartilha recursos que já existem no sistema operacional hospedeiro. No sistema hóspede de máquinas virtuais tradicionais, recursos são virtualizados para todos sistemas operacionais convidados. A CVM dá para ambos os sistemas controle completo da máquina hóspede, enquanto máquinas virtuais tradicionais definem para todo sistema operacional convidado um estado não-privilegiado para acessar a máquina real.

O termo cooperativo é utilizado para descrever duas entidades que trabalham em paralelo. *Cooperative Linux* se transforma em dois diferentes *kernels* de sistemas operacionais dentro de duas grandes sub-rotinas. Cada *kernel* tem seu próprio contexto completo da CPU e seu espaço de endereçamento e cada *kernel* decide quando devolver o controle ao seu companheiro. Como o próprio nome sugere, funciona apenas instalado em *Microsoft Windows*, tanto na versão 2000 como na XP.

[4]Técnica que permite criar um código executável para uma plataforma diferente da que o cross compiler foi executado.

[5]Técnica que permite fazer *debug* de um código executável para uma plataforma diferente da qual foi projetado.

2.5 Conclusão do capítulo

É importante conhecer o surgimento da virtualização e perceber como aproveitá-la no dia-a-dia, seja para reduzir a quantidade de servidores físicos, gerar economia de recursos ou melhorar o aproveitamento dos recursos computacionais disponíveis através da diminuição da ociosidade das máquinas. Os principais fornecedores de *hardware* e *software* estão entrando com muita disposição no mundo da virtualização, um conceito que apesar de ter surgido nos anos 80, nos dias de hoje ainda é visto como revolucionário na área da computação. Dia após dia são inúmeras as melhorias apresentadas pelos monitores de máquinas virtuais, sejam elas a maior compatibilidade com *drivers* dos mais diversos fabricantes, a otimização de comunicação entre máquinas virtuais, a diminuição de camadas de comunicação entre subsistemas, a implementação via *hardware* de rotinas de virtualização, dentre outras. Todos esses avanços tecnológico objetivam facilitar o uso da virtualização para usuários mais leigos, além de melhorar o desempenho global das aplicações que são virtualizadas.

Este capítulo apresentou outras soluções existentes de virtualização, mas que não foram utilizadas neste trabalho pelos mais diversos motivos: licença com custo muito elevado, desempenho não satisfatório, pouco suporte pela comunidade, fraco referencial bibliográfico, dentre outros.

3 Máquina Virtual Xen

Atualmente no mercado existem diversos produtos para virtualização de servidores, cada qual com suas peculiaridades e público alvo. Há soluções de custo elevado com código fonte fechado e direcionado para grandes *data centers*, além de soluções baseadas em código fonte aberto, para uso acadêmico ou em empresas de porte médio, com bom desempenho e grande aceitação no mercado.

O uso da virtualização permite reunir aplicações de sistemas operacionais distintos em uma só máquina realizando, assim, uma melhor gerência da capacidade de processamento dos computadores, otimizando o uso da CPU real do sistema através da diminuição da ociosidade do processador, além de facilitar a gerência dos múltiplos ambientes virtuais pelo fato de estarem fisicamente no mesmo local.

Este capítulo faz uma abordagem do monitor de máquina virtual Xen, uma solução de virtualização consolidada no mercado que, além de gratuita, possui código fonte aberto permitindo a realização de análises no código-fonte e possibilitando a compreensão de como funciona a comunicação do software com o hardware da máquina.

3.1 Estrutura e componentes

Xen [7] é um monitor de máquina virtual desenvolvido pela Universidade de Cambridge, licenciado pela GPL[1], que utiliza técnicas de para-virtualização para emular a arquitetura de processadores das famílias x86, x86-64, IA-64 e Power PC.

No mundo do Xen, utiliza-se a nomenclatura *dom0* para designar a máquina hóspede do sistema operacional e *domU* para designar o sistema operacional virtualizado, desprivilegiado em relação à comunicação com os dispositivos do sistema. Todas as chamadas de sistema de um *domU* precisam obrigatoriamente passar pelo *dom0*. Em uma máquina podem existir vários *domU*, porém só existe um único *dom0*.

Diferente do VMware [53] e do Virtual PC [13], para o Xen funcionar é necessário substituir o *kernel* do sistema *dom0* por um *kernel* modificado. Este deve ser apto a interpretar as instruções recebidas pelas máquinas virtuais do sistema operacional convidado, o mesmo ocorrendo com o *kernel* das máquinas virtuais *domU*.

O modelo de proteção da arquitetura de processadores x86 é construído por quatro anéis: o

[1]GPL é o acrônimo para *General Public License*.

anel 0 é para o sistema operacional e o anel 3 é para as aplicações de usuário. Os anéis 1 e 2 não são usados, exceto em casos raros, como no sistema operacional OS/2 [16]. No Xen, um *hypervisor* executa no anel 0, enquanto sistemas operacionais convidados executam no anel 1 e aplicações executam no anel 3. Na arquitetura x86/64 o sistema operacional convidado e suas aplicações executam no anel 3 [44]. A Figura 3 ilustra os anéis de proteção na arquitetura x86 do Xen.

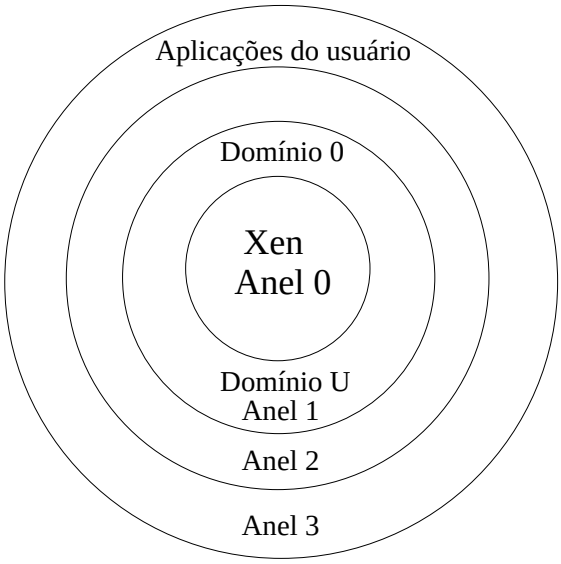

Figura 3 – Topologia em anel no Xen [46].

O Xen é chamado de *hypervisor* porque opera em um nível mais privilegiado que o código do sistema operacional convidado que ele hospeda. No momento do boot do sistema, o Xen [46] é carregado na memória do anel 0 e então inicia um *kernel* modificado no anel 1; que é chamado de *dom0*. Neste domínio é possível criar outros domínios, destruí-los, executar operações de migração de domínios virtuais entre máquinas reais, definir parâmetros de um domínio, aumentando e diminuindo dinamicamente a alocação de memória, entre outras funções. Os domínios criados também executam esses *kernel*s no anel 1.

3.1.1 O servidor XCS

O Xen possui um servidor responsável por realizar toda a comunicação entre as máquinas dos sistemas operacionais convidados com a máquina do *dom0*. Este servidor recebe o nome de XCS[2].

O servidor XCS abre duas portas TCP, uma para o controle da conexão e outra para o controle dos dados. O controle de conexão é síncrono enquanto o controle dos dados assíncrono. Essas portas são responsáveis por todo o tráfego de requisições entre duas máquinas virtuais,

[2] Acrônimo para *Xen Control Server*.

com o sistema que possui o *kernel* modificado do Xen. Qualquer problema ou gargalo nesse componente é um ponto crítico do sistema e pode afetar toda a comunicação entre as máquinas virtuais.

Uma conexão ao servidor XCS é representada por um objeto. Depois que uma conexão é ativada, ela é adicionada à lista de conexões e passa por um processo de iteração a cada cinco segundos para ver se chegaram novas mensagens de controle ou de dados. Controlar as mensagens, que podem ser de controle ou de dados, é tarefa das *funções handle_control_message()* ou do *handle_data_message()*, respectivamente [46].

3.1.2 O *daemon* Xend

O *daemon* Xend é o controlador do Xen responsável por criar novos domínios, destruir domínios, realizar *live migration* e outras tarefas de gerenciamento. De acordo com [46] grande parte de seu funcionamento está baseado em executar um servidor HTTP. A porta padrão do soquete HTTP é a 8000, que pode ser configurada. Os vários pedidos para controlar os domínios são gerenciados através de pedidos HTTP para criar domínios, desligar, salvar, restaurar, realizar migração de domínios e outras funcionalidades. Uma grande parte do código do Xend é escrita na linguagem *Phyton* e usa chamadas aos métodos da linguagem C dentro de *scripts Phyton*.

Após carregar o *kernel* modificado do Xen, o Xend se inicia. O trabalho do Xend *daemon* é baseado na interação com o servidor XCS, que funciona semelhante a um *switch* de controle. Quando o Xend *daemon* é iniciado, ele verifica se o servidor XCS está ativo e em execução, caso contrário tentará iniciar o XCS.

3.1.3 Escalonamento da CPU

As primeiras versões do Xen escalonavam domínios de acordo com os algoritmos *Round Robin*, *Atropos* e *Borrowed Virtual Time* (BVT) [22]. O algoritmo *Round Robin* é um dos algoritmos mais antigos, simples e justo, que apesar de eficiente era pobre nos parâmetros de configuração, permitindo ao usuário somente escolher o tempo máximo que cada domínio poderia executar, antes que a próxima decisão do escalonador fosse tomada. O algoritmo *Atropos* permite ao usuário reservar CPU para trabalhar exclusivamente com determinado domínio, além de prover garantias de tempo para os domínios que são sensíveis à latência. O escalonador BVT divide o tempo de processamento em modo proporcional. Apesar deste método penalizar domínios que realizam muitas operações de E/S, ele possui um mecanismo de compensação que tenta reduzir esse efeito.

Versões mais recentes do Xen utilizam o algoritmo de escalonamento *Credit* [2] o qual

introduz o conceito de Virtual CPU (VCPU). Esse algoritmo gerencia uma fila local de VCPUs em cada máquina virtual, ordenada pela prioridade de cada uma delas, que pode ter somente dois valores: acima ou abaixo. Enquanto uma VCPU está em execução ela consome créditos. Os créditos negativos implicam em uma prioridade acima, representando que a máquina virtual ou VCPU teve um razoável tempo de processamento. Enquanto uma VCPU não consumir os créditos que lhe foram alocados, sua prioridade estará abaixo.

3.1.4 Tradução de endereço virtual

Como ocorre nos seus outros subsistemas, o Xen tenta virtualizar o acesso à memória com o menor *overhead* possível. O *hypervisor* é o responsável por acessar as tabelas de páginas virtuais, validar atualizações e propagar mudanças [7].

A virtualização completa força o uso de tabelas de página *shadow*, a qual consiste em manter duas tabelas de páginas durante toda a vida de um domínio. Mantém assim a tabela de páginas atuais e a tabela de página *shadow*. Esta é uma mera cópia guardada durante a execução dos processos, a qual somente é utilizada para restaurar a tabela de páginas atuais, se ocorrer alguma falha, caso contrário é descartada.

O Xen trabalha de forma que somente precisa ser comunicado das atualizações das tabelas de páginas, para impedir que sistemas operacionais convidados façam mudanças inaceitáveis. A abordagem do Xen é registrar tabelas de páginas de sistemas operacionais convidados diretamente com o *Memory Management Unit* (MMU) e restringir aos sistemas operacionais convidados acesso somente leitura. As atualizações de tabelas de páginas são passadas para o Xen via *hypercall*[3]. Para aumentar a segurança os pedidos são validados antes de serem aplicados [7].

3.1.5 Acesso ao disco

Somente o *dom0* tem acesso direto ao disco rígido físico, todos os outros domínios acessam o disco através de uma abstração de *Virtual Block Devices* (VBD) . Os VBDs são gerenciados pelo *dom0*, mantendo um mecanismo muito simples de gerência de disco no Xen.

Um VBD inclui uma lista de extensões com associações de proprietários dos arquivos e controle do acesso das informações. O algoritmo de escalonamento do sistema operacional convidado pode reordenar as requisições com prioridade maior para enfileirar no anel de transmissão, em uma tentativa de reduzir o tempo de resposta ou de fornecer o serviço de forma diferenciada [7].

Uma tabela de tradução é mantida dentro do *hypervisor* para o VBD de cada sistema ope-

[3]Hypercall é uma chamada de sistema do hypervisor do Xen para um sistema operacional.

racional convidado. As entradas desta tabela são instaladas e controladas pelo *dom0* através de uma interface de controle privilegiada. Ao receber uma requisição do disco, o Xen inspeciona o identificador e o *offset* do VBD e produz o endereço correspondente do setor e do dispositivo físico. Nesse momento também ocorrem as verificações de permissão, mantendo assim a integridade e o isolamento de cada *domU*.

Às máquinas virtuais compete o acesso ao disco em um algoritmo simples de *round robin*. As suas requisições são passadas para um escalonador antes de alcançarem o disco.

3.1.6 Ambiente de rede no Xen

O Xen oferece a abstração de um *Virtual Firewall Router*(VFR), onde cada domínio tem uma ou mais *Virtual Interfaces* (VIFs), logicamente unida ao VFR. Uma VIF possui uma moderna interface de rede, permitindo enviar e receber dados. Em cada sentido de envio/recebimento existe uma lista de regras associadas na forma (<padrão>, <ação>), onde se o padrão combinar com a regra associada, a ação é aplicada [7].

O *dom0* é o responsável por inserir e remover as regras de envio e recebimento de pacotes, realizando um complexo roteamento entre as máquinas virtuais. Em casos típicos, regras podem ser inseridas para prevenir mascaramento de IP e assegurar desmultiplexação baseados no endereço IP e porta de destino. As regras também podem ser associadas com as interfaces de *hardware* do VFR, permitindo instalar regras para executar funções tradicionais de *firewall*, como por exemplo prevenir tentativas de conexão em portas inseguras, detectar IP *spoofing*, etc.

Para transmitir um pacote, o sistema operacional convidado envia uma descrição do *buffer* no anel de transmissão. O Xen copia a descrição e, para aumentar a segurança, copia o cabeçalho do pacote e executa todas as regras iguais ao filtro que foi definido [7]. Quando são transmitidos vários pacotes, o Xen implementa um simples escalonador *round-robin*. Quando um pacote é recebido, imediatamente verifica-se o conjunto de regras recebidas para determinar o VIF de destino.

O Xen virtualiza em cada domínio uma interface de rede conhecida como *eth0*. No *dom0* o Xen cria uma ponte que no sistema Linux recebe o nome de *xenbr0*, cuja finalidade é enviar e receber pacotes para as VIFs. São criadas também uma série de interfaces virtuais, uma para cara máquina virtual, as quais são mostradas na Figura 4.

No exemplo da Figura 4, cada *domU* utiliza uma interface virtual do dispositivo *eth0*. O Xen utiliza uma abordagem que permite com que o usuário acesse o dispositivo de rede da mesma maneira com que acessa uma máquina real. As VIFs presentes no dispositivo virtual *xenbr0* são utilizadas para rotear e encaminhar pacotes para fora da rede local [23].

Para o Xen ser instalado, ele exige a instalação prévia do pacote *bridge-utils*, o qual possui todos os utilitários necessários para ativar, configurar e monitorar o funcionamento de uma interface de rede em forma de ponte. As principais conexões de pontes criadas no Xen são:

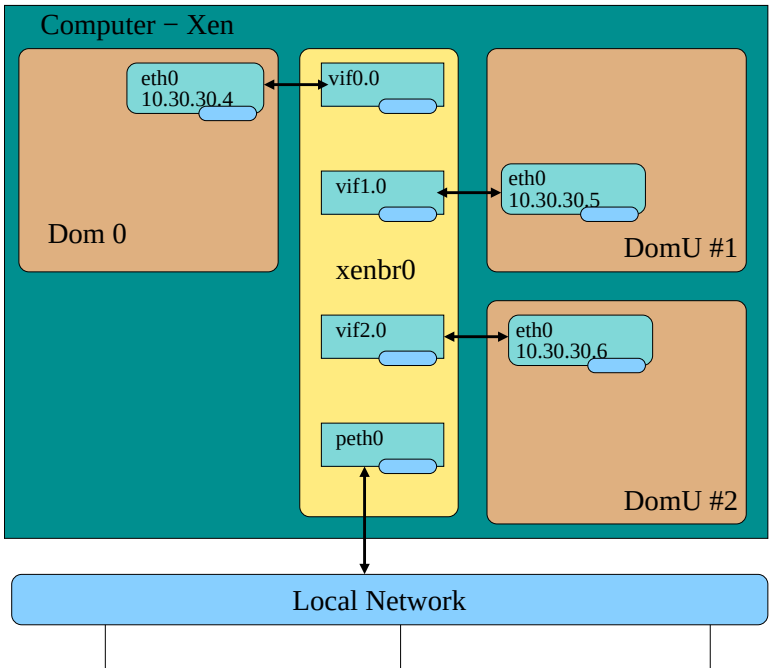

Figura 4 – Máquinas virtuais utilizando VIFs para roteamento de pacotes.

peth0 Porta que conecta a interface de rede física ao sistema;

vif0.0 Ponte utilizada para rotear os pacotes de/para o *dom0*;

vifX.0 Ponte utilizada para rotear os pacotes de/para o domX.

3.2 Ferramentas de gerenciamento

Quando é realizada a primeira instalação do Xen, a única opção disponível para criar e destruir máquinas virtuais, alterar alocação de memória e outras opções de gerenciamento é através da ferramenta *xm*, original do inglês *Xen Manager*.

Pelo fato do *xm* ser totalmente em modo texto, tornava-se difícil para os usuários mais leigos gerenciar ambientes com muitas máquinas virtuais e acompanhar simultaneamente as máquinas ativas, tal qual ocorre com o VMware. Foi para suprir essa carência que desenvolvedores independentes criaram ferramentas em modo gráfico para gerência das máquinas virtuais do Xen. As principais soluções para gerenciamento em modo gráfico são: *XenMan* [56], *Enomalism* [15] e *Virtual Machine Manager* [26].

Uma das primeiras soluções desenvolvidas em modo gráfico para facilitar a gerência de máquinas virtuais foi o XenMan [56], surgindo como opção no mercado em meados de 2006. Ele utiliza um túnel SSH (*Secure Shell*) para realizar comunicação com as máquinas virtuais remotas e permite o gerenciamento de múltiplos servidores *dom0* e *domU*. A Figura 5 exibe o seu funcionamento, permitindo ao usuário realizar o acompanhamento da quantidade de CPU e

memória que está sendo consumida por cada servidor.

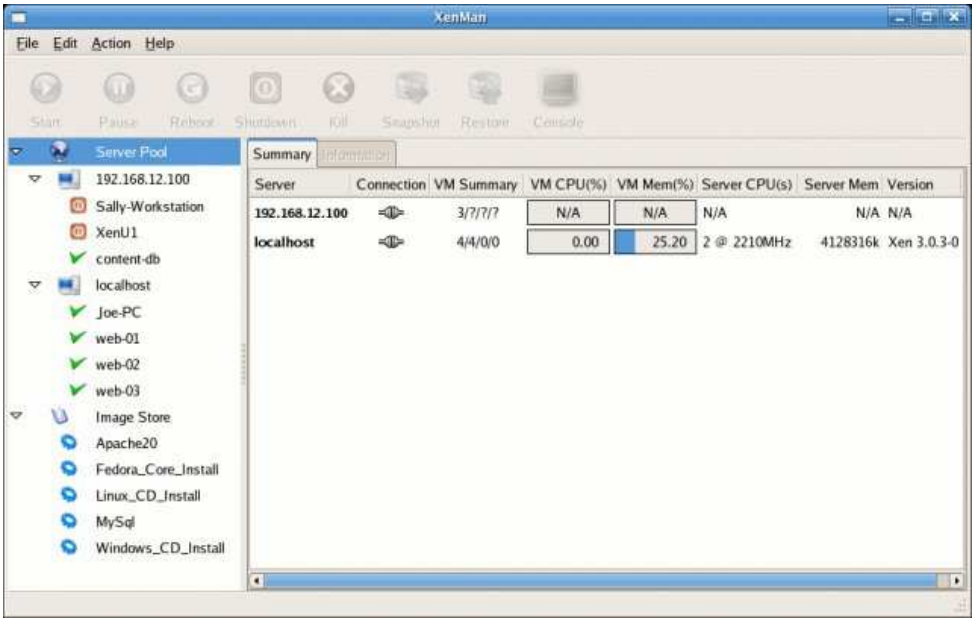

Figura 5 – *XenMan* em funcionamento.

Não menos importante que o *XenMan*, há outra solução gráfica muito difundida para gerência de máquinas virtuais baseadas em Xen, trata-se do *Virtual Machine Manager* [26]. Desenvolvido pela Red Hat, é bastante difícil instalar em outras distribuições, devido a várias incompatibilidades. A instalação só é bem sucedida através da aplicação de vários *patches*. Pelo fato do *Virtual Machine Manager* utilizar chamadas da API *libvirt* [38], a mesma é pré-requisito para a instalação. A Figura 6 ilustra o funcionamento da ferramenta, exibindo uma máquina *dom0* e 3 máquinas *domU*, todas em execução a partir de uma mesma máquina física.

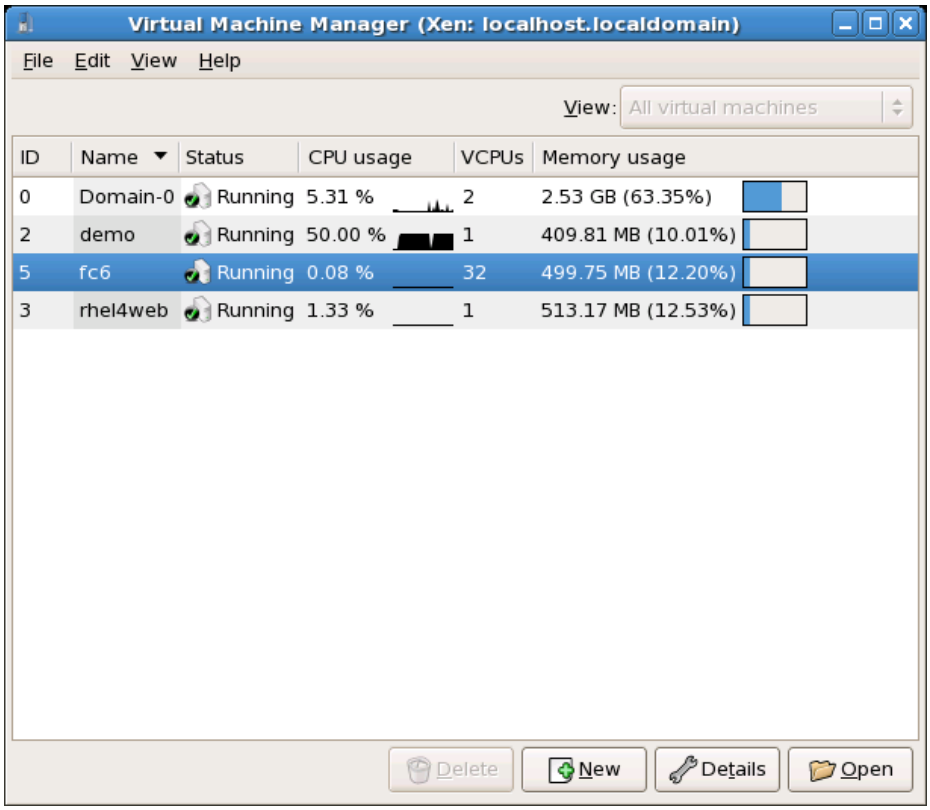

Figura 6 – *Virtual Machine Manager* em funcionamento.

3.3 Conclusão do capítulo

Este capítulo apresentou a arquitetura do Monitor de Máquina Virtual Xen, uma solução gratuita e de excelente desempenho para virtualizar servidores baseados no sistema operacional Linux (Windows funciona somente como sistema operacional convidado - *domU*). Além dos motivos citados anteriormente, a motivação extra para o uso desta solução dá-se pelo fato do Xen possuir código fonte aberto, permitindo entender como ocorre o seu funcionamento interno e como é realizada a comunicação entre os seus sistemas operacionais convidados.

O destino do Xen ainda é incerto. Inicialmente desenvolvido como um projeto de pesquisa na Universidade de Cambridge e liderado por Ian Pratt [44], fundador da XenSource Inc. a qual foi adquirida pela Citrix Systems em outubro de 2007. Pelo fato de surgir através de uma pesquisa acadêmica, o Xen possuía forte apoio das maiores empresas de computação do mundo. Com a aquisição por parte da Citrix, ainda é incerto se esse apoio continuará pelos próximos anos. A nova versão do Xen, 3.2.0 é aguardada para início de 2008, já se encontrando em fase de *release candidate*[4].

[4] *Release candidate* é uma das fases do ciclo de desenvolvimento de software. Após as fases *alpha* e *beta*, esta fase é a que possui maior potencial para ser a versão final do software.

4 Detecção de intrusos

Este capítulo apresenta a área de detecção de intrusos, mencionando vantagens e desvantagens encontradas. Em um segundo momento é apresentado o *Snort*, uma solução para detectar intrusos em rede. Este foi adaptado para trabalhar juntamente com *honeypots*, objetivando reforçar a segurança do ambiente computacional se ambos estiverem trabalhando em conjunto.

De maneira abstrata, um intruso pode ser definido como alguém que tenta invadir um sistema ou fazer mal uso do mesmo. Entretanto, é muito difícil definir o que significa fazer mal uso de um sistema, ou ainda, definir quais usuários são classificados como intrusos. Um usuário legítimo que tenta acessar um sistema por três ou quatro vezes consecutivas e erra a senha em todas as tentativas, pode ser classificado como um invasor pelo sistema de detecção de intrusos.

Devido à insegurança inerente do protocolo TCP/IP [45], torna-se fundamental o desenvolvimento de ferramentas de *scanners* de rede, *sniffers* e outras ferramentas de auditoria e detecção de intrusos, objetivando prevenir brechas de segurança que por ventura possam existir. Sistemas de detecção de intrusos baseados em rede (*Network Intrusion Detection Systems* ou simplesmente NIDSs) tornaram-se popular com o avanço da Internet em tamanho e tráfego.

NIDS [42] é um *software* que tenta detectar pacotes estranhos e evitar que eles comprometam a segurança de uma rede [12]. Um NIDS trabalha em modo promíscuo, permitindo que seja capturado todos os pacotes que passam pela sua interface de rede, mesmo aqueles que são descartados pelo sistema operacional e não somente pacotes endereçados a um endereço IP em particular. Sistemas detectores de intrusos podem analisar todos os pacotes que circulam por uma rede e verificar se algum deles contém *strings* suspeitas. Caso haja algum pacote suspeito o NIDS decide qual ação será adotada, baseada em níveis de prioridade, onde um nível de prioridade maior requer uma ação mais drástica, como por exemplo os bloqueios de portas específicas de um determinado endereço de destino, ou ainda o bloqueio total de pacotes, de acordo com a severidade da intrusão.

Quando um sistema sofre um ataque, o NIDS pode interagir com o *firewall* do sistema para criar novas regras com a finalidade de bloquear o endereço IP que está tentando acessar indevidamente o sistema. É possível ainda que o administrador receba via *pager*, telefone móvel ou e-mail, alertas de segurança quando um sistema sofrer algum tipo de tentativa de ataque.

4.1 Problemas de um NIDS

Como toda ferramenta de proteção de sistemas, além de uma série de vantagens, os NIDS também apresentam alguns aspectos desfavoráveis à sua utilização, ou ainda alguns pontos críticos onde não garantem toda a proteção a qual se espera.

Os atacantes estão utilizando *exploits* cada vez mais sofisticados e diversificados, bem como novos cenários de ataques mais complexos, o que torna difícil manter um sistema de detecção de intrusos com as assinaturas 100% atualizadas. Ataques do tipo *0-day* exploram vulnerabilidades que ainda não possuem *patch*[1] no mercado. Outro problema que impacta diretamente na diminuição da eficiência de um NIDS é a utilização de mensagens criptografadas, principalmente utilizadas pelos protocolos HTTPS, SSL e TLS. Se um atacante inserir algum tipo de ataque em um protocolo de transporte criptografado a detecção fica impossibilitada.

Em redes com grande volume de dados, alguns pacotes podem passar despercebidos pelo fato do NIDS não conseguir dar conta de coletar e processar simultaneamente todo o tráfego recebido. Sistemas detectores de intrusos podem trazer riscos ao tomar ações de resposta automáticas, servidores importantes em uma rede local podem ser bloqueados por engano e causar sérios transtornos.

Alguns NIDS possuem um número relativamente grande de falsos-positivos e falsos-negativos, o que torna difícil diagnosticar se realmente o sistema está sinalizando uma ocorrência verdadeira. Essa situação remete ao parágrafo anterior, onde foi descrito que, se um ataque for detectado indevidamente, uma máquina poderá ser bloqueada de forma incorreta.

4.2 Snort

Snort [50] é um NIDS largamente utilizado para realizar auditoria de pacotes em rede e compará-los com um banco de dados de assinaturas de ataques conhecidos. Consegue realizar análise de tráfego em tempo real e detecta uma variedade de ataques, como: *buffers overflows*, *ataques CGI*, *stealth port scanners*, *SMB probes*, *OS fingerprinting*, além de outras variações. Possui suporte a vários sistemas operacionais, tais quais: praticamente todas as variações de Linux, OpenBSD, FreeBSD, NetBSD, Solaris, SunOS, HP-UX, AIX, IRIX, Tru64, MacOS X.

Por ser uma ferramenta de detecção leve e rápida consome poucos recursos do sistema. As assinaturas de ataques podem ser criadas facilmente através de explorações contra o próprio sistema enquanto é executado um *sniffer*, que realizará a captura do texto ou *string* binária utilizada pela ferramenta de ataque contra o seu próprio servidor. Os caracteres significativos desta *string* são adicionados ao descritor de conteúdo do *Snort* fazendo com que respostas a

[1] *Patch* é uma correção via *software*, disponibilizada normalmente pelo fabricante do aplicativo, podem acrescentar novas funcionalidades ao sistema ou apenas corrigir erros no código lançado anteriormente.

ataques sejam retornadas através de *logs*. No entanto, quando há um ataque de maior gravidade é possível emitir alertas por e-mail.

Para o funcionamento adequado do *Snort*, faz-se necessária a criação de regras, pois serão através delas que o sensor terá conhecimento de quais serviços, portas ou fluxos de dados são prejudiciais ao ambiente de rede. São regras que decidem o processamento que um pacote terá, qual decisão será tomada, qual será o novo caminho do pacote e se ele será aceito ou rejeitado pelo sistema.

A Figura 7 nos mostra um exemplo de uma regra extraída do banco de dados do *Snort*, onde o sistema será notificado quando houver suspeita de ataque do tipo *Denial of Service* (DOS), utilizando o protocolo de gerenciamento de grupo *Internet Group Management Protocol* (IGMP).

```
alert ip $EXTERNAL_NET any -> $HOME_NET any
(msg:"DOS IGMP dos attack"; fragbits :M+; ip_proto:2; reference:bugtraq,514;
reference:cve,1999-0918; classtype:attempted-dos; sid:272; rev:9;)
```

Figura 7 – Exemplo de regra do Snort

Na Figura 7 as variáveis *$EXTERNAL_NET any -> $HOME_NET any* fazem parte do cabeçalho da regra e servem para sinalizar que qualquer pacote que entrar na rede interna, em qualquer uma das portas, faz parte da regra. O próximo passo é definir as opções da regra, geralmente iniciadas por palavras chaves do tipo *"msg"* ou *"content"* e em seguida são definidas as peculiaridades de cada tipo de ataque. No exemplo da Figura 7 foi utilizada a palavra chave *fragbits*, cuja finalidade é testar se existe fragmentação de bits no cabeçalho IP, o que sinaliza um possível ataque do tipo *Denial Of Service*.

Quando um pacote chegar na rede e combinar com a regra do *Snort*, é gerado um alerta para o sistema. Este alerta pode ser armazenado em um banco de dados, tal qual o*Analysis Consoles for Intrusion Detection* (ACID), BASE e *SnortView* [4, 8, 34] ou ainda pode ser armazenado no formato *plain text*. A Figura 8 apresenta um exemplo do log de saída do *Snort* para um ataque do tipo de negação de serviço.

```
[**] [1:272:9] DOS IGMP dos attack [**]
[Classification: Attempted Denial of Service] [Priority: 2]
08/15-17:35:33.019334 192.168.100.201 -> 192.168.100.234
IGMP TTL:64 TOS:0x0 ID:5723 IpLen:20
DgmLen:1500 MF Frag Offset: 0x02E4 Frag Size: 0x05C8
```

Figura 8 – Exemplo de ataque do tipo negação de serviço.

A assinatura da Figura 9 aponta para um possível "*Bad Traffic*" e exibe o site do CERT[2] como referência para consultar a vulnerabilidade que o invasor tentou explorar.

[2]CERT é o acrônimo para Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança. A entidade é responsável por receber, analisar e responder incidentes de segurança envolvendo redes de computadores.

```
[**] [1:527:8] BAD-TRAFFIC same SRC/DST [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/14-11:15:22.471873 255.255.255.255 -> 255.255.255.255
PROTO139 TTL:255 TOS:0x0 ID:1 IpLen:20 DgmLen:50
[Xref => http://www.cert.org/advisories/CA-1997-28.html][Xref => http://cve.mitre.org/cgi-
bin/cvename.cgi?name=1999-0016][Xref => http://www.securityfocus.com/bid/2666]
```

Figura 9 – Exemplo de ataque do tipo *Bad Traffic*.

4.2.1 Arquitetura do *Snort*

O *Snort* foi implementado de acordo com uma arquitetura modular com foco na otimização do desempenho, na coleta e análise de pacotes. Existe o conceito de subsistemas onde cada qual utiliza um *plug-in* correspondente para desempenhar determinada tarefa:

- **Pré-processamento:** Fica localizado entre o *packet sniffing* e o processamento do *engine* de detecção. Sua função principal é realizar a decodificação dos pacotes;
- **Detecção:** Subsistema que entra em operação durante o processamento do *engine* de detecção;
- **Saída:** É o último processo a ser executado, após o processamento do engine de detecção. Sua função é registrar as ocorrências encontradas e alertar o usuário.

A Figura 10 mostra os principais módulos de análise e diagnóstico do *Snort*. O Decodificador recebe os pacotes da interface de rede e os analisa no nível dos protocolos e dos cabeçalhos, encontrando IPs, portas, tamanho do pacote e sua integridade, tudo isso é feito antes deles serem remontados para uma aplicação.

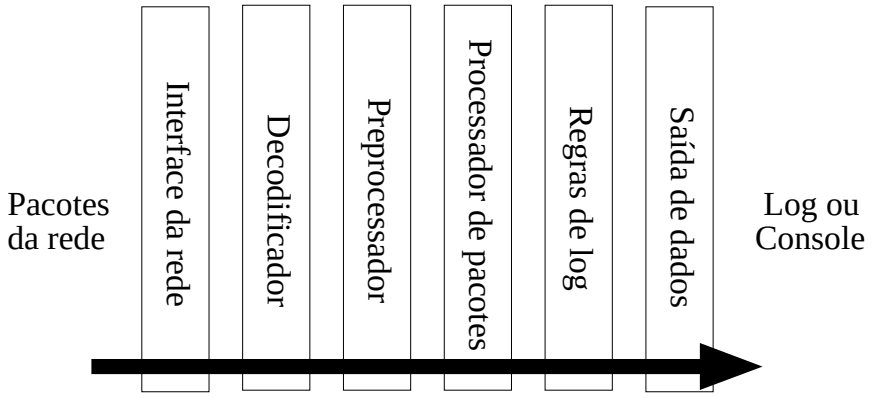

Figura 10 – Arquitetura do *Snort*

O preprocessador analisa os pacotes com comportamento suspeito e remete para a análise de conteúdo no módulo seguinte. O processador de pacotes realiza a comparação do conteúdo e natureza de cada pacote com as assinaturas de ataques conhecidos. Caso seja encontrado algum pacote suspeito, o mesmo pode ser registrado em um arquivo de *log* ou impresso no console.

4.3 Modos de funcionamento

O *Snort* ao contrário do que pode parecer, não funciona exclusivamente como um sistema de detector de intrusos. Por fazer uso de uma poderosa biblioteca de captura de pacotes, a *libpcap*. Existem modos de operação do *Snort* que simplesmente capturam pacotes de forma semelhante ao *tcpdump*[3] e trazem a opção de gerar arquivos de log com os pacotes capturados, ou ainda exibir no console do usuário. De acordo com a documentação oficial [50], o *Snort* possui pelo menos quatro modos de operação:

- **Modo *sniffer*:** neste modo o *Snort* somente lê os pacotes capturados na rede e exibe no console em tempo real;
- **Modo *logger* de pacotes:** neste modo o *Snort* armazena no disco os pacotes lidos, ao invés de apresentar no console;
- **Modo detector de intrusão:** é o modo em que o *Snort* é mais utilizado. Ao operar neste modo, os pacotes lidos não são apresentados no console ou armazenados no disco, mas cada pacote é analisado e interpretado segundo as assinaturas definidas e decisões são tomadas de acordo com as regras pré-estabelecidas. É o modo completo do *Snort*;
- **Modo *inline*:** para operar neste modo a biblioteca de captura de pacotes *libpcap* não é utilizada diretamente. O *Snort* faz a leitura dos pacotes capturados pelo *iptables* para automatizar decisões de descartar ou autorizar pacotes.

4.4 Outras ferramentas

Apesar do *Snort* ser uma grande ferramenta para detecção de intrusos, ele possui algumas carências. Pelo fato de funcionar unicamente como detector de intrusos, não realiza procedimentos extras para barrar o atacante quando ocorre alguma intrusão. Os registros de *log* da invasão são criados, o administrador do sistema é comunicado do ocorrido, mas o atacante pode persistir fazendo mau uso do sistema.

Para contornar essas limitações do *Snort*, usuários independentes estão criando ferramentas para auxiliar o uso do *Snort*, principalmente nos módulos de maior deficiência, que são o não muito eficiente e desorganizado sistema de *logging*, e o fato de não existir nenhuma ação concreta para impedir um atacante de continuar acessando indevidamente o recurso da rede.

O *Snort* atua como forma passiva, medidas de segurança não são tomadas a partir dos resultados dos logs. O *XenGuardian* é uma ferramenta criada para expandir o *Snort* para ambientes

[3] *Tcpdump* é uma poderosa ferramenta para *sniffer*, disponível em sistemas Linux.

virtualizados. Existe também outras três ferramentas famosas e muito utilizadas em conjunto com o *Snort*.

4.4.1 Barnyard

Barnyard é um pós-processador que trabalha através do processamento dos logs gerados pelo *Snort*. Ele oferece suporte a praticamente todos os *plugins* de saída do *Snort*, desde formato ASCII, XML até *log* em banco de dados. É possível configurá-lo para operar em modo contínuo ou fazer chamadas esporádicas a qualquer momento que necessário.

A grande vantagem da utilização do *barnyard* é que o *Snort* faz a análise de somente um pacote por vez. Se ele não for utilizado, a cada operação de escrita diretamente no banco de dados que é realizada, os novos pacotes não serão analisados até que haja a confirmação de sucesso da operação de escrita. Redes com grande tráfego podem ter pacotes que passem despercebidos pelo *Snort*, decrementando a confiabilidade do programa e se o banco de dados cair, acontecerá o mesmo com o *Snort*.

O diagrama da Figura 11 ilustra o funcionamento do *Barnyard*, é possível perceber que o *Snort* fica livre para realizar a análise dos pacotes enquanto que o trabalho de ler os registros de alertas e organizá-los em um banco de dados fica a cargo do *Barnyard*.

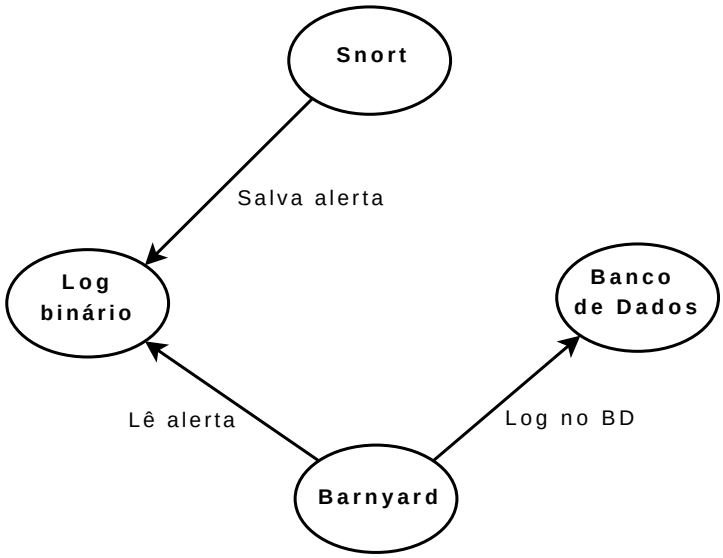

Figura 11 – Diagrama de funcionamento do Barnyard

4.4.2 Oinkmaster

O *Oinkmaster* é uma ferramenta baseada em *scripts Perl* que facilita o gerenciamento das regras utilizadas pelo *Snort*. Ele tem a capacidade de fazer o download automaticamente do

arquivo de regras mais atualizado do *Snort*, fazer backup das regras antigas e instalar as novas no sistema. Funciona tanto em linha de comando, como também através de uma interface gráfica.

Pode se tornar muito útil quando existem vários sensores instalados em um ambiente distribuído. Todos são atualizados com as mesmas regras através do *download* de um site principal ou múltiplas localizações ao mesmo tempo.

4.4.3 Guardian

Guardian é um programa que trabalha em conjunto com o *Snort* para atualizar automaticamente as regras de *firewall*, em conformidade com os alertas que são gerados pelo sistema detector de intrusos. As regras de *firewall* que são geradas bloqueiam os dados dos endereços IPs das máquinas que estão realizando ataques, de acordo com os diagnósticos de alertas. Para evitar que uma máquina seja bloqueada indevidamente, há recursos do próprio *iptables* [4] para prevenir que máquinas amigas ou importantes na rede sejam bloqueadas.

Do mesmo modo que o *Oinkmaster*, o *Guardian* também é desenvolvido utilizando scripts em *Perl*. Além da compatibilidade com o *iptables*, é compatível com outros tipos de *firewalls*, tais quais o *ipchains* e *ipfw* [5], pode ainda ser compatível com outras soluções de *firewalls* desde que sejam realizados os devidos ajustes na ferramenta.

4.5 Honeypot como ferramenta de segurança

Honeypot [6] é uma ferramenta de estudo de segurança, onde sua função principal é capturar informações do atacante. Um *honeypot* consiste em um sistema que possui falhas de segurança propositais, permitindo que seja atacado e comprometido (invadido) e o fruto desta invasão possa ser estudado.

O principal propósito para criar um sistema honeypot é identificar *port scanners* e ataques automatizados, pelo fato de permitir a simulação de processos e abertura de portas específicas [48]. Um honeypot além de permitir a simulação de ambientes baseados nas plataformas Windows e/ou Linux é capaz de simular *uptime* de uma máquina, utilizando assim um subsídio a mais para enganar pessoas de má fé que tentam fazer mal uso do sistema.

Os principais tipos de *honeypots* existentes na atualidade são [48]:

- ***Honeypots* de pesquisa:** são utilizados para observar as ações de atacantes ou invasores, permitindo analisar detalhadamente as ferramentas utilizadas e as vulnerabilidades

[4] *Iptables* é um simples e poderoso *firewall* orientado via linha de comando e largamente utilizado em sistemas operacionais Linux, utiliza o módulo *netfilter*.

[5] O *ipfw* ou *ipfirewall* é o *firewall* padrão do sistema operacional FreeBSD.

exploradas;

- ***Honeypots* de produção:** utilizados em redes de produção como complemento da segurança, ou ainda, utilizados no lugar de sistemas de detecção de intrusão;
- **Baixa interatividade:** apenas emulam serviços e sistemas operacionais. Os atacantes não tem acesso à máquina real, sendo apropriados para *honeypots* de produção, uma vez que são um excelente adicional para sistemas de detecção de intrusos. Pelo fato do *honeypot* ter capacidades mínimas, fica reduzido o risco de um atacante comprometê-lo e lançar ataques nos outros recursos da rede. Justamente pela sua simplicidade e interação limitada é mais fácil para um atacante perceber que está acessando um *honeypot*;
- **Alta interatividade:** utilizam serviços legítimos, permitindo que o atacante assuma controle total do *honeypot*. Coletam metodologias de ataques, análises de ferramentas utilizadas e ataques do tipo *0-day* que exploram vulnerabilidades descobertas recentemente. É necessário cautela neste tipo de *honeypot*, evitando que seja acessado indevidamente por um usuário mal intencionado com intenção de realizar ataques em outros sistemas.

A Figura 12 ilustra o uso de um sistema de *honeypot* no ambiente do *XenGuardian*, onde coexistem máquinas virtuais e máquinas virtuais honeypots. Pelo fato de não fazerem parte de um sistema de informação da organização onde está instalado, todo o tráfego de um *honeypot* é malicioso, sem a existência de falso-positivo[6].

4.5.1 Honeynet

Honeynet é um tipo de *honeypot* de alta interatividade que objetiva pesquisar e obter informações de invasores. Normalmente pertence a um segmento de rede onde estão presentes *honeypots* de diversos sistemas operacionais oferecendo variadas aplicações e serviços. Com o advento das máquinas virtuais, as *honeynets* foram divididas em duas categorias: *honeynets* reais e *honeynets* virtuais:

Honeynets Reais

Os dispositivos que compõe uma *honeynet* real, incluindo os mecanismos de proteção, de alerta e coleta de informações, os diversos *honeypots*, são todos físicos. Uma única máquina executa um único *honeypot* com um sistema operacional real juntamente com as suas aplicações e serviços [28].

Uma *honeynet* real ainda precisa de um computador com *firewall* para atuar como mecanismo de contenção e coleta de dados, além de outra máquina física com sistema de detecção

[6]Falso-positivo é uma expressão utilizada para designar uma situação em que um *firewall*, *honeypot* ou IDS detecta uma atividade como sendo um ataque, quando na verdade é uma ação legítima.

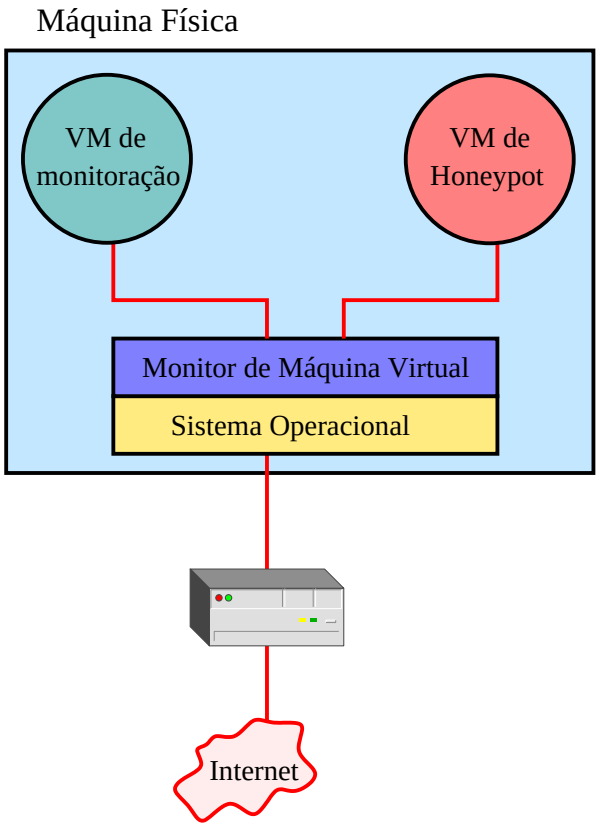

Figura 12 – *Honeypot* em um monitor de máquina virtual.

de intrusos para atuar como mecanismo de geração de alertas e coletas de informações sobre ataques. Além de toda essa estrutura, faz-se necessário uma máquina para servir como repositório dos dados coletados e não menos importante switches/hubs, roteadores e cabos necessários para interligar toda a estrutura que está sendo criada. A Figura 13 ilustra o exemplo de uma *honeynet* constituída de três *honeypots*.

A grande vantagem deste tipo de *honeynet* é a tolerância a falhas, pois o ambiente é distribuído. Outro ponto forte é a interação dos atacantes com o ambiente real, tornando maior a dificuldade de percepção que o ambiente explorado. Como desvantagem tem-se um maior espaço físico para os equipamentos, além de consumir mais recursos de infra-estrutura torna a manutenção trabalhosa e difícil, elevando o custo total de implementação da *honeynet*.

Honeynets Virtuais

Uma *honeynet* virtual como o próprio nome já sugere, parte do princípio que todos os componentes da *honeynet* são implementados em um reduzido número de dispositivos físicos. Para este propósito, é necessário um sistema operacional hóspede para executar o software de virtualização, podendo ser o VMware [53], UML [21] ou ainda o Xen [44]. Uma máquina virtual permitirá executar múltiplas instâncias de sistemas operacionais que estarão centralizados em uma única máquina física.

As *honeynets* virtuais se subdividem em duas categorias: auto-contenção e híbridas [28].

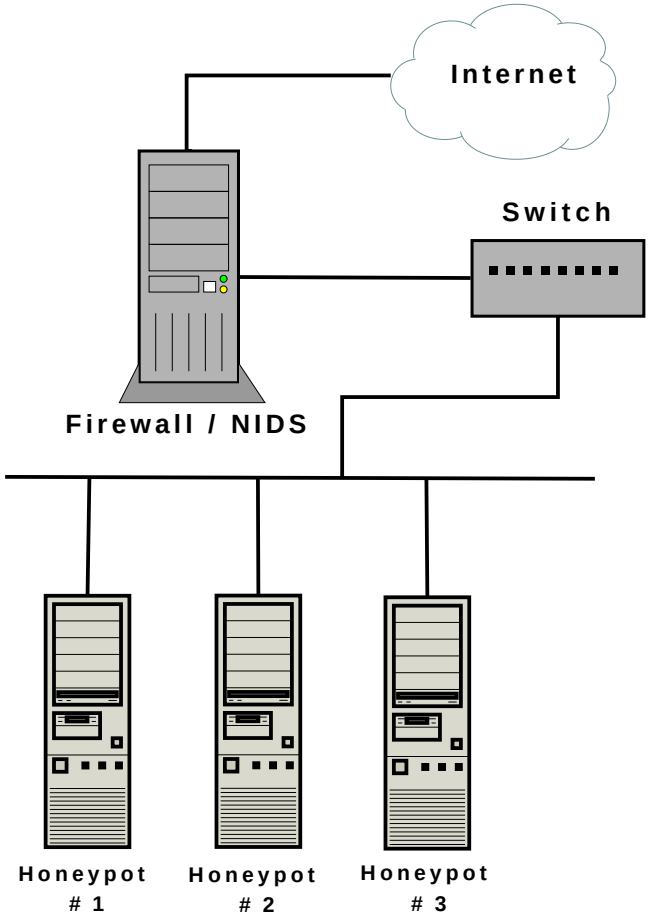

Figura 13 – Exemplo de uma *honeynet* real.

No primeiro caso os mecanismos de contenção, captura e coleta de dados, geração de alertas e os diversos *honeypots* são centralizados em um único computador através da virtualização. Nas híbridas os mecanismos de contenção, captura e coleta de dados e geração de alertas estão em dispositivos distintos, enquanto que os *honeypots* permanecem em um único computador utilizando virtualização.

A vantagem dessa solução é a manutenção de forma mais simples, pois os recursos tendem a ficar centralizados em um número reduzido de dispositivos físicos, o que também poupa espaço físico para equipamentos e reduz o custo com manutenção e consumo de eletricidade. A Figura 14 demonstra como esse ambiente está constituído com a utilização do monitor de máquina virtual Xen.

Como desvantagens podem ser mencionados os altos custos para cada dispositivo já que a virtualização requer máquinas mais robustas. O fato de existirem várias *honeypots* em uma única máquina física cria uma desvantagem no quesito tolerância a falhas, pois concentra todos os dispositivos em um mesmo recurso. Em alguns casos o *software* de virtualização poderá limitar o *hardware* e os sistemas operacionais utilizados, eliminando alguns recursos fundamentais dos sistemas de honeypots. Ainda há o risco do *software* de virtualização sofrer alguma falha que permita o atacante acessar outras partes do sistema, pois os recursos de um mesmo dis-

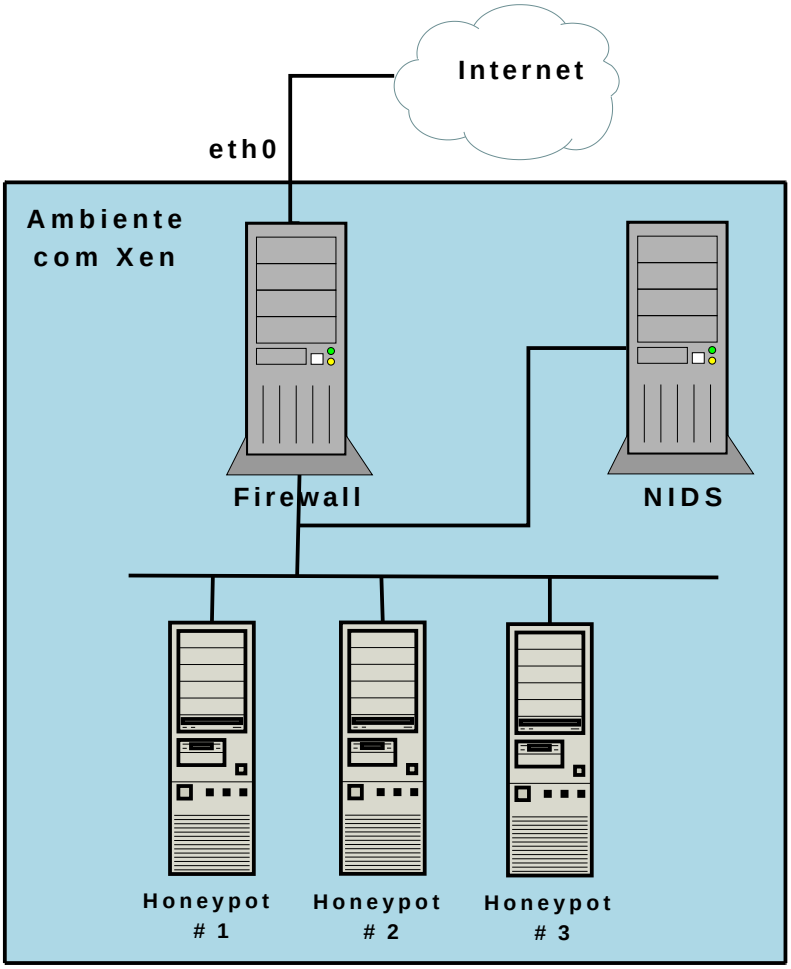

Figura 14 – Exemplo de uma *honeynet* virtual.

positivo são compartilhados. Na virtualização uma placa de rede pode estar em uso por dois sistemas operacionais distintos ao mesmo tempo. Além disso, existe o risco do atacante descobrir que está interagindo com um ambiente virtual, o deixando furioso por ter sido enganado e muitas vezes podendo motivá-lo a derrubar o sistema a qualquer custo.

4.5.2 Honeyperl

Quando o *XenGuadian* detecta uma tentativa de ataque, o atacante pode ser redirecionado para um sistema de *honeypot*, de modo transparente para o invasor, fazendo-o acreditar que está obtendo acesso ao sistema real, quando na verdade ele está acessando um sistema honeypot com serviços *fakes* que não comprometem o bom funcionamento do ambiente computacional no qual está inserido.

Após o redirecionamento de um atacante para um sistema de *honeypot*, cada ação executada é logada, permitindo ao administrador do sistema obter informações técnicas do ataque e conhecer as vulnerabilidades exploradas por este para obter acesso indevido ao sistema.

Neste trabalho, nós utilizamos como sistema de *honeypot*, um projeto chamado *honeyperl* [1], o qual implementa serviços e portas *fakes* e permite registrar logs de atividade dos usuários que estão interagindo com os falsos serviços. Até o momento são suportados os seguintes serviços e portas: pop3(110), echo(7), ftp(21), httpd(80), smtp(25), squid(3128), além de oferecer módulos para capturar assinaturas de vírus em portas específicas.

O nível de interação do usuário com o honeypot é baixo, utilizando os recursos básicos dos principais serviços que se deseja emular. O ambiente é seguro, de tal modo que o atacante não possui acesso aos serviços reais que estão sendo executados na máquina. Serviços desnecessários não permanecem em execução para reforçar a segurança.

4.6 Conclusão do capítulo

Este capítulo apresentou os sistemas detectores de intrusos, popularmente conhecidos como NIDS. Além das vantagens trazidas por eles, foram também apresentadas as desvantagens dos mesmos, utilizando-se como exemplo o *Snort*, produto mais popular nesta categoria. Foram demonstradas algumas ferramentas que trabalham em conjunto com o *Snort*, inserindo nele as mais diversas funcionalidades.

Por fim, o capítulo fez uma apresentação dos sistemas de *honeypot*, encaixando-os em um ambiente de virtualização e demonstrando como podem ser utilizados para reforçar a segurança de sistemas virtualizados.

5 Como o *XenGuardian* funciona

Este capítulo descreve o funcionamento do *XenGuardian*, uma solução para detecção de intrusos em máquinas virtuais implementada para trabalhar em conjunto com o *Xen* e o detector de intrusos *Snort*. É apresentada a comunicação via *sockets*, na qual é baseada toda a troca de informações entre o *dom0* e os vários *domU*. Em um segundo momento é apresentada a biblioteca *libvirt*, a qual é responsável por extrair informações do *hypervisor* do Xen, facilitando assim o gerenciamento de múltiplas máquinas virtuais.

O *XenGuardian* trabalha no modo cliente-servidor, onde cada máquina virtual *domU* executa em segundo plano uma versão cliente do serviço e envia informações sobre possíveis tentativas de intrusão para o domínio *dom0*. Essas informações enviadas consistem no tráfego de saída coletado no *log* do *Snort* e são classificadas em níveis de prioridade, de acordo com a gravidade do suposto ataque e enviadas através de uma comunicação via *sockets*.

Até o momento o *XenGuardian* foi implementado para executar em sistemas Linux, que executam o monitor de máquina virtual Xen. Os testes descritos no decorrer do capítulo foram realizados na versão 3.1 do Xen e do sistema operacional Debian Etch, versão 4.0, ambos com as últimas atualizações de segurança lançadas.

5.1 Comunicação via Berkeley sockets

A API Berkeley sockets [25] consiste em uma *Application Programming Interface* (API) para desenvolver aplicações em linguagens de programação que necessitam comunicação entre processos, principalmente as que envolvem uma rede de computadores.

Berkeley sockets, também conhecida como *BSD socket API*, foi lançada em 1983 juntamente com a versão 4.2 do sistema operacional BSD. No entanto, apenas em 1989 a Universidade de Berkeley lançou versões gratuitas do BSD e da biblioteca de rede sockets.

Um *socket* é composto por:

- Protocolo de transporte: TCP, UDP ou outro;
- Endereço IP local;
- Porta de comunicação local;
- Endereço IP remoto;

- Porta de comunicação remota.

Existem dois tipos de *sockets*, dos quais veremos a seguir uma breve descrição de cada um [25]:

Stream sockets: baseado no protocolo TCP. Estabelece uma conexão antes da transmissão dos dados, o que gera um overhead maior que o *Datagram sockets*, porém a comunicação oferecida é confiável e respeita a ordem de entrega FIFO[1].

Datagram sockets: baseado no protocolo UDP. O endereço de destino é especificado em cada *datagrama*, gerando menos *overhead* na comunicação. No entanto não garante *confiabilidade* de entrega e nem ordenação.

Neste trabalho será utilizada comunicação entre as máquinas virtuais através do protocolo TCP, pois é imprescindível que as mensagens trocadas entre as máquinas virtuais e a máquina hóspede cheguem na ordem correta, sendo inclusive preferível que as mensagens demorem a chegar ao invés de chegarem fora de ordem. A Figura 15 demonstra como funciona a troca de mensagens utilizando *stream sockets*.

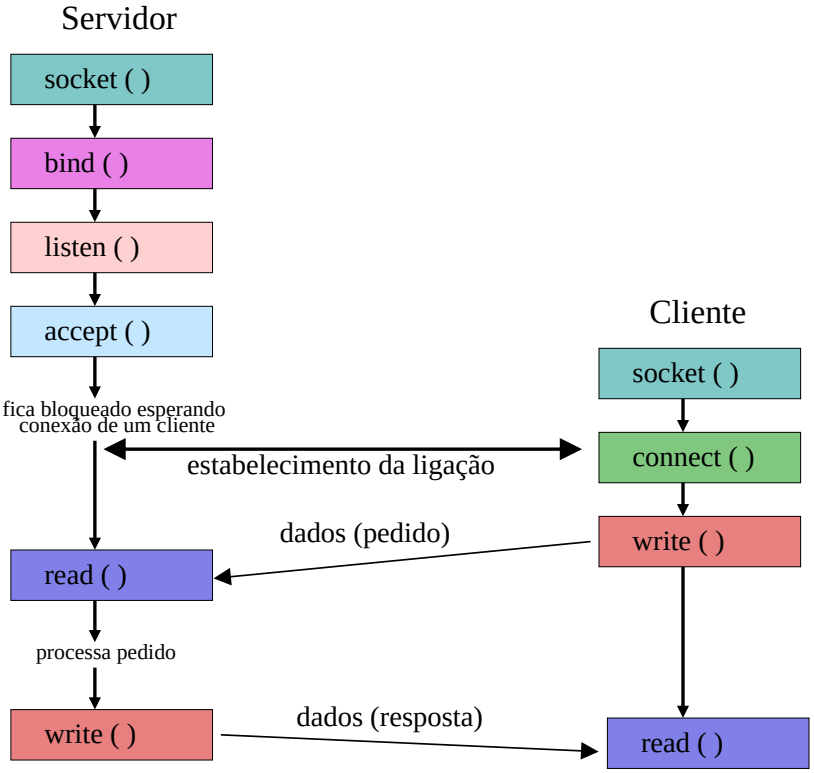

Figura 15 – Estrutura de um *sockets* TCP orientado à conexão.

A biblioteca de comunicação *sockets* possui uma série de funções para estabelecer ligações entre os clientes, a seguir veremos o que faz cada uma das funções utilizadas na Figura 15:

[1] Acrônimo para First In First Out.

socket: cria um ponto para comunicação assíncrona e bidirecional, onde são definidos os protocolos e o domínio do *socket* (local ou remoto);

bind: associa um nome a um *socket* já criado;

listen: indica que o *socket* especificado pode receber ligações;

accept: bloqueia o processo até um processo remoto estabelecer uma ligação. Retorna o identificador de um novo *socket* para transferência de dados;

connect: estabelece uma conexão entre o *socket* local e o *socket* remoto;

read: recebe uma mensagem do *socket* local para o *socket* remoto associado;

write: envia uma mensagem através do *socket* local, para o *socket* associado (local ou remoto).

5.1.1 Biblioteca *libvirt*

A *libvirt* [38] é uma biblioteca que extrai do *hypervisor* do Xen [44] e do QEMU [9] informações sobre as máquinas virtuais que estão em execução no sistema.

Quando o *kernel* modificado do Xen está em execução, os programas podem utilizar a *libvirt* para extrair informações sobre o *dom0*, o qual é o primeiro sistema operacional Linux que realiza o processo de boot no sistema.

O *kernel* deste sistema operacional provê a maioria, se não todos, os drivers utilizados pelos domínios virtuais. Funciona também o mecanismo *Xen Store*, o qual consiste em um banco de dados de informações compartilhadas pelo *hypervisor*, *kernels*, *drivers* e o *Xen daemon*. Este supervisiona o controle e a execução do conjunto de domínios, enquanto o *hypervisor*, *drivers*, *kernels* e *daemons* comunicam-se através de um barramento compartilhado pelo *hypervisor*. A Figura 16 ilustra o funcionando deste ambiente.

A biblioteca *libvirt* surgiu recentemente [38]. Sua primeira versão data de dezembro de 2005. Após mudança no seu nome, correção de muitas falhas e incremento de novas características, seu funcionamento está chegando perto do ideal. A última versão 0.3.3 ainda não funciona como deveria ser, pois não consegue extrair do *hypervisor* todas as informações a que se propõe, no entanto essas limitações não comprometerão o desenrolar deste trabalho.

A biblioteca *libvirt* pode ser inicializada de duas formas, dependendo do level de privilégio do usuário que está operando o sistema. Se o usuário possuir acesso completo ao sistema, a função *virConnectOpen()* pode ser utilizada, permitindo o uso de três maneiras diferentes de conectar-se à infraestrutura do Xen:

- Conexão ao *Xen Daemon* através de uma camada HTTP RPC;
- Permissão de leitura e escrita ao mecanismo *Xen Store*;

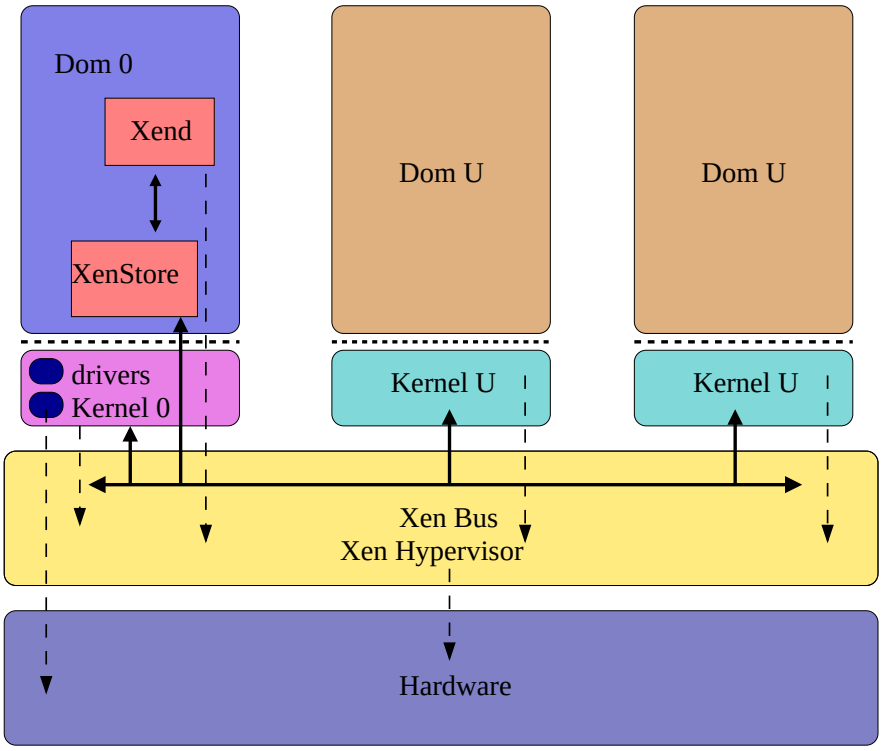

Figura 16 – Arquitetura do sistema no Xen.

- Utilização de chamadas diretas ao *Xen Hypervisor*.

Quando a biblioteca é utilizada por um usuário sem acesso completo ao sistema, ele possuirá acesso somente-leitura e não poderá modificar as estruturas de dados do *Xen Store*. Neste caso, utiliza-se a função *virConnectOpenReadOnly()* para a inicialização da biblioteca, permitindo acesso somente leitura, o que é útil apenas para casos de monitoramento das máquinas virtuais.

A biblioteca interage com o *daemon* do Xen para cada operação que modificar o estado global do sistema, porém para desempenho e exatidão dos resultados é possível realizar comunicação diretamente com o *hypervisor*, afim de recolher informação do estado mais recente possível.

5.2 Detectando uma intrusão

Segundo Denning [19], detecção de intrusão é o processo de monitorar eventos que ocorrem em um sistema computacional ou em redes de sistemas computacionais, analisando-os em busca de problemas de segurança. Em um ambiente virtualizado com Xen, a grande vantagem da detecção de intrusão é que todos os pacotes que trafegam pela interface de rede da máquina virtual *domU*, devem transitar obrigatoriamente pelo sistema hóspede, no caso a máquina *dom0*, o que a torna um pequeno *switch* roteador de pacotes. Este recebe todas as requisições de suas máquinas virtuais e encaminha-as para os seus devidos destinos, sejam eles a rede externa ou a

rede interna, enviando para outra máquina virtual.

O mesmo autor desenvolveu em 1987 uma pesquisa de detecção de intrusão através da análise de mudança nos padrões de usuários, onde são apurados valores em relação à utilização normal do sistema considerando o uso da CPU, número de conexões por minuto, número de processos ativos por usuário, dentro outros fatores. A variação, quando significativa, destes padrões pode sinalizar uma tentativa de intrusão, tendo em vista a ocorrência de um padrão anormal de uso do sistema.

É muito difícil realizar detecção de intrusos através das mudanças de padrões pelo fato de não existir um padrão pré-determinado que possa ser monitorado, o que pode ocasionar em várias ocorrências de falsos positivos. Atualmente existem duas principais técnicas de detecção de intrusos: detecção por anomalia e detecção por assinatura. Veremos adiante como as duas funcionam, além de suas vantagens e desvantagens.

5.2.1 Detecção por anomalia

Sistemas de detecção de intrusos por anomalia também são chamados de sistemas de detecção por comportamento. Estes procuram criar modelos para representar o comportamento normal ou esperado do sistema, por exemplo: o sistema baseia-se nas últimas sessões de um determinado usuário e modela um comportamento de que o usuário executará no máximo 10 processos por sessão e utilizará no máximo 20 conexões TCP a cada login no servidor. Se ocorrer alguma situação fora desse comportamento pré-definido, o sistema sinalizará uma tentativa de intrusão e pode bloquear o acesso do usuário ao sistema computacional.

Os princípios utilizados para traçar o perfil de cada usuário do sistema são baseados em estatísticas, tais quais: tempo de uso de determinado aplicativo e média de comandos por sessão. É importante que os perfis dos usuários sejam atualizados constantemente com o decorrer natural da utilização do sistema, pois os usuários tem padrões de uso do sistema que variam com o decorrer do tempo.

São poucos os benefícios proporcionados pelos sistemas com detecção de intrusos por comportamento, podendo-se citar:

- Capacidade de detectar ataques sem ter grandes conhecimentos sobre eles;
- Permite produzir informações para serem utilizadas na geração de assinaturas para sistemas de detecção por assinaturas.

Entretanto, em maior número, pode-se citar como desvantagens da detecção por comportamento:

- Usuário fica "preso” ao sistema, se um usuário legítimo precisar realizar uma atividade fora do comportamento pré-definido, podendo ter seu acesso restringido;

- Requer um tempo de treinamento para poder traçar os perfis de uso do sistema para cada usuário;
- É possível que um atacante faça o sistema aceitar seu comportamento anormal como sendo legítimo, através de pequenas mudanças ao longo do tempo;
- Grande ocorrência de falsos-positivos e falsos-negativos.

5.2.2 Detecção por assinatura

Os sistemas de detecção de intrusos por assinatura são conhecidos pelo fato de procurarem por padrões de ataques e intrusões previamente conhecidos através de padrões de assinaturas ou em conhecimentos prévios do IDS. Para estes sistemas funcionarem adequadamente, faz-se necessário a existência de ataques com características bem definidas, não-aleatórias e que possam ser codificadas em um sistema especialista.

O *Snort* é o mais popular sistema de detecção de intrusos que utiliza a técnica de detecção por assinatura. Neste trabalho ele é apresentado na seção 4.2. Ele pode facilmente implementar regras para ataques bem definidos, como por exemplo, o popular ataque *"Ping of Death"* que consiste em enviar um pacote malformado muito grande, o qual acarretava no travamento do sistema recebedor do pacote. Um *ping* possui normalmente o tamanho de 64 *bytes*. Para gerar o ataque *"Ping of Death"* usuários mal intencionados alteravam ferramentas de *ping* para gerar pacotes com tamanho maior e, como alguns sistemas operacionais não estavam tratando pacotes recebidos com tamanho acima de 64 *bytes*, era gerada uma falha que ocasionava no travamento do sistema. Atualmente quase todos os sistemas operacionais possuem proteção contra alguns tipos de pacotes TCP/IP malformados, mas na época em que a falha surgiu foram grandes os contratempos causados aos administradores.

As vantagens apresentadas através da detecção por assinatura são:

- Possibilidade de detectar ataques sem gerar excessivos falsos positivos;
- Permitem diagnosticar de maneira rápida e segura o uso de determinadas ferramentas de *exploits*, as quais serão apresentadas na seção 6.1;
- Possibilidade de atualização das assinaturas semanalmente/diariamente de acordo com o surgimento de novos ataques;
- O próprio administrador pode criar suas assinaturas locais de acordo com as suas necessidades;
- Pelo fato do sistema detector de intrusos não receber atualizações muito constantes (somente são atualizadas as assinaturas), torna-se mais fácil propagar e manter a detecção atualizada em todos os computadores da rede local.

Por outro lado, podem-se citar algumas desvantagens ocasionadas pela detecção por assinaturas:

- Conseguem detectar somente tipos conhecidos de ataques;
- Devido à rigidez com que as assinaturas são elaboradas, algumas variantes de um mesmo ataque podem passar despercebidas e não serem detectadas.

5.2.3 Modos de operação de uma interface de rede

Os sistemas que monitoram a rede em busca de conexões ou pacotes maliciosos são compostos obrigatoriamente por um módulo de captura de pacotes denominado *sniffer*. Para que um *sniffer* possa funcionar, ele deve configurar a placa de rede para trabalhar em modo promíscuo. A diferença entre o modo normal para o promíscuo é que, enquanto no modo normal, se um pacote não estiver endereçado ao próprio computador (excluindo aqui pacotes *broadcast* e *multicast*) ele será imediatamente descartado. Por outro lado, no modo promíscuo a placa de rede consegue capturar todos os pacotes independente de quais sejam os seus endereços de destino. Para um melhor entendimento as Tabelas 1 e 2 exibem os cabeçalhos dos protocolos IPv4 e TCP, os quais são capturados quando a placa de rede estiver operando em modo promíscuo.

Tabela 1 – Cabeçalho do protocolo IPv4

Bits	0-3	4-7	8-15	16-18	19-31
0	Versão	Tamanho do Cabeçalho	Tipo de Serviços (ToS) (agora DiffServ e ECN)	Comprimento do pacote	
32				Flags	Fragment Offset
64	Tempo de Vida (TTL)		Protocolo	Header Checksum	
96	Endereço origem				
128	Endereço destino				
160	Opções (caso existir)				
192	Dados				

Tabela 2 – Cabeçalho do protocolo TCP

Bits	0-3	4-7	8-15	16-31
0	Porta origem			Porta destino
32	Número de sequência			
64	Acknowlegement			
96	Tamanho	Reservado	Flags	Window
128	Checksum			Urgent Pointer
160	Opções (caso existir)			
160/192+	Dados			

5.2.4 Arquitetura do *XenGuardian*

A estrutura hierárquica da arquitetura do Xen permite que uma *dom0* possua conhecimento de cada pacote que é trafegado por suas máquinas virtuais, aceitando realizar intervenções quando algum pacote suspeito tiver destino para alguma máquina virtual que estiver sendo executada sob o *dom0*. A Figura 17 apresenta o modelo proposto para a arquitetura do *XenGuardian*,

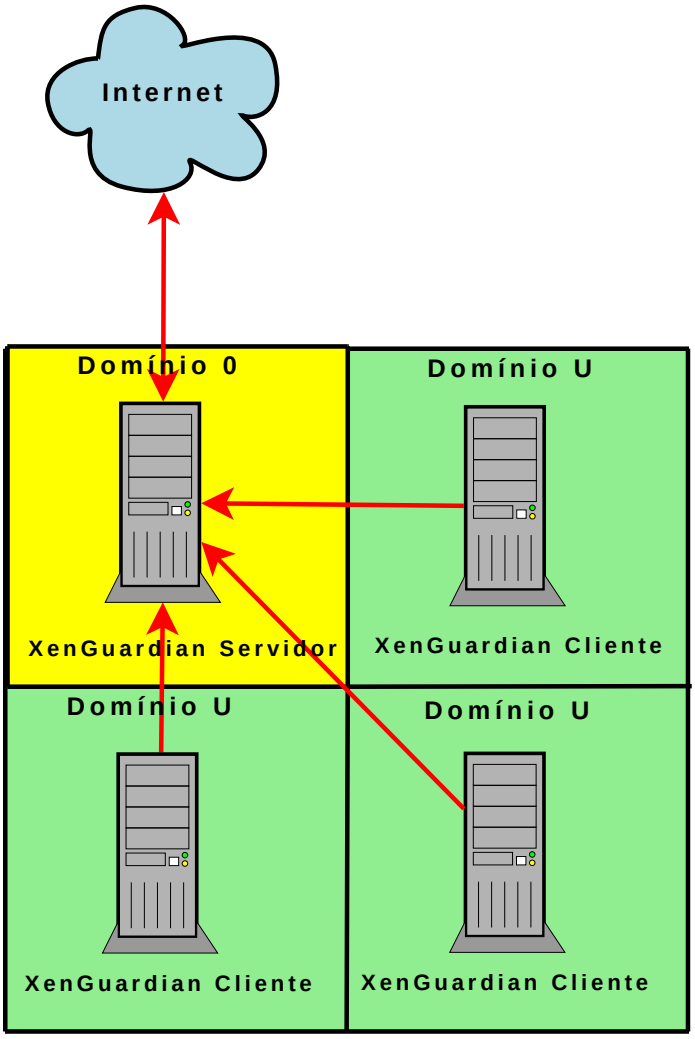

Figura 17 – Arquitetura do XenGuardian

Com base nos alertas recebidos, a máquina *dom0* pode definir qual o dano do ataque, baseado em níveis de prioridade que variam de 1 a 5, onde 5 consiste em um ataque perigoso que requer mais atenção. Após a detecção do ataque ser bem sucedida, o atacante é direcionado de forma transparente para uma honeypot [55] ou ainda existe a possibilidade de bloquear o acesso à interface de rede, até que o administrador do sistema possa realizar uma ação corretiva.

A próxima seção apresenta como o *XenGuardian* realiza a análise do *log* do *Snort*, esse procedimento ocorre após a correta identificação de um intruso tentando obter acesso ao sistema.

5.3 Analisando o log do *Snort*

Por padrão, toda vez que o *Snort* encontrar algum tráfego suspeito na interface de rede que está sendo analisada, será gerado um arquivo de *log* contendo informações detalhadas a respeito do ataque detectado. Essas informações são a data e a hora do ataque, o protocolo no qual a falha foi explorada, além dos respectivos endereços IPs de origem e destino que foram utilizados.

O *XenGuardian* trabalha realizando a leitura do *log* de saída do *Snort*. Esta seção trata de apresentar como estão estruturadas as informações coletadas, citando três tipos de ataques que são detectados e salientando qual o prejuízo que os mesmos trazem para o sistema.

1. **DOS IGMP dos attack:** Ataque baseado na referência CVE [2] 1999-0918. É causado através de pacotes mal formados, normalmente causados por *sniffers*, podendo sinalizar uma tentativa de ataque de negação de serviço;

2. **BAD-TRAFFIC loopback traffic:** Este ataque acontece quando existe "*BAD-TRAFFIC*" no endereço local é caracterizado quando alguém envia um ataque para determinado endereço IP forjando o endereço de origem para 127.x.x.x. A Figura 18 exibe um exemplo do arquivo de saída do *Snort*, quando ocorre este tipo de ataque;

3. **BAD-TRAFFIC same SRC/DST:** Baseado na referência *CERT Advisory CA-1997-28 IP Denial-of-Service Attacks*, esta assinatura protege contra dois tipos diferentes de ataques: *Teardrop* e *Land*. No caso do *Teardrop*, ocorre quando o atacante insere informações confusas no *offset* dos pacotes IP. Para trafegar entre roteadores os pacotes são divididos em fragmentos, e se o sistema operacional não conseguir tratar essas informações confusas no fragmento, pode causar travamento do sistema. Para um ataque de *Land* ocorrer, o atacante deve realizar um *spoofing* do endereço IP da vítima, com o mesmo número de porta de origem e destino, o que pode fazer com que a máquina tente responder o pacote para si mesma indefinidamente.

```
[**] [1:528:5] BAD-TRAFFIC loopback traffic [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/14-12:19:42.875221 127.255.202.17 -> 192.168.100.234
ICMP TTL:255 TOS:0x0 ID:40560 IpLen:20 DgmLen:84 MF
Frag Offset: 0x0000 Frag Size: 0x0040
```

Figura 18 – Exemplo de um ataque do tipo *BAD-TRAFFIC loopback.*

O *XenGuardian* implementa um *parser* do arquivo de *log* do *Snort*, coletando as seguintes informações: tipo do ataque, classificação, data e hora do ataque e o protocolo que o atacante tentou explorar. O Algoritmo 1 exibe quais são as informações extraídas do *log* do *Snort* e

[2] *Common Vulnerabilities and Exposures*, trata-se de uma associação que mantém um banco de dados com informações públicas a respeito de vulnerabilidades em computadores.

após a leitura é criado um registro no *XenGuardian* sinalizando que um ataque foi identificado pelo sistema. Se o tipo de ataque que estiver registrado no *log* do *Snort* for reconhecido, as informações da ocorrência serão gravadas em uma estrutura de dados e é feito o bloqueio do usuário que está tentando acessar indevidamente o sistema. As informações coletadas pelo *XenGuardian* são:

- Tipo de ataque: nomeia o tipo de ataque, podendo ser uma tentativa de *denial of service*, *bad traffic*, entre outros;
- Classificação: apresenta uma breve descrição e categoriza o ataque de acordo com as suas características;
- Prioridade: a prioridade pode variar de 1 (menor gravidade) até 5 (ataque de maior gravidade). A partir da prioridade 4 um ataque pode causar dano considerável ao sistema;
- Data: armazena a informação da data em que foi detectado o ataque. Essa informação é coletada de acordo com o horário do sistema, por isso é de fundamental importância manter o horário correto para não prejudicar o funcionamento da ferramenta;
- Hora: registra a hora em que a tentativa de ataque foi detectada pelo *XenGuardian*, informação também coletada através do horário existente no sistema;
- Tipo do pacote: o tipo do pacote registra o protocolo explorado para realizar a tentativa de intrusão, pode ser do tipo: IGMP, ICMP, TCP ou UDP.

Algoritmo 1: Estrutura de parser do *XenGuardian*

Entrada: Variáveis do tipo *char*: tipodeataque, classificacao, prioridade, data, hora, tipopacote.

```
1 enquanto Não chegar ao final do arquivo de log faça
2     Leia o log do Snort;
3     se Ataque.Tipodeataque = Ataque.Reconhecido então
4         Executa rotina para gravar ataque na estrutura de dados.
5     fim se
6 fim enquanto
```

Quando um ataque é detectado pelo Snort, o mesmo é identificado pelo XenGuardian cliente, que faz a leitura do arquivo de *log* e realiza o devido tratamento da ocorrência. Durante esse tratamento classifica-se a gravidade do ataque, para em seguida enviar um aviso para o XenGuardian servidor, o qual ficará encarregado de bloquear o endereço IP do suposto atacante, conforme ilustra a Figura 19.

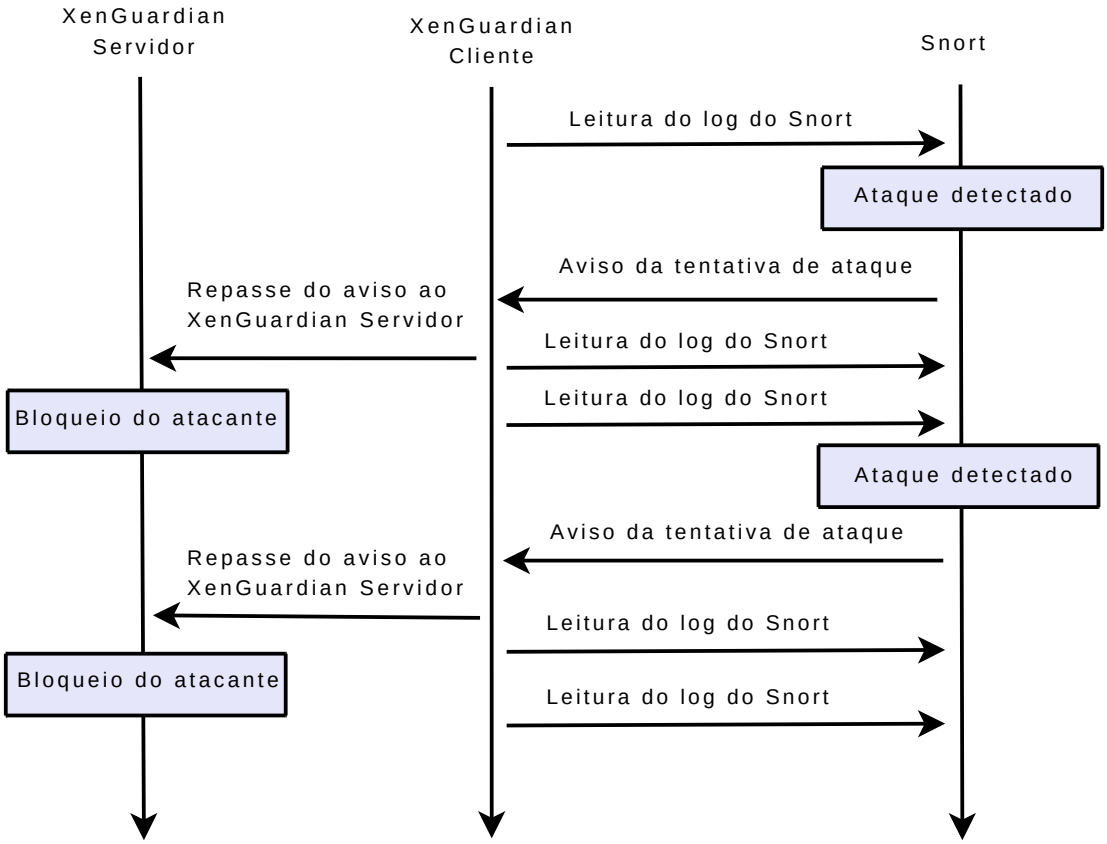

Figura 19 – Funcionamento do XenGuardian

5.4 Conclusão do capítulo

Este capítulo apresentou a ferramenta *XenGuardian*, com ênfase para os princípios que norteiam o seu funcionamento e a sua arquitetura. Também foi citada a biblioteca *libvirt* e o funcionamento da comunicação via *sockets*, as quais norteiam o funcionamento da solução apresentada neste trabalho. Todas as máquinas virtuais (*domU*) de um domínio, trocam informações com a máquina hóspede (*dom0*) através de uma comunicação via *sockets*, além da máquina hóspede extrair informações do *hypervisor* através da biblioteca *libvirt*.

No capítulo também foi apresentada a arquitetura do *XenGuardian*, além de três tipos de ataques que são reconhecidos pelo sistema proposto e detalhado o procedimento de *parseing*, além de demonstrados exemplos de *logs* do *Snort*, os quais são lidos, monitorados, interpretados e realizado o bloqueio das tentativas de ataque, quando houver necessidade.

6 Resultados experimentais

Este capítulo apresenta os resultados obtidos com a execução de testes com o *Netperf* e com *exploits*, sempre buscando mensurar efetividade e interferência. Entende-se por efetividade o quão eficiente é a solução apresentada e até que ponto ela satisfaz as expectativas, protegendo o sistema contra tentativas de ataques. Por interferência, entendemos que trata-se de uma métrica importante, que corresponde a perda de desempenho a partir do momento em que o *XenGuardian* está sendo executado em *background*. Essa perda pode ser causada pelo aumento de tráfego na interface de rede, aumento da memória principal que está sendo utilizado, além de outros fatores - quanto menor for essa perda, melhor.

6.1 Exploits utilizados para testes

Para validar a solução proposta, realizou-se testes com *exploits*, que nada mais são do que ferramentas desenvolvidas com o propósito de explorar falhas de segurança introduzidas em aplicativos ou ainda por erros involuntários de programação em sistemas computacionais.

Um *exploit* pode ser preparado para atacar um sistema local ou remoto, pelo fato de explorarem falhas específicas. Normalmente há um *exploit* para cada tipo de falha, para cada versão de determinado *software* ou ainda para cada tipo de sistema operacional.

Geralmente um *exploit* se aproveita de uma falha conhecida como *buffer overflow* ou estouro de *buffer*, caso que ocorre quando um programa tenta gravar uma informação em uma variável, passando no entanto, uma quantidade de dados maior do que a que estava prevista pelo programa. Quando essa situação ocorre é possível executar um código arbitrário, desde que ele seja posicionado dentro da área de memória do processo que gerou a falha de *buffer overflow*.

Exploits também exploram as falhas inerentes do protocolo TCP [45] para derrubar redes de computadores através da sobrecarga no envio de pacotes. Normalmente nos ataques em redes de computadores, os *exploits* geram propositalmente pacotes mal formados que ficam trafegando e congestionando as redes, impedindo que usuários legítimos possam fazer uso dos recursos computacionais, já que os mesmos estarão ocupados respondendo grandes quantidades de pacotes que foram geradas no ataque.

Para o *XenGuardian* funcionar a contento, é indispensável que o *Snort* possua a capacidade de reconhecer e detectar pelo menos os ataques mais comuns das redes, partindo do pressuposto

básico que não é possível proteger um ataque que não é conhecido. Para averiguar a capacidade do *Snort* de reconhecer ataques, foram testadas três ferramentas populares de *Denial of Service*, com as descrições e respectivos exemplos descritos a seguir:

Datapool: A última versão do *Datapool* [43] combina 106 ataques de *Denial of Services* dentro de um mesmo *script*. Essa versão possui uma base de dados poderosa que aprende de acordo com os ataques bem sucedidos contra cada *host*. Assim, da próxima vez que o mesmo *host* for atacado será utilizado primeiramente o tipo de ataque mais bem sucedido realizado na última execução. Novas características desta versão são: *log* de ataques, especificação de intervalos de portas, opções para ataques contínuos, ataques em múltiplos endereços IP simultaneamente além de realizar ataques repetitivos de múltiplos endereços IPs. A Tabela 20 exibe o *Datapool* em ação executando alguns tipos de ataques.

```
Checking to see if 192.168.1.204 is alive ...
Host is alive, starting portscan...
Running IGMPICMPUDPTCP attack (flatline)...
Running ICMP attack (moyari13)...
Running SYNUDPICMP attack (spender)...
Running UDP datagram attack (arnup100)...
Running inetd attack (inetd.DoS)...
Running ICMP attack (DoS-Linux)...
...
```

Figura 20 – Ferramenta *Datapool* em ação.

Hping2: É uma ferramenta utilizada em linha de comando. Trata-se de um *ping* convencional com uma grande quantidade de recursos extras e avançados para administradores de rede. Além de ser um analisador de pacotes TCP/IP possui suporte aos protocolos TCP, UDP, ICMP e RAW-IP [29]. A Tabela 21 exibe um exemplo da execução do hping2 utilizando pacotes do tipo *syn*, uma maneira de realizar ping em servidores que bloqueiam pacotes do tipo *Internet Control Message Protocol* (ICMP).

```
xensystem: exploits# hping2 syn 192.168.1.202 p 25
HPING 192.168.1.202 (eth0 192.168.1.202): S set, 40 headers + 0 data bytes
len=40 ip=192.168.1.202 ttl=64 DF id=0 sport=25 flags=RA seq=0 win=0 rtt=1.0 ms
len=40 ip=192.168.1.202 ttl=64 DF id=0 sport=25 flags=RA seq=1 win=0 rtt=0.5 ms
len=40 ip=192.168.1.202 ttl=64 DF id=0 sport=25 flags=RA seq=2 win=0 rtt=0.4 ms

— 192.168.1.202 hping statistic —
3 packets transmitted, 3 packets received, 0round-trip minavgmax = 0.40.61.0 ms
```

Figura 21 – Ferramenta Hping2 em ação.

IDSwakeup: Consiste em uma coleção de ferramentas que permitem a realização de testes de sistemas detectores de intrusos baseados em rede. Permite gerar falsos ataques que

imitam tipos muito bem conhecidos, objetivando verificar se o NIDS detecta determinado tipo de ataque e se ele gera falsos-positivos [31]. Na Tabela 22 é possível verificar o IDSwakeup enviando alguns tipos de ataques diversos para um determinado *host*.

```
src_addr:192.168.1.202 dst_addr:192.168.1.204 nb:5 ttl:5
sending : teardrop ...
sending : land ...
sending : get_phf ...
sending : bind_version ...
sending : get_phf_syn_ack_get ...
sending : ping_of_death ...
sending : syndrop ...
sending : newtear ...
...
```

Figura 22 – Ferramenta IDSwakeup em ação.

Pelo fato das ferramentas de ataques utilizarem vários *scripts* com diversos ataques distintos em cada um deles, fica difícil saber se o *Snort* é capaz de detectar todo e qualquer tipo de ataque, se formos analisar os *scripts* de maneira individual. No entanto quando os *exploits* estão em execução e realizam ataques na rede que o *Snort* está protegendo, diversos alertas são gerados e as assinaturas do *Snort* são capazes de detectar os ataques e na maioria das ocasiões as ocorrências foram devidamente detectadas e registradas em arquivos de *log*.

Após a gravação com sucesso do *log* registrando os ataques sofridos, o *XenGuardian* é capaz de realizar a leitura do arquivo em tempo de execução e realizar o devido tratamento para cada um dos ataques ocorridos. O tratamento pode ser o bloqueio total do endereço IP do atacante ou ainda direcionar o mesmo de modo transparante para um sistema de *honeypots*, permitindo que o atacante tenha a ilusão de estar acessando os serviços reais do sistema, quando na verdade ele está em um ambiente totalmente controlado e interagindo com serviços *fakes*.

6.2 Resultados

Esta seção apresenta os resultados obtidos dos testes realizados com o *XenGuardian*. Foram realizados testes com as ferramentas de ataques à rede de computadores, descritas na seção 6.1, objetivando testar se os ataques seriam detectados pelo *XenGuardian*, consistindo assim a nossa análise de efetividade. Em um segundo momento, objetivando realizar testes de interferência, foi utilizado o *benchmark Netperf* para medir o quão impactante é a atividade do *XenGuardian* no desempenho do sistema.

As configurações das máquinas utilizadas nos testes estão descritas a seguir, com pequenas alterações da *dom0* para a *domU*.

***Dom0*:** Memória: 131264 Kb, CPU: Pentium 3 700 MHZ, IDE HD: 9GB, NIC: 100Mpbs.

***DomU*:** Memória: 65728 Kb, utilizando *file-backed virtual block device*, partição swap: 128Mb e partição principal: 1Gb.

6.2.1 Análise de efetividade

Objetivando medir a efetividade do Xen, usou-se os *exploits* descritos na seção 6.1 para avaliar se a solução apresentada era capaz de reconhecer certos padrões de ataque e barrá-los assim que fossem encontrados.

Com a execução do *exploit* Hping, visto na seção 6.1, utilizando como máquina de origem a detentora do endereço IP 10.30.30.209 com destino para a máquina 10.30.30.194, o *XenGuardian* detectou e bloqueou imediatamente a tentativa de ataque, como pode ser observada na Figura 23 onde a ferramenta detectou um ataque do tipo DOS IGMP, o qual é apresentado com maiores detalhes na seção 5.3.

```
Tipo de ataque: DOS IGMP dos attack
Classificação: Attempted Denial of Service
Prioridade [1-3]: 2
Origem: 10.30.30.209
Destino: 10.30.30.194
Data do ataque [mes/dia]: 01/28
Hora do ataque [hr:min:seg]: 18:12:15
Tipo do pacote [UDP/TCP/ICMP/IGMP]: IGMP
ENDEREÇO IP DE ORIGEM BLOQUEADO!!
```

Figura 23 – *XenGuardian* detectando e bloqueando ataque DOS IGMP.

Para dar continuidade aos testes com *exploits*, além do Hping, foram utilizadas as outras ferramentas descritas na seção 6.1. A Figura 24 mostra o *XenGuardian* detectando um ataque do tipo *BAD-TRAFFIC loopback traffic*, visto com maiores detalhes na seção 5.3. Para realizar a tentativa de ataque foi utilizada a ferramenta IDSwakeup que consiste em um *exploit* desenvolvido com uma série de *scripts* elaborados para ataques específicos em sistemas detectores de intrusos. O *XenGuardian* foi eficiente o suficiente para detectar o ataque, no entanto pelo fato do endereço IP de origem ter sido forjado e possuir o conteúdo 127.255.202.17, não foi possível realizar o bloqueio do suposto atacante. Como esse tipo de ataque possui uma prioridade pequena e não causa maiores danos ao sistema, o não bloqueio do invasor não compromete o sistema.

Do mesmo modo, a imagem da Figura 25 foi obtida utilizando o *exploit Datapool*, onde é identificado ataque do tipo *BAD-TRAFFIC same SRC/DST*, o qual é visto com maiores detalhes na seção 6.1.

Foram realizados testes com as três ferramentas de *exploits*, citadas na seção 6.1. Nos cenários de testes quando não eram realizados muitos ataques simultaneamente, o *XenGuardian*

Tipo de ataque: BAD-TRAFFIC loopback traffic
Classificação: Potentially Bad Traffic
Prioridade [1-3]: 1
Origem: 127.255.202.17
Destino: 10.30.30.194
Data do ataque [mes/dia]: 01/28
Hora do ataque [hr:min:seg]: 18:20:37
Tipo do pacote [UDP/TCP/ICMP/IGMP]: ICMP
ATENÇÃO!! ENDEREÇO IP DE ORIGEM FORJADO!!
BLOQUEIO NÃO REALIZADO

Figura 24 – *XenGuardian* detectando ataque BAD-TRAFFIC.

Tipo de ataque: BAD-TRAFFIC same SRC/DST
Classificação: Potentially Bad Traffic]
Prioridade [1-3]: 3
Origem: 10.30.30.209
Destino: 10.30.30.194
Data do ataque [mes/dia]: 01/28
Hora do ataque [hr:min:seg]: 18:31:42
Tipo do pacote [UDP/TCP/ICMP/IGMP]: ICMP
ENDEREÇO IP DE ORIGEM BLOQUEADO!!

Figura 25 – *XenGuardian* detectando e bloqueando ataque same SRC/DST.

foi capaz de identificar o ataque e bloquear o endereço IP de origem, salvo no caso do endereço IP de origem ter sido forjado para realizar o ataque, como é o caso da Figura 24. No entanto, se uma grande quantidade simultânea de ataques de tipos variados for desferida contra a máquina que está sendo protegida, a ferramenta de proteção fica impossibilidade de atender a todas as requisições e algumas tentativas de ataques passaram despercebidas.

Acredita-se que a impossibilidade de bloquear uma quantidade grande de pacotes no *XenGuardian* ocorre pelo fato da comunicação entre as máquinas virtuais ser realizada via *sockets* com o protocolo TCP, o qual garante entrega ordenada e confiável, mas em contrapartida tende a demorar mais para atender a todas as solicitações por parte das máquinas virtuais.

6.2.2 Análise de interferência

Para medir o desempenho e respectivo *overhead* do *XenGuardian* em um ambiente de rede, foi escolhida a ferramenta de *benchmark Netperf* [27]. A escolha se deve pelo fato desta permitir a realização de uma grande variedade de testes em diversos aspectos de uma rede, além de ser utilizada por várias instituições em todo o mundo e possuir uma vasta documentação disponível, em detrimento com outros benchmarks pesquisados, tais quais o NetPIPE [49].

O *benchmark Netperf* é composto de dois módulos distintos, existe o *Netserver* (servidor) e o *Netperf* (cliente). Ambos podem ser utilizados em uma mesma máquina, sendo executando

em diferentes portas. Para executar nossos testes, a metodologia escolhida foi enviar pacotes para uma máquina virtual *domU* com e sem o *XenGuardian* instalado e medir se a existência da ferramenta no sistema afeta o desempenho geral do sistema.

O *benchmark* foi executado dez vezes, onde foram descartados o melhor e o pior resultados e realizada uma média aritmética simples com os resultados restantes. Dentre as várias opções de testes disponibilizadas pelo *Netperf*, para o protocolo TCP, foram escolhidos os testes de *TCP Request/Response*, para realizar teste de latência, *TCP Stream*, para realizar teste de envio de arquivo e *TCP Maerts*, para testar velocidade de recebimento de dados. Por último foi realizado o teste *TCP Connect/Request/Response* [41] para emular o tráfego padrão de um servidor *web*, sendo utilizados os seguintes parâmetros para cada tipo diferente de teste, respectivamente:

- /usr/local/bin/netperf -t TCP_RR -H 192.168.1.202 -i 10,2 -I 99,5
- /usr/local/bin/netperf -t TCP_STREAM -H 192.168.1.202 -i 10,2 -I 99,5
- /usr/local/bin/netperf -t TCP_MAERTS -H 192.168.1.202 -i 10,2 -I 99,5
- /usr/local/bin/netperf -t TCP_CRR -H 192.168.1.202 -i 10,2 -I 99,5

Para efeito de comparação, também foram realizados dois testes com o *Netperf* através do protocolo UDP: teste de latência através do UDP *Request/Response* e teste de envio de arquivo com o UDP *Stream*. Para realização dos referidos testes foram utilizados os seguintes parâmetros:

- /usr/local/bin/netperf -t UDP_RR -H 192.168.1.202 -i 10,2 -I 99,5
- /usr/local/bin/netperf -t UDP_STREAM -H 192.168.1.202 -i 10,2 -I 99,5

Os parâmetros -I e -i permitem especificar o nível e o intervalo de confiança, para assim diminuir a margem de erro e obter resultados mais precisos. Foi utilizada a taxa de 99% para o nível de confiança e de 5% para mais ou para menos para o intervalo de confiança. Quando esses parâmetros são utilizados, o *Netperf* requer a especificação da quantidade mínima e máxima de iterações que serão realizadas para tentar obter os níveis de confiança desejados. Os parâmetros utilizados para essa opção foram o mínimo 2 e o máximo 10 iterações.

Todas as chamadas do *Netperf* foram realizadas a partir de uma máquina externa ao ambiente virtualizado, a máquina virtual possuía o IP 192.168.1.202 e executou o *netserver daemon* para abrir uma porta em modo *bind*. Os gráficos constantes nas Figuras 26 e 27 exibem os resultados comparativos das execuções do *Netperf* para os testes *TCP Request/Response* e *TCP Connect/Request/Response* que estão medidos em transações por segundo, enquanto os testes de *TCP Stream Test* e *TCP Maerts Test* são mensuráveis em *megabits* (10^6 bits/sec).

Os gráficos das Figuras 26 e 27 permitem constatar que o sistema onde o *XenGuardian* estava em execução, teve um desempenho ligeiramente menor, se comparado ao sistema que não possuía o monitoramento realizado pelo *XenGuardian* e estava desprotegido contra os ataques.

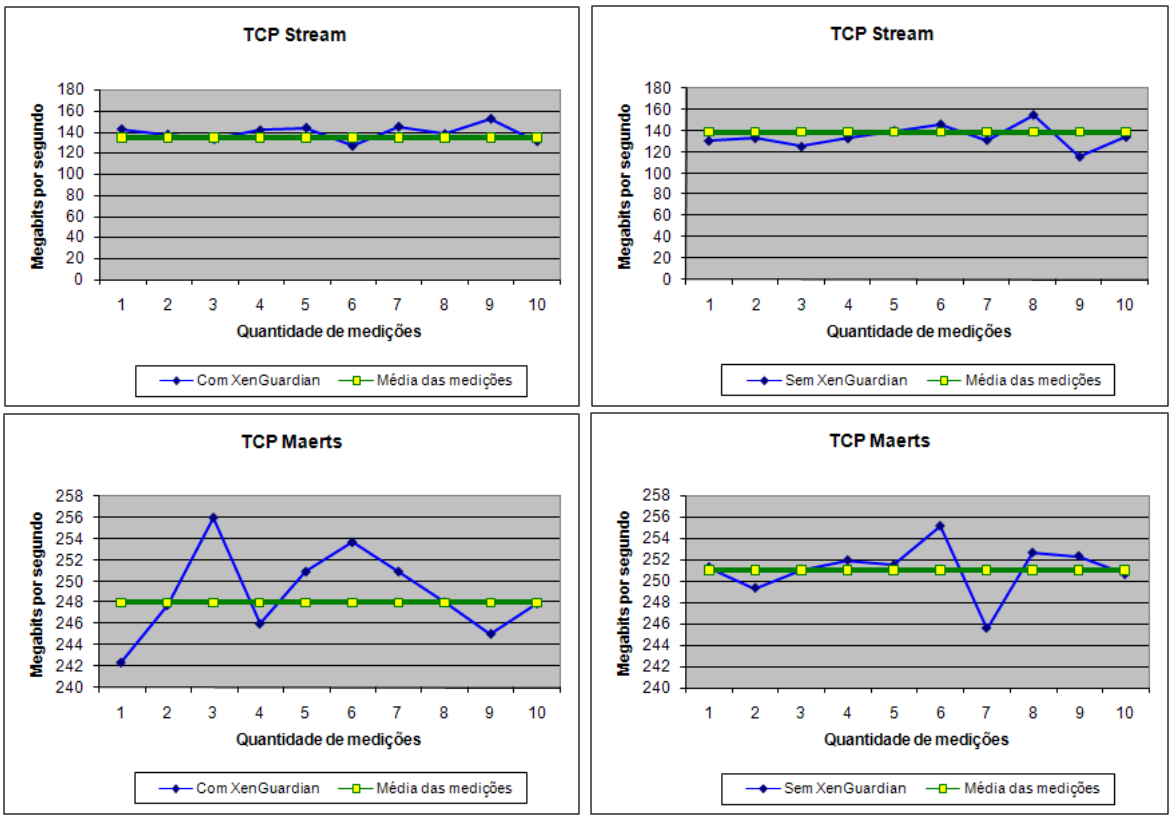

Figura 26 – Gráficos comparativos utilizando protocolo TCP - em *megabits*.

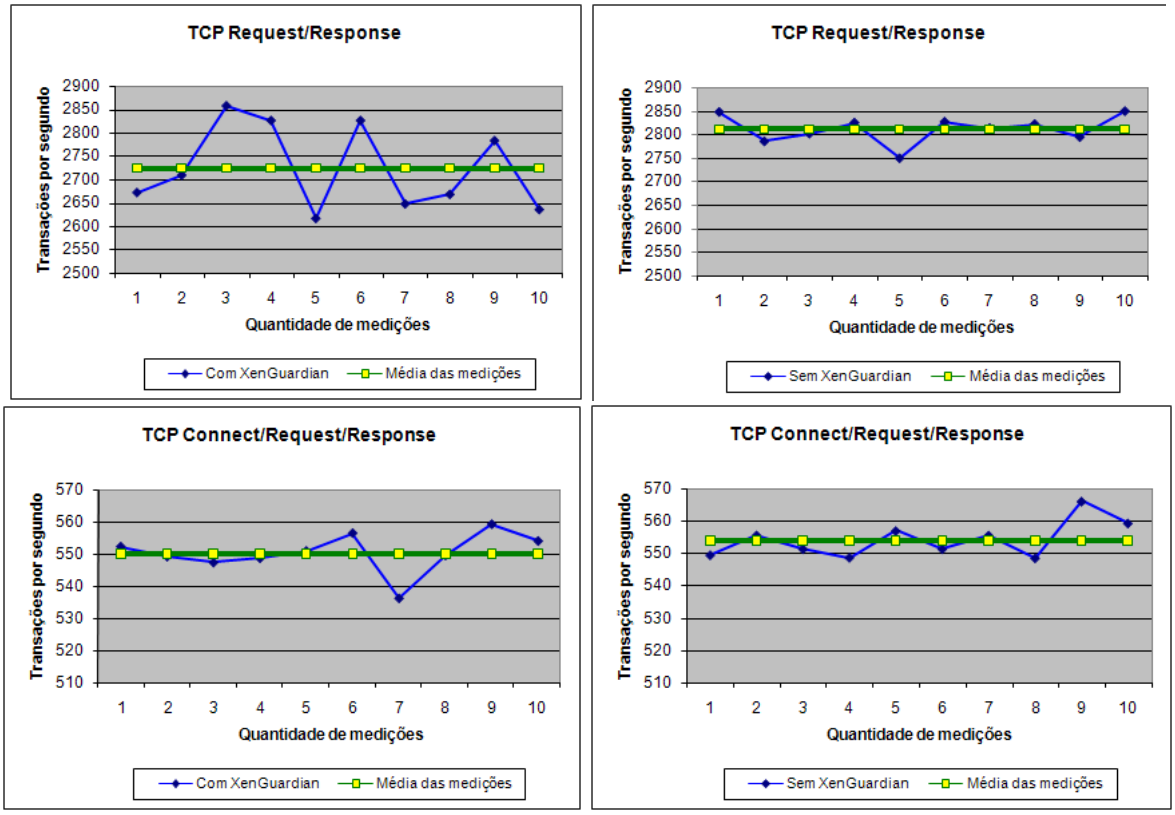

Figura 27 – Gráficos comparativos utilizando protocolo TCP - em transações/s.

Visando obter testes com um protocolo diferente do TCP, foram realizadas medições e obtidos os gráficos da Figura 28. O protocolo UDP teve um desempenho melhor que o TCP em todas as medições, esse fato ocorre pelo fato do protocolo UDP não oferecer garantia de entrega de mensagens, tampouco ordenamento na entrega e nem reenvio de pacotes em caso de perda. Esses fatores justificam o melhor desempenho obtido com o protocolo em questão.

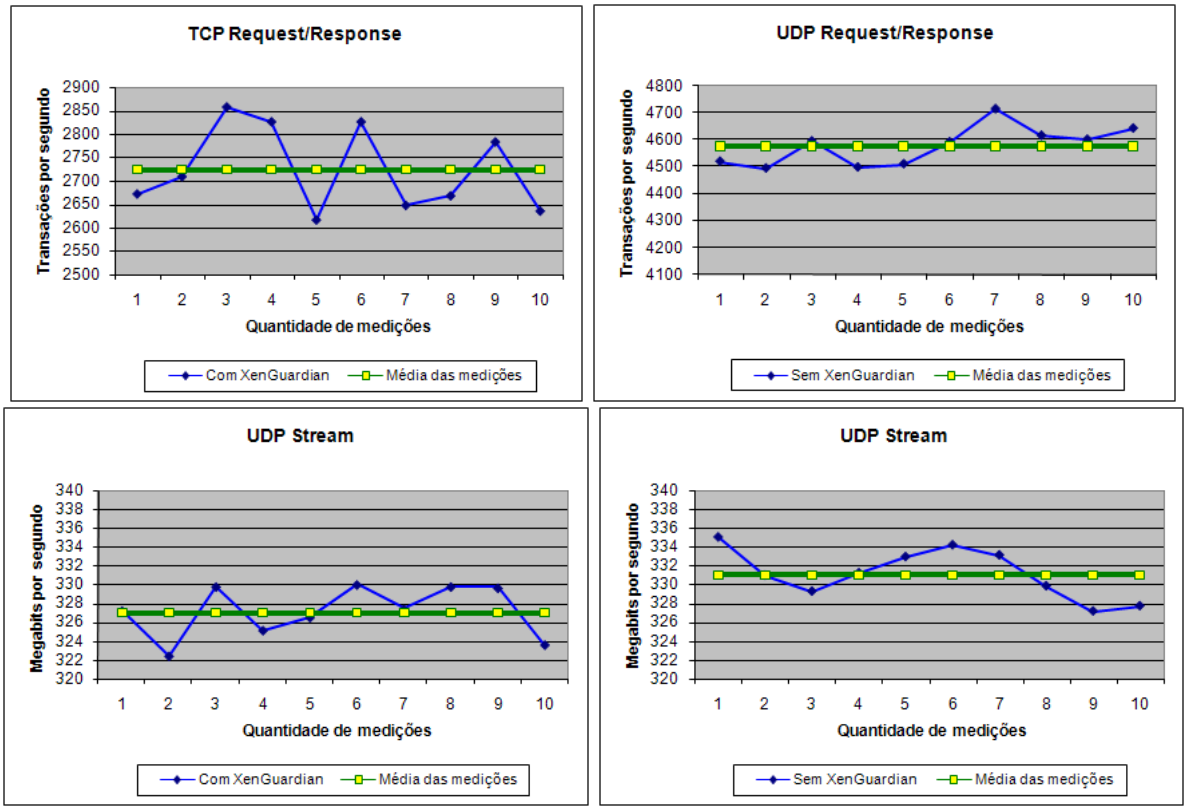

Figura 28 – Gráficos comparativos utilizando protocolo UDP.

Os testes realizados sob o protocolo são: UDP *Request/Response*, medido em transações por segundo e UDP *Stream* medido em *megabit/s* (10^6 bits/sec). Medições foram realizadas com e sem o *XenGuardian* instalado, para efeitos de comparação de quão era a interferência no sistema.

As Figuras 26, 27 e 28 apresentam gráficos comparativos utilizando os protocolo TCP e UDP. Nos gráficos é possível perceber uma pequena perda de desempenho com a utilização do *Netperf* para realização dos testes em máquinas que estão protegidas pelo *XenGuardian*. No entanto, a perda obtida não é prejudicial ao desempenho geral do sistema, justificando assim, o uso efetivo da solução proposta.

6.2.3 Vantagens e desvantagens

Como vantagem apresentada pela solução desenvolvida, podemos citar todas aquelas pertinentes às facilidades proporcionadas pela virtualização: compatibilidade de *software*, isolamento, encapsulamento e alto desempenho. Porem ainda ser acrescentadas as vantagens obtidas com o uso de sistemas detectores de intrusões: detecção de ataques sem conhecê-los em detalhes e sem gerar excessivos falsos-positivos, facilidade de criação de novas assinaturas de acordo com a necessidade e facilidade de atualização das assinaturas.

Entretanto, implementar um sistema de detecção de intrusos que utilize o *XenGuardian* pode apresentar desvantagens e tende a ser um processo trabalhoso dependendo da quantidade de equipamentos e/ou máquinas virtuais que se deseja proteger. Cada máquina *domU* além de possuir o *Snort* previamente instalado e configurado, deve também possuir uma versão cliente do *XenGuardian*, configurada para realizar a comunicação com o *XenGuardian* servidor, que se encontra no *dom0*.

6.3 Conclusão do capítulo

Para validar a solução proposta, procurou-se pesquisar os principais *exploits* existentes no mercado e realizar testes massivos contra a ferramenta que foi desenvolvida. Para analisar a interferência no sistema, utilizou-se o *Netperf*, famoso *benchmark* de redes, realizando testes com os protocolos TCP e UDP, o primeiro realiza comunicação mais confiável, com confirmação de recebimento de pacotes no destino, ordenação de mensagens e reenvio de pacotes em caso de perda. Foram realizados testes em máquinas virtuais com e sem a solução *XenGuardian*, para efeito de comparação de desempenho entre diferentes protocolos de transporte.

A proposta apresentada insere uma perda não-significativa no desempenho global do sistema, evitando que a ferramenta desenvolvida possa afetar a qualidade de outros serviços que estejam em execução. Isto torna o *XenGuardian* um ótimo aditivo para sistemas que utilizam a virtualização através da máquina virtual Xen e zelam pela segurança em primeiro lugar.

7 Conclusões e trabalhos futuros

É notório que cada vez mais a virtualização de servidores está presente na vida de administradores de sistemas e também inicia sua presença junto à usuários domésticos. Enquanto o poder computacional aumenta, os recursos computacionais tendem a ficar grande parte do tempo ociosos. A virtualização nasceu para otimizar o uso dos recursos disponíveis, permitindo realizar um número maior de tarefas com a mesma capacidade de processamento. Delegar para usuários a tarefa de gerenciar a segurança de um ambiente virtualizado pode causar falhas de segurança, além de gerar transtornos pela falta de conhecimento de técnicas de proteção em sistemas interligados via rede.

Neste trabalho foi apresentada uma nova estratégia para reforçar a segurança de sistemas virtualizados, a qual se baseia no uso do monitor de máquina virtual Xen. O Xen tornou-se muito popular graças a sua confiabilidade, desempenho, segurança e, principalmente, por ter seu código fonte aberto, baseado nas políticas do *software* livre. Para reforçar a segurança da máquina virtual, faz-se necessário a inclusão de uma camada extra de segurança, a qual será responsável por realizar a leitura dos pacotes que chegam da rede externa para a rede interna. Cada pacote é analisado individualmente e através de consulta em uma base de conhecimento ele poderá ser descartado caso haja alguma suspeita de ataque ou pacote malformado, ou ainda poderá ser aceito caso esteja dentro da normalidade.

Essa camada de abstração possui um software executando permanentemente e em *background*, fruto deste trabalho, o qual recebeu o nome de *XenGuardian*, esta ferramenta é um protótipo, cuja finalidade é reforçar a segurança de sistemas virtualizados com a máquina virtual Xen.

No decorrer do texto o tratamento que ocorre para cada tipo de pacote foi abordado e uma arquitetura geral do *XenGuardian* foi apresentada. O modelo de proteção baseia-se em regras pré-determinadas que podem ser facilmente incluídas conforme a necessidade do usuário, tendo as regras a finalidade de barrar pacotes indesejáveis. Pelo fato do *Snort* ser um sistema de detecção amplamente difundido, fica fácil trocar informações com outros sistemas de terceiros, ampliando assim a gama de recursos que podem ser oferecidos. Já existem vários *plugins* que trabalham juntamente com o Snort na detecção de intrusão, cada um agregando funcionalidades e recursos extras, porém não havia até o momento um *plugin* que pudesse comunicar-se com máquinas virtuais para realizar filtros de pacotes, o que motivou a realização deste trabalho.

O trabalho foi finalizado apresentando um desempenho adequado à tarefa à qual se propôs realizar: a interferência introduzido no sistema pela ferramenta proposta é imperceptível e não chega a comprometer o sistema no qual está em atividade além da sua eficiência ter sido com-

provada com a detecção dos ataques para a qual foi elaborada. A proposta apresentada não exige a modificação da máquina virtual Xen, tampouco de outros aplicativos do sistema operacional. Somente faz-se necessário a instalação de alguns aplicativos extras nas máquinas virtuais *domU* e na máquina hóspede *dom0*. Os aplicativos requeridos são o *Snort* para as máquinas *domU* e a biblioteca *libvirt* para o *dom0*, pelo fato de serem *softwares* que ocupam espaço reduzido no disco, o uso de memória é mínimo e não chega a causar impacto relevante no sistema.

Como possíveis desdobramentos futuros, derivados deste trabalho, estão:

- Adaptação para outros monitores de máquinas virtuais, tais quais o UML, VMware e outros;
- Aumento na quantidade de regras disponíveis, abrangendo assim uma variedade maior de ataques;
- Otimização para redes com grande fluxo de pacotes;
- Adaptação para funcionar em outros sistemas operacionais derivados do Linux;
- Analisar outros indicativos do sistema que podem sinalizar ataques, tais como: *logs*, início e término de programas, níveis de uso de CPU, quantidade de usuários conectados no sistema, etc.

A inclusão de outras funcionalidades ao *XenGuardian*, além daquelas apresentadas neste trabalho, poderão contribuir para uma eficácia ainda maior da ferramenta. Pelo fato de novos ataques a redes de computadores serem descobertos a cada dia, é de fundamental importância a atualização das regras da ferramenta, para se adequar às novas realidades. A busca e o combate a intrusos em sistemas computacionais é uma arte que não tem fim.

Referências

[1] Projeto Honeypot Brasil. Disponível em http://www.honeypot.com.br/, acessado em 02/08/2007.

[2] Credit-Based CPU Scheduler. Disponível em http://wiki.xensource.com/xenwiki /CreditScheduler, acessado em 14/07/2007.

[3] Data Center World. Disponível em http://www.datacenterworld.com, acessado em 16/09/2007. Dallas, Texas.

[4] ACID. Analysis Console for Intrusion Databases. Disponível em http://acidlab.sourceforge.net, acessado em 02/08/2007.

[5] I. Advanced Micro Devices. Amd releases - pacifica specification for amd64 technology. Disponível em http://www.amd.com/us-en/Weblets/0,,7832_8366_7595˜98372,00.html, acessado em 20/05/2007.

[6] K. Asrigo, L. Litty, and D. Lie. Using VMM-based sensors to monitor honeypots. In *VEE '06: Proceedings of the 2nd international conference on Virtual execution environments*, pages 13–23, New York, NY, USA, 2006. ACM Press.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *SOSP '03: Proceedings of the nineteenth ACM Symposium on Operating Systems Principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[8] BASE. Basic Analysis and Security Engine. Disponível em http://base.secureideas.net, acessado em 02/08/2007.

[9] F. Bellard. QEMU, a Fast and Portable Dynamic Translator. In *USENIX Annual Technical Conference*, pages 41–46, Anaheim, CA, USA, 2005. ACM Press.

[10] F. Bellard. QEMU Accelerator Technical Documentation. Disponível em http://fabrice.bellard.free.fr/qemu/kqemu-tech.html, acessado em 30/11/2007.

[11] S. M. Bellovin. Virtual machines, virtual security? *Commun. ACM*, 49(10):104, 2006.

[12] P. Bueno. Paranoid penguin: Understanding IDS for Linux. *Linux J.*, 2002(97):11, 2002.

[13] G. Carnevale. Microsoft Virtual PC. Disponível em http://www.microsoft.com/brasil/technet/Colunas/GuilhermeCarnevale/VirtualPC.mspx, acessado em 18/05/2007.

[14] coLinux. Getting Started with coLinux. Disponível em http://wiki.colinux.org/mediawiki, acessado em 21/05/2007.

[15] E. O. S. Consulting. Enomalism Virtualized Management Dashboard. Disponível em http://www.enomalism.com/, acessado em 30/11/2007.

[16] Corporation Intel. Ia-32 Intel Architecture Software Developer Manual. Volume 1: Basic Architecture. Section 4.5. ftp://download.intel.com/design/Pentium4/manuals/25366521.pdf, acessado em 10/11/2007.

[17] S. Crosby and D. Brown. The virtualization reality. *Queue*, 4(10):34–41, 2007.

[18] R. Davoli. Teaching operating systems administration with user mode linux. In *ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, pages 112–116, New York, NY, USA, 2004. ACM Press.

[19] D. E. Denning. An intrusion-detection model. *IEEE Trans. Softw. Eng.*, 13(2):222–232, 1987.

[20] A. M. Devices. AMD I/O Virtualization Technology (IOMMU) Specification. http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/34434.pdf, acessado em 23/05/2007.

[21] J. Dike. A user-mode port of the linux kernel. Proceedings of the 4th Annual Linux Showcase & Conference, Atlanta, Abril 2006.

[22] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (bvt) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 261–276, New York, NY, USA, 1999. ACM Press.

[23] T. Eastep. Xen and Shorewall. Disponível em http://shorewall.net/Xen.html, acessado em 03/05/2007.

[24] H. Eiraku and Y. Shinjo. Running bsd kernels as user processes by partial emulation and rewriting of machine instructions. In *BSDC'03: Proceedings of the BSD Conference 2003 on BSD Conference*, pages 10–10, Berkeley, CA, USA, 2003. USENIX Association.

[25] B. B. J. Hall. Beej's Guide to Network Programming. Disponível em http://beej.us/guide/bgnet, acessado em 27/06/2007.

[26] R. Hat. Virtual Machine Manager. Disponível em http://virt-manager.et.redhat.com, acessado em 30/11/2007.

[27] Hewlett Packard Company. Netperf: A Network Performance Benchmark. Disponível em http://www.netperf.org/netperf/training/Netperf.html, acessado em 07/06/2007.

[28] C. Hoepers, K. S. Jessen, and M. H. P. C. Chaves. Honeypots e Honeynets: Definições e Aplicações. Disponível em http://www.cert.br/docs/whitepapers/honeypots-honeynets, acessado em 16/08/2007.

[29] Hping2. Hping Network Security Tool. Disponível em http://wiki.hping.org, acessado em 02/08/2007.

[30] S. Hu and J. E. Smith. Using dynamic binary translation to fuse dependent instructions. In *CGO '04: Proceedings of the international symposium on Code generation and optimization*, page 213, Washington, DC, USA, 2004. IEEE Computer Society.

[31] IDSwakeup. IDSwakeup: false positive generator. Disponível em http://www.hsc.fr/ressources/outils/idswakeup, acessado em 02/08/2007.

[32] S. Iren, P. D. Amer, and P. T. Conrad. The transport layer: tutorial and survey. *ACM Comput. Surv.*, 31(4):360–404, 1999.

[33] S. T. King, G. W. Dunlap, and P. M. Chen. Operating system support for virtual machines. In *ATEC'03: Proceedings of the USENIX Annual Technical Conference 2003 on USENIX Annual Technical Conference*, pages 6–6, Berkeley, CA, USA, 2003. USENIX Association.

[34] H. Koike and K. Ohno. Snortview: visualization system of snort logs. In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 143–147, New York, NY, USA, 2004. ACM Press.

[35] M. Laureano, C. Maziero, and E. Jamhour. Intrusion Detection in Virtual Machine Environments. In *EUROMICRO '04: Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04)*, pages 520–525, Washington, DC, USA, 2004. IEEE Computer Society.

[36] K. Lawton, B. Denney, and N. D. G. et al. Bochs User Manual. Disponível em http://bochs.sourceforge.net/doc/docbook, acessado em 20/05/2007.

[37] K. P. Lawton. Bochs: A portable pc emulator for unix/x. *Linux J.*, 1996(29es):7, 1996.

[38] Libvirt. Libvirt - The Virtualization API. Disponível em http://libvirt.org, acessado em 20/08/2007.

[39] S. E. Madnick and J. J. Donovan. Application and analysis of the virtual machine approach to information system security and isolation. In *Proceedings of the workshop on virtual computer systems*, pages 210–224, New York, NY, USA, 1973. ACM Press.

[40] S. McDowell and E. Wahlig. Amd Pacifica Virtualization Technology. AMD Reviewer's Day in Austin Texas, acessado em 29/11/2007 http://www.amdboard.com/pacifica_033005.html.

[41] Network Working Group. Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks - RFC 2884. Disponível em http://tools.ietf.org/html/rfc2884, acessado em 20/07/2007.

[42] S. Northcutt and S. Northcult. *Network Intrusion Detection: An Analyst's Handbook*. New Riders Publishing, Thousand Oaks, CA, USA, 1999.

[43] C. Peikari and A. Chuvakin. *Security Warrior*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004.

[44] I. Pratt, D. Magenheimer, and P. Barham. Xen 3.0 and the Art of Virtualization. In *Proceedings of the Linux Symposium*, pages 65–77, 2005.

[45] I. Red Hat. *Guia de Segurança: Red Hat Enterprise Linux 4*. Red Hat, Inc., USA, 2005.

[46] R. Rosen. Introduction to the Xen Virtual Machine. Disponível em http://www.linuxjournal.com/article/8540, acessado em 15/05/2007.

[47] M. Rosenblum. The reincarnation of virtual machines. *Queue*, 2(5):34–40, 2004.

[48] K. Sadasivam, B. Samudrala, and T. A. Yang. Design of network security projects using honeypots. *Journal of Computing Sciences in Colleges*, 20(4):282–293, 2005.

[49] Q. Snell, A. Mikler, and J. Gustafson. Netpipe: A Network Protocol Independent Performance Evaluator. Disponível em http://www.scl.ameslab.gov/netpipe/paper/full.html, acessado em 09/06/2007.

[50] Snort. Snort IDS. Disponível em http://www.snort.org, acessado em 02/08/2007.

[51] P. Team. Plex86 Virtual Machine Program. Disponível em http://www.plex86.org, acessado em 29/05/2007.

[52] G. Torres and C. Lima. Como funciona a tecnologia de virtualização da intel. Disponível em http://www.clubedohardware.com.br/artigos/1144/2, acessado em 15/05/2006.

[53] VMware. VMware Products. Disponível em http://www.vmware.com/products/, acessado em 14/04/2007.

[54] VMware. A Performance Comparison of Hypervisors. www.vmware.com/pdf/hypervisor_performance.pdf, acessado em 15/05/2007.

[55] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 148–162, New York, NY, USA, 2005. ACM Press.

[56] XenMan. Xenman - Open Source Virtualization Management. Disponível em http://xenman.sourceforge.net, acessado em 30/11/2007.

[57] XenSource. A Performance Comparison of Commercial Hypervisors. http://blogs.xensource.com/rogerk/wp-content/uploads/2007/03/hypervisor_performance_comparison_1_0_5_with_esx-data.pdf, acessado em 02/02/2007.