Análise de Séries Temporais Multivariadas para Detecção de Intrusão em Nuvens Computacionais

Mayk Fernando Choji[1], **Paulo Lício de Geus**[1]

[1]Instituto de Computação
Universidade Estadual de Campinas (UNICAMP)
Campinas, SP – Brasil

{mayk,paulo}@lasca.ic.unicamp.br

Abstract. *Despite the numerous advantages offered by cloud computing services, security aspects constitute a critical issue when considering to adopt this kind of service. Novel intrusion detection methods have been proposed in order to mitigate attacks toward the several layers from cloud architectures. In this paper, we present an intrusion detection system for IaaS clouds. Our approach relies on the analysis of performance metrics gathered by virtual machine introspection techniques, which allows one to monitor clients directly from the host machine. Anomaly detection is performed by treating the collected data as multivariate time series.*

Resumo. *Apesar das inúmeras vantagens oferecidas por serviços de computação em nuvem, aspectos de segurança constituem um dos pontos críticos ao se considerar adotar este tipo de serviço. Novos métodos de detecção de intrusão têm sido propostos para mitigar ataques direcionados às várias camadas da arquitetura de nuvem. Neste trabalho, apresentamos um sistema de detecção de intrusão para nuvens do modelo IaaS. A abordagem baseia-se na análise de métricas de desempenho obtidas por meio de técnicas de introspecção de máquina virtual, permitindo a monitoramento de clientes a partir da máquina hospedeira. Anomalias são identificadas tratando-se os dados coletados como séries temporais multivariadas.*

1. Introdução

Embora vários estudos tenham sido realizados sobre técnicas de detecção de intrusão [Liao et al. 2013, Zhu and Zheng 2008], a chegada de novos paradigmas como a computação em nuvem traz consigo novos desafios na área de segurança da informação. A dinamicidade intrínseca dessas arquiteturas, somada a fatores de privacidade e confidencialidade de dados entre cliente e provedor de serviço, por exemplo, faz com que sistemas de segurança tradicionais tenham que ser repensados para atender às novas demandas.

Neste trabalho, utilizamos métricas de desempenho coletadas de máquinas virtuais (VMs) de forma não-invasiva para detectar anomalias de segurança utilizando um algoritmo baseado em percurso aleatório sobre grafo. Nosso objetivo é apresentar um sistema de detecção de intrusão capaz de identificar anomalias em máquinas virtuais utilizando somente dados coletados a partir do *hypervisor* e que não tenha a necessidade de submeter o sistema a testes de ataque definidos *a priori*. Esta abordagem visa garantir a

integridade do sistema e a continuidade dos serviços contratados por clientes de nuvens computacionais.

O conjunto de métricas coletadas ao longo do tempo é tratado como uma série temporal multivariada, e a indução de um grafo pela matriz *kernel* desses dados possibilita descrever cada observação em termos de sua conectividade, ou semelhança, com as demais observações. Partindo do pressuposto de que ataques causem perturbações no desempenho da máquina virtual, observamos que essas ocorrências mapeiam-se como nós com baixa conectividade no grafo, destacando as anomalias. O intervalo de coleta de dados, métricas utilizadas e níveis de segurança são controlados pela arquitetura de gerenciamento de níveis de acordo de serviço (SLA) introduzida em [Ferreira 2013].

O restante deste artigo está organizado da seguinte forma. Na Seção 2 descrevemos a arquitetura de nuvem e a técnica utilizada para detecção de anomalias. Além disso, descrevemos o método para geração de tráfego *web* adotado nos testes. Detalhes sobre os sistemas computacionais e os parâmetros do sistema de detecção de intrusão (IDS) são descritos na Seção 3. Na Seção 4 mostramos os resultados obtidos pelo método implementado ao analisarmos ataques de *port scan* e ataques distribuídos de negação de serviço (*DDoS*). Uma breve discussão sobre trabalhos anteriores é feita na Seção 5, e a Seção 6 conclui este trabalho e indica pesquisas futuras.

2. Metodologia

Nesta Seção, apresentamos a metodologia utilizada neste trabalho, incluindo a organização de máquinas físicas e virtuais, e as principais ferramentas utilizadas.

2.1. Arquitetura

Conforme esquematizado na Figura 1, a arquitetura lógica do ambiente utilizado neste trabalho pode ser decomposta em três componentes principais. O primeiro é formado pelos nós que compõem a nuvem de infraestrutura como serviço (*IaaS*). Aqui, utilizamos a plataforma *OpenStack (release Juno)* com um nó rodando os serviços de controle e um nó rodando os serviços de virtualização (e.g. *nova-compute*), que é onde são alocadas as máquinas virtuais dos clientes da nuvem.

O segundo componente representa a arquitetura de monitoramento de níveis de acordo de serviço (*Security-SLA*) apresentada em [Ferreira 2013]. Nesta arquitetura encontra-se um elemento que atua como gerenciador dos componentes de monitoramento (*sla-manager*), um nó que oferece os serviços de gerenciamento por meio de mensagens SOAP (*sla-server*), os agentes de monitoramento em si, e uma base de dados onde são armazenadas informações sobre clientes, SLAs, métricas coletadas, entre outras.

O terceiro e último componente é formado pelas máquinas responsáveis por gerar tráfego em direção a uma VM, conforme detalhado na Seção 2.2, e executar os ataques contra ela. Uma *botnet* baseada no protocolo IRC foi desenvolvida para facilitar a sincronização da geração de tráfego e dos ataques a partir de um ponto central, e também facilitar a escalabilidade dos testes. Os nós pertencentes a este componente rodam uma aplicação cliente (*bot*) que se conecta a um servidor IRC, o qual atua como um servidor de comando e controle (C&C) da *botnet*. Os comandos para os *bots* devem ser passados por um usuário específico a partir de qualquer computador conectado a este servidor IRC.

Dado o aspecto modular desta arquitetura, o IDS pode ser inserido em qualquer região, desde que tenha acesso ao banco de dados que armazena os dados coletados das VMs a serem analisadas. Por este motivo ele não foi atrelado a nenhum dos três componentes descritos anteriormente. Uma vantagem direta desta flexibilidade é que este IDS torna-se menos vulnerável a ataques externos, se comparado com aplicações que obrigatoriamente devem ser instaladas no sistema analisado—e eventualmente comprometido.

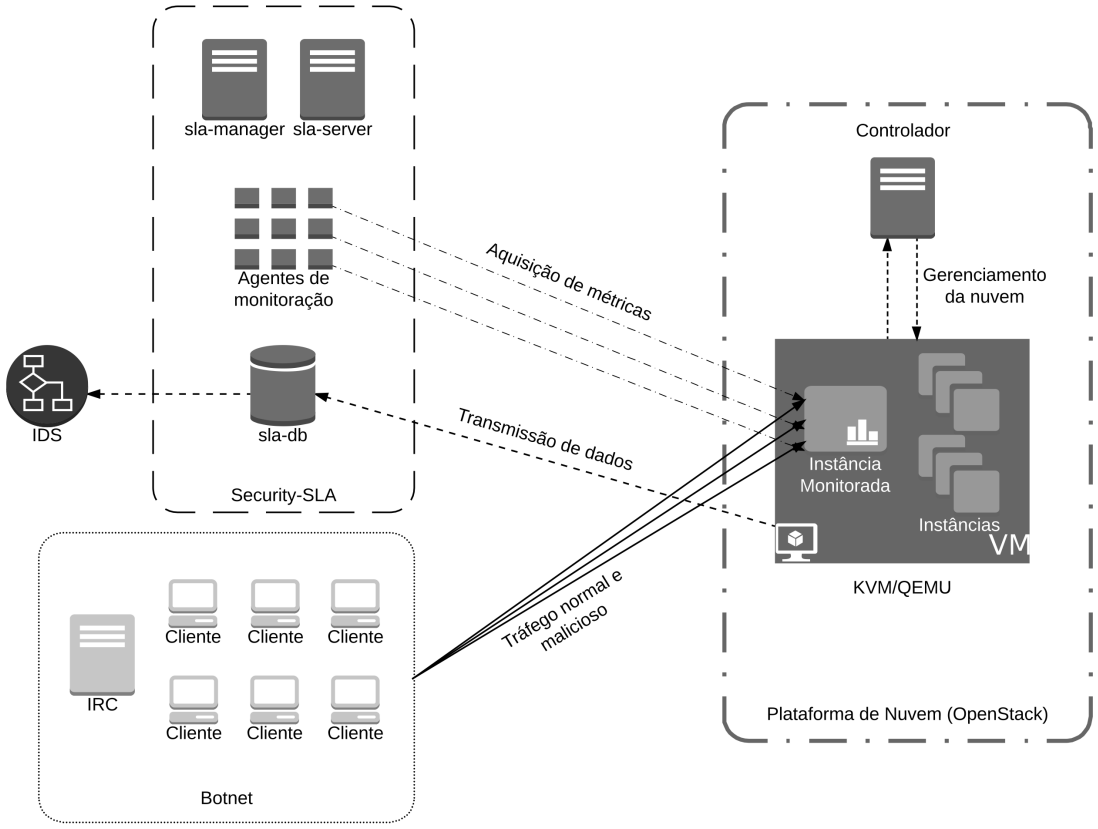

Figura 1. Arquitetura lógica do ambiente

2.2. Geração de Tráfego

A análise do IDS proposto em um cenário de estudo realístico é uma das principais preocupações deste trabalho. Mais especificamente, desejamos analisar o impacto de ataques contra um servidor *web* em termos de variação nas medidas de desempenho. Neste aspecto, é importante que a máquina virtual analisada apresente características realísticas. No caso de requisições HTTP, portanto, devemos nos preocupar com fatores como tamanho dos arquivos transmitidos, quantidade de arquivos transmitido por sessão, intervalo entre sessões etc.

Diversos estudos abordam o problema de geração de tráfego, como [Megyesi et al. 2015, Vishwanath and Vahdat 2009, Avallone et al. 2004, Mah 1997]. Em nossos testes, adotamos a ferramenta *SURGE* (abreviação para *Scalable URL Reference Generator*) [Barford and Crovella 1998] para gerar carga de trabalho sobre a máquina virtual alvo. Esta ferramenta é capaz de gerar tráfego HTTP/1.0 e HTTP/1.1 se-

guindo modelos estatísticos, levando o servidor *web* a operar como se estivesse recebendo requisições de usuários reais.

2.3. Agentes e Métricas de Desempenho

Todos as métricas utilizadas para monitorar a máquina alvo são coletadas por agentes externos à ela. Neste trabalho, apoiamo-nos em agentes baseados em técnicas de introspecção de máquina virtual (VMI), comunicação com *hypervisor* e sistema operacional (SO) do hospedeiro. Graças a essa abordagem, nenhum *software* precisa ser instalado nos sistemas convidados (*guest*), garantindo que o IDS possa continuar sua execução normal mesmo que um *guest* seja comprometido.

Estendendo o trabalho de [Ferreira 2013], os agentes foram refatorados para melhoria de desempenho, suportar outras plataformas e coletar novas métricas—rdbytes_persec, wrbytes_persec, tcp_reset_in_persec, tcp_reset_out_persec e tcp_currestab_persec. Os agentes baseados em VMI utilizam agora a versão 0.11 da biblioteca *libVMI* e são compatíveis com a plataforma de nuvem *OpenStack*. Apesar da implementação atual dos agentes CPU e IO suportar apenas os *hypervisors Xen* e *KVM/QEMU*, outros modelos podem ser facilmente adicionados à arquitetura. Na Tabela 1 estão relacionados os nomes dos agentes e a técnica de coleta de dados utilizada por cada um, e na Tabela 2 constam as métricas utilizadas neste trabalho.

Tabela 1. Método de coleta utilizado por cada agente

Agente	Método
CPU	Hypervisor
IO	Hypervisor
Memory	VMI
MIB	VMI
NetStatistics	SO
Netfilter	SO
Process	VMI

2.4. Sistema de Detecção de Intrusão

O conjunto das medidas de desempenho coletadas de uma VM ao longo do tempo pode ser visto como uma série temporal multivariada. Assim, utilizamos o algoritmo baseado em grafo para detecção de anomalias em séries temporais multivariadas proposto por [Cheng et al. 2008], descrito brevemente a seguir.

Seja $X = x_1, x_2, \ldots, x_T$ uma série temporal constituída de uma sequência de medidas para uma variável X nos intervalos de tempo $1, 2, \ldots, T$. Uma série temporal multivariada $\mathcal{D} = \{X_i\}_{i=1}^{p}$ é uma coleção de séries temporais correspondentes a medidas de p variáveis coletadas no mesmo intervalo de tempo [Cheng et al. 2008].

Uma matriz de similaridade—ou matriz *kernel*—K define o grau de similaridade entre cada par de observações de uma série temporal. A similaridade entre dois pontos i e j quaisquer é dada pela Equação 1—função de base radial (RBF) Gaussiana:

$$K(i,j) = \exp[-\frac{\sum_{k=1}^{p}(x_{ik} - x_{jk})^2}{2\sigma^2}], \quad (1)$$

Tabela 2. Descrição das métricas de desempenho

Agente	Métrica	Descrição
CPU	utilization	Uso de CPU
[4*IO]	rdbytes_persec	Média de bytes lidos por segundo
	rdreq_persec	Média de requisições de leitura por segundo
	wrbytes_persec	Média de bytes escritos por segundo
	wrreq_persec	Média de requisições de escrita por segundo
[3*Memory]	pfmajor_psec	Média de *major page fault* por segundo
	pfminor_pesec	Média de *minor page fault* por segundo
	phys_perc_usage	Percentagem de memória utilizada
[2*MIB]	[2*tcp_currestab_persec]	Média de conexões TCP em estado ESTABLISHED ou CLOSE_WAIT por segundo
[2*NetStatistics]	traffic_in	Média de tráfego de entrada por segundo
	traffic_out	Média de tráfego de saída por segundo
[4*Netfilter]	tcp_reset_in_persec	Média de pacotes TCP de entrada com *flag* RST por segundo
	tcp_reset_out_persec	Média de pacotes TCP de saída com *flag* RST por segundo
[3*Process]	avg_wrksetsize	Média de *working set size*
	processes	Número total de processos
	threads	Número total de *threads*

onde σ é um parâmetro livre chamado largura de banda do *kernel*.

[Cheng et al. 2008] propõem tratamentos distintos dependendo se a série temporal multivariada possui uma variável alvo ou não. No nosso problema, todas as métricas possuem igual importância. Neste caso, o algoritmo considera cada uma das p variáveis como alvo em uma iteração e constrói duas matrizes *kernel*: uma utilizando apenas a variável p, chamada KY_i, e outra utilizando as $p-1$ variáveis restantes (preditores), chamada K_i. Uma vez construídas, é realizado um alinhamento de *kernel* para se obter uma matriz kernel ajustada $\widehat{K_i}$ que maximiza a correlação entre K_i e KY_i. A matriz *kernel* alinhada final $\widehat{K_\alpha}$ é obtida pelo produto Hadamard das matrizes individuais $\widehat{K_i}$. Esta matriz pode ser transformada em uma representação de grafo valorado, $\mathcal{G} = (V, E)$, onde V é o conjunto de nós—neste caso, cada observação de métricas em um instante de tempo—, e $E \subseteq V \times V$ é o conjunto de arestas.

A partir de $\widehat{K_\alpha}$, obtemos a matriz de transição S utilizando a Equação 2:

$$S(i,j) = \frac{\widehat{K_\alpha(i,j)}}{\sum_{j=1}^{n} \widehat{K_\alpha(i,j)}}, \tag{2}$$

onde i e j são os índices da linha e coluna, respectivamente, e n é a ordem da matriz $\widehat{K_\alpha}$. Cada (i,j)-ésimo elemento desta matriz representa a probabilidade de transição do nó i para o nó j.

Por fim, um passeio aleatório é executado sobre o grafo visando encontrar o grau de conectividade c de cada nó. O valor para cada nó é calculado aplicando-se a Equação 3

iterativamente até que alguma condição de parada seja atingida. Nossa implementação utiliza como critério de parada um valor de tolerância $\epsilon = 10^{-4}$ e um número máximo de iterações $m = 1000$.

$$c = \frac{d}{n} + (1 - d)Sc, \tag{3}$$

onde d é o fator de amortecimento. Em cada iteração do passeio aleatório, existe uma probabilidade d de se permanecer no nó atual, contra $1 - d$ de se visitar algum nó seguindo a matriz de transição S. A ideia principal por trás deste algoritmo, conforme descrito em [Cheng et al. 2008], é que nós anômalos serão altamente dissimilares em relação aos demais. Esses nós, portanto, serão pouco visitados durante a execução do passeio aleatório e, consequentemente, terão um baixo valor de conectividade. No contexto deste trabalho, assumimos que as perturbações causadas em uma máquina virtual por conta de atividades maliciosas resultem em nós com baixa conectividade, ou seja, anomalias.

3. Experimentos

Nesta Seção, apresentamos detalhes sobre os testes realizados, dividido por etapas.

3.1. Máquina Virtual

Criamos uma VM na nuvem privada rodando o sistema operacional *Linux Ubuntu* 10.04 LTS, com 20GB de disco, 2GB de RAM, 1GB de *swap* e 2 vCPU. Esta instância possui 2 endereços IP: um interno, visível apenas dentro da nuvem, e um externo, visível na rede local—a mesma do sistema hospedeiro e diversas outras máquinas do laboratório. Esta máquina roda os serviços *Apache* v2.2.14, *Exim* v4.77 e *MySQL Server* v5.1.73.

3.2. Geração de Tráfego

Conforme descrito na Seção 2.2, utilizamos a ferramenta *SURGE* para gerar tráfego HTTP em direção à VM analisada. Além de se valer de métodos estatísticos para controlar os tempos e as quantidades das requisições, esta ferramenta gera arquivos texto com tamanhos variáveis, também seguindo um modelo estatístico configurável, para serem hospedados no servidor *web* sob teste. A Tabela 3 mostra alguma das propriedades tratadas pela ferramenta.

Tabela 3. Algumas propriedades do tráfego HTTP simulado. A coluna *Valor* mostra um valor pré-determinado ou a distribuição de probabilidade utilizada

Propriedade	Valor
Total de arquivos	2000
Total de requisições para o arquivo mais popular	20000
Tamanho dos arquivos	Pareto e Log-Normal
Número de arquivos por objeto	Pareto
Tempo entre requisições	Pareto

Três classificações de tráfego foram consideradas neste trabalho: i) tráfego normal; ii) tráfego de sobrecarga; e iii) tráfego de ataque. O tráfego normal foi gerado utilizando-se duas máquinas. O programa *SURGE* foi executado em cada uma utilizando 2 processos, cada um com 25 *threads*, durante 5000 segundos. Este foi o tempo que encontramos como necessário para que todas, ou quase todas, as requisições estipuladas pela

ferramenta pudessem ser realizadas. Os dois clientes mantiveram-se executando o programa durante toda a duração do teste, ou seja, a VM sob estudo sempre esteve recebendo requisições HTTP. Uma terceira máquina foi utilizada para gerar tráfego de sobrecarga durante 1 hora, representando uma situação onde o servidor *web* fosse mais acessado. A mesma configuração do *SURGE* apresentada anteriormente foi utilizada neste caso. O tráfego de ataque é descrito na Seção 3.3. As configurações dos sistemas utilizados para geração de tráfego estão descritas na Tabela 4. O tráfego normal foi gerado pelos clientes #1 e #2, enquanto o tráfego de sobrecarga foi gerado pelo cliente #3.

Tabela 4. Configuração dos sistemas utilizados na geração de tráfego

Cliente	Número de processadores	RAM (GB)	Sistema Operacional
#1	2	2	Linux Ubuntu 14.04 64-bit
#2	4	4	Linux Ubuntu 14.04 64-bit
#3	6	4	Linux Ubuntu 14.04 64-bit

3.3. Ataques

Nesta etapa, executamos ataques de *port scan* e de negação de serviço (*DDoS*) contra a VM hospedada na nuvem, ao longo de um dia. Primeiro, simulamos um atacante realizando uma varredura em 65535 portas TCP. Para verificar a variação dos distúrbios causados na VM e a sensibilidade do IDS, executamos tanto uma abordagem agressiva (alta quantidade de portas por segundo) quanto uma mais segura (poucas portas por segundo). Em relação aos ataques de *DDoS*, utilizamos as máquinas relacionadas na Tabela 4 para executar ataques na camada de aplicação, com auxílio das ferramentas *hulk* e *GoldenEye*. A Tabela 5 contém os horários aproximados de início e fim de cada ataque.

Tabela 5. Tempos de início e fim de cada ataque

Ataque	Variação	Início	Fim
[2*Port Scan]	Rápido	01:15:40	01:16:30
	Lento	02:41:09	02:46:00
[1*DDoS]	Camada de Aplicação	17:24:30	17:50:30

3.4. Pré-processamento dos Dados

Visto que nosso sistema utiliza métricas com diferentes unidades e magnitudes, é necessário realizar um pré-processamento dos dados antes de eles serem enviados ao IDS. Nosso objetivo com isso é impedir que uma métrica com grande intervalo de valores afete as medidas de distâncias das observações. Assim, para cada variável, subtraímos a média e dividimos pelo desvio padrão, padronizando-as. Após a conclusão desta etapa, todas as variáveis possuem média igual a 0 e variância igual a 1.

3.5. Sistema de Detecção de Intrusão

Neste trabalho, investigamos o impacto da escolha de dois parâmetros ligados ao IDS: i) tamanho de subsequência (δ); e ii) largura de banda do *kernel* (σ). O parâmetro δ representa a quantidade de observações do conjunto de dados que são agrupadas em um subconjunto. Visto de outra forma, seu objetivo é permitir que se possa

comparar subsequências de valores ao invés de observações atômicas entre si. Com isso, o algoritmo pode identificar um sequência anômala em uma série temporal multivariada. Detalhes sobre o cálculo da matriz *kernel* para subsequências podem ser encontrados em [Cheng et al. 2008]. O segundo parâmetro tem influência na separação dos dados e sua escolha e técnicas de otimização ainda são objetos de estudo [Benoudjit et al. 2002, Wu and Wang 2009, Li et al. 2014]. Para verificar a consequência da escolha dessas propriedades, executamos o algoritmo repetidas vezes, combinando os valores de $\delta = 1, 2, 6, 10$ e $\sigma = 2^{-15}, 2^{-13}, 2^{-11}, \ldots, 2^{3}$.

4. Análise

Iniciamos nossa análise inspecionando as características das métricas e as relações entre elas. A Figura 2 representa a matriz de dispersão gerada a partir da análise de 5 métricas utilizadas durante os testes. Algumas variáveis apresentam uma relação clara entre si, como *traffic_in* e *traffic_out*, o que era esperado uma vez que o tráfego gerado nos testes é do tipo requisição-resposta (HTTP). Um ponto importante de se confirmar essa relação é que validamos a corretude dos agentes de coleta.

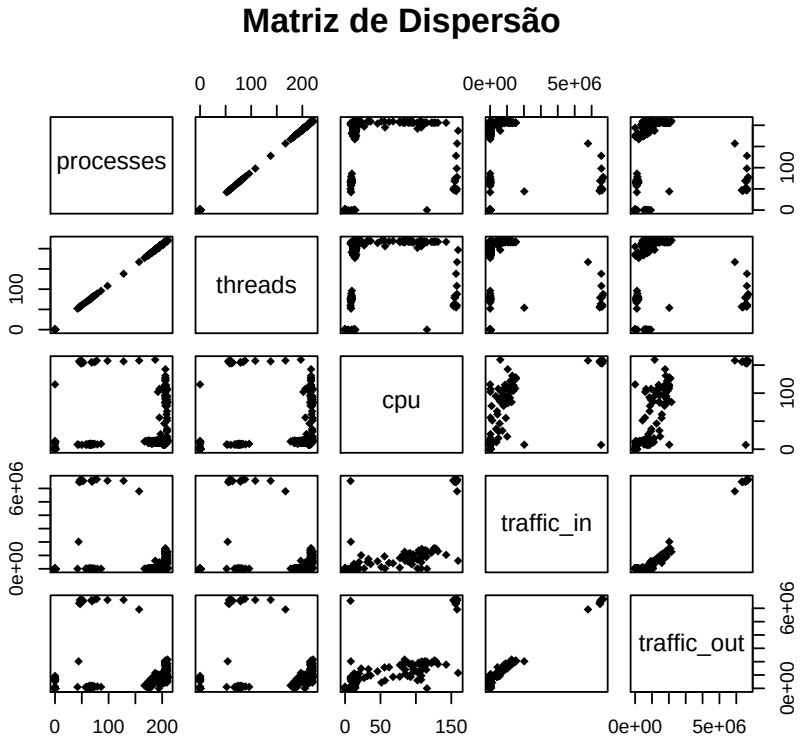

Figura 2. Matriz de dispersão das métricas *processes*, *threads*, *cpu*, *traffic_in* e *traffic_out*

A seguir, analisamos o efeito de cada ataque no desempenho da máquina virtual. Em todos os casos, analisamos intervalos de 1 hora. Após o cálculo de conectividade e padronização dos resultados, consideramos nós anômalos aqueles com valor menor ou igual a -3 (*threshold*). Este valor de *threshold* foi definido empiricamente e mostrou-se

satisfatório para os experimentos realizados. Métodos para a escolha do melhor valor e a comparação deste com outros cenários de teste são objetos de trabalhos futuros.

4.1. Identificação de Port Scan

Na Figura 3 mostramos o perfil das métricas utilizadas ao longo do período analisado. A análise dessas métricas pelo algoritmo implementado resulta em um vetor contendo o *score* (classificação) padronizado de anomalia para cada observação ou sequência da série temporal multivariada. Esses valores indicam o quanto uma observação ou sequência está distante da média em termos de desvio padrão. Na Figura 4 mostramos o *score* resultante da execução do IDS com parâmetros $\delta = 1$ e $\sigma = 2^{-5}$. O resultado indica que o *port scan* agressivo gerou distúrbios significativos na VM, identificado no instante $i = 01 : 16 : 10$. Acreditamos que a outra anomalia encontrada na análise seja resultado da inicialização dos serviços da máquina virtual, pois coincide com o tempo de *boot*. Utilizando os mesmos parâmetros, também foi possível identificar o ataque de *stealth port scan* (variação mais lenta), identificado na Figura 5 no instante $i = 02 : 42 : 40$. Analisando o gráfico de perfil deste caso, identificamos distúrbios relevantes nos instantes $i = 02 : 00 : 10$ e $i = 02 : 20 : 10$, porém não identificamos a causa.

4.2. Identificação de DDoS

Nesta etapa, repetimos o procedimento descrito na Seção 4.1 no intuito de identificar ataques de *DDoS* na camada de aplicação. Os testes para $\delta = 6$ e $\delta = 10$ não encontraram nenhuma anomalia com o *threshold* de -3 utilizado anteriormente. Para $\delta = 2$, o algoritmo encontrou uma anomalia, porém não no instante referente ao ataque. Melhores resultados foram obtidos novamente com os parâmetros $\delta = 1$ e $\sigma = 2^{-5}$. Para esta configuração, o IDS identificou corretamente os distúrbios causados pelo ataque de negação de serviço e também outras duas anomalias nos instantes $i = 16 : 55 : 43$ e $i = 17 : 04 : 13$. O *score* de anomalia está apresentados na Figura 7. Para $\sigma = 2^{-15}, 2^{-13}, \ldots, 2^{-7}$ o resultado é semelhante, apenas a anomalia no instante $i = 16 : 55 : 43$ não é encontrada.

4.3. Discussão

A análise dos resultados mostra que a escolha dos parâmetros *tamanho de subsequência* e *largura de banda do kernel* tem grande impacto sobre a detecção de anomalias. Para evitar que regiões adjacentes em um intervalo de anomalia apresentem-se como anomalias isoladas, após o cálculo do *score* de anomalia realizamos uma agregação de nós anômalos. Isto é feito agrupando-se nós referentes a instantes de tempo consecutivos em uma única notificação de anomalia, enfatizando o início e o término da sequência anômala.

5. Trabalhos Relacionados

Em [Avritzer et al. 2010], os autores descrevem um *framework* para geração de assinaturas de desempenho de máquinas *Windows* reais e um algoritmo para detectar ataques de segurança automaticamente, considerando alguns testes comuns de segurança como *Buffer Overflow* e ataques de negação de serviço. As assinaturas são definidas analisando-se dados de sistema gerados pelo *Microsoft Windows Management Instrumentation API* (WMI)—uso de CPU, memória, rede e entrada/saída (I/O). Cada experimento (teste de ataque) é conduzido em duas etapas: uma etapa de uso regular do sistema para coletar o perfil de desempenho base, e uma etapa de teste de segurança para coletar o perfil do

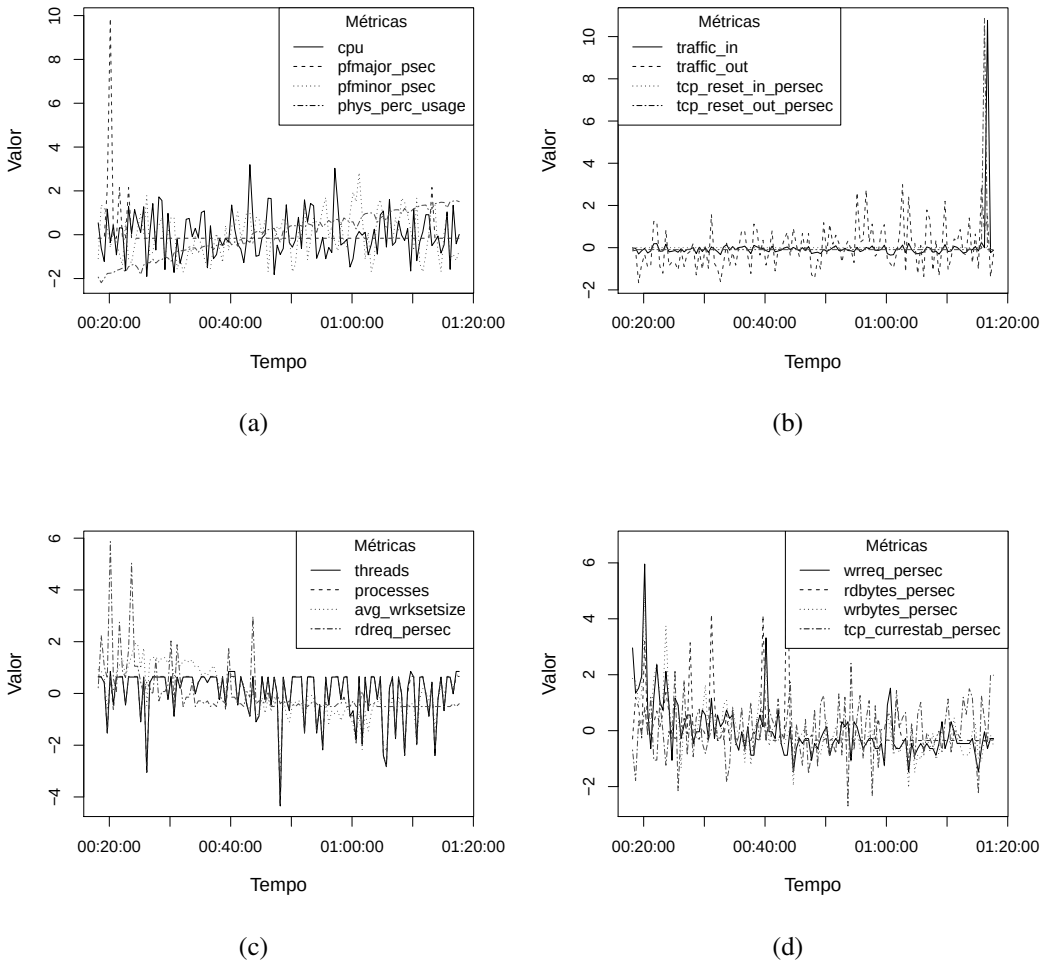

Figura 3. Perfil das métricas de desempenho durante um ataque de *port scan* agressivo

ataque. O principal problema desta abordagem é a incapacidade de se detectar novos ataques. Além disso, qualquer evento que resulte em uma alteração do perfil base impacta negativamente na capacidade do sistema identificar as ameaças mapeadas anteriormente. Neste trabalho, utilizamos uma abordagem que elimina a necessidade de se definir um perfil base *a priori* e que é capaz de detectar ataques desconhecidos. Uma extensão deste trabalho deve permitir que a anomalia identificada seja classificada em tipos específicos de ataques, aumentando a qualidade da informação apresentada para os clientes da nuvem.

Seguindo a metodologia de [Avritzer et al. 2010], [Ferreira 2013] apresenta um sistema de monitoramento de segurança baseado em acordos de níveis de serviço para nuvens de infraestrutura. No entanto, ao invés de dados coletados a partir de máquinas *Windows* reais, o autor aplica a técnica no monitoramento de máquinas virtuais rodando o sistema operacional *Linux* em um ambiente de nuvem no modelo IaaS. As métricas utilizadas para geração e análise de assinaturas são provenientes de introspecção de máquina virtual e dados coletados a partir do monitor de máquinas virtuais, ao passo que a análise das assinaturas é feita utilizando-se técnicas de aprendizado de máquina. Embora os re-

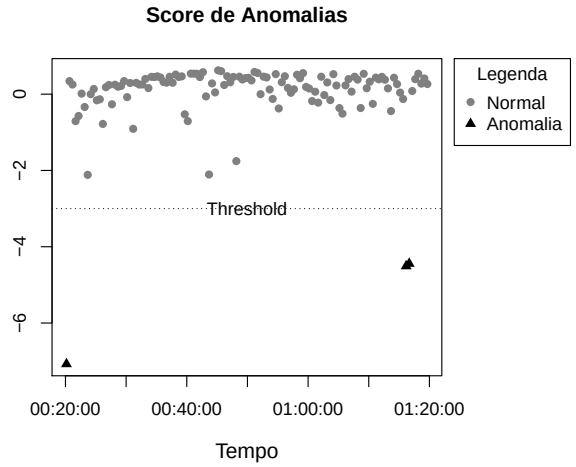

Figura 4. Score de anomalias gerado pelo IDS com parâmetros $\delta = 1$ e $\sigma = 2^{-5}$

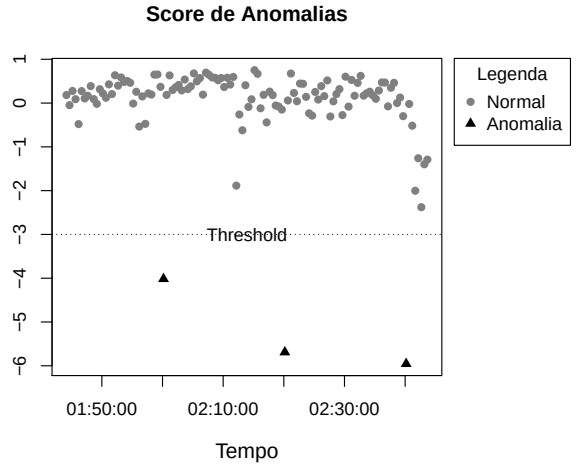

Figura 5. Score de anomalias gerado pelo IDS com parâmetros $\delta = 1$ e $\sigma = 2^{-5}$

sultados mostrem a factibilidade de se utilizar métricas de desempenho para detecção de ataques em um ambiente de nuvem, esta abordagem também requer que um perfil base seja definido *a priori*. As propriedades do tráfego de rede também não são levadas em consideração, visto que o perfil base é definido por uma carga constante na VM analisada, sem sinais de variância no intervalo entre sessões ou na quantidade de dados transferidos. Este ponto é tratado neste trabalho por meio da ferramenta *SURGE*, descrita na Seção 2.2. Os resultados da análise sobre um tráfego com mais variações reforça a expectativa de que a análise de métricas de desempenho para problemas de detecção de intrusão possa ser empregada em ambientes reais.

[Nikolai and Wang 2014] utilizam métricas coletadas a partir do *hypervisor* para identificar ataques de negação de serviço em máquinas virtuais rodando um servidor *web*. Embora os autores mostrem que é possível detectar ataques utilizando apenas 5 métricas oferecidas pelo *hypervisor* (pacotes transmitidos, pacotes recebidos, requisições de leitura e de escrita em disco, e uso de CPU), apenas uma variação do valor das métri-

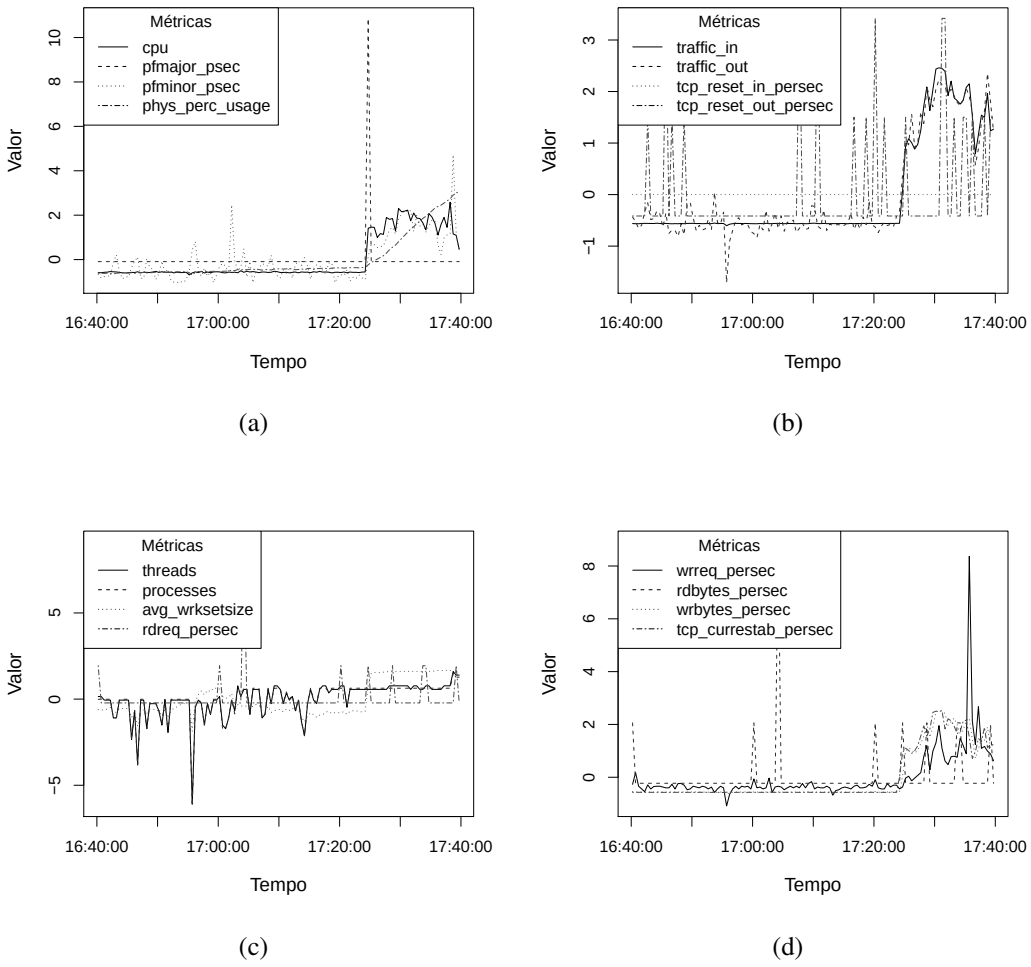

Figura 6. Perfil das métricas de desempenho durante um ataque de *DDoS* na camada de aplicação

cas em relação à média é utilizada para detectar intrusão baseada em assinaturas geradas manualmente. Além disso, a carga aplicada sobre a máquina testada também não considera aspectos realísticos de um servidor *web*, como intervalo entre sessões e número de requisições por sessão. Em nosso trabalho, utilizamos um total de 16 métricas de desempenho, e o algoritmo empregado determina a relação das métricas entre si para identificar possíveis intervalos de anomalia.

Detalhes sobre outras abordagens para sistemas de detecção de intrusão em nuvens computacionais são apresentadas em [Modi et al. 2013] e [Patel et al. 2013], notando-se que o último enfatiza a necessidade de se desenvolver sistemas de detecção e prevenção de intrusão projetados especificamente para nuvens, ao invés do uso de ferramentas tradicionais.

6. Conclusão e Trabalhos Futuros

Neste trabalho, apresentamos um sistema de detecção de intrusão baseado em anomalias para identificação de ataques contra máquinas virtuais hospedadas em uma nuvem com-

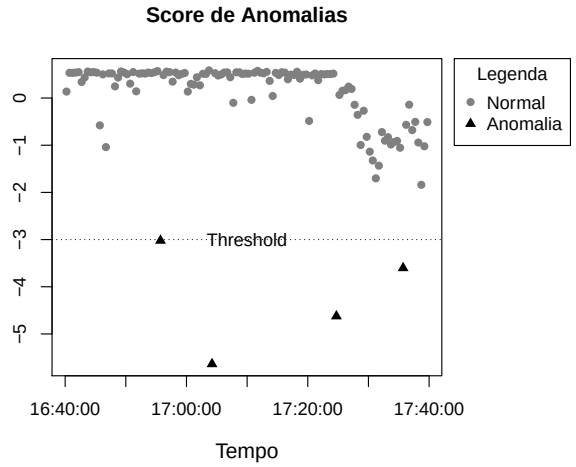

Figura 7. Score de anomalias gerado pelo IDS com parâmetros $\delta = 1$ e $\sigma = 2^{-5}$

putacional. Mostramos que o método baseado em passeio aleatório apresenta resultados satisfatórios quando escolhidos os parâmetros corretos. Ainda, tratando as métricas de desempenho coletadas como uma série temporal multivariada, nosso sistema mostrou-se capaz de detectar ataques de *port scan* e *DDoS*. Por se tratar de um método baseado em anomalias, a arquitetura é capaz de detectar ataques desconhecidos e outros distúrbios pontuais.

Embora seja possível realizar a detecção de anomalias comparando-se subsequências da série temporal, experimentos mostraram que esta abordagem não apresenta resultados satisfatórios. Um dos motivos é que, ao contrário do estudo feito por [Cheng et al. 2008], os dados de desempenho de VMs não apresentam ciclos bem definidos, além de serem altamente dependentes do fluxo de dados e *softwares* instalados.

Como trabalho futuro, pretendemos aplicar este método de detecção para avaliar uma VM rodando serviços reais, disponíveis publicamente na *Internet*, e integra-lo a um classificador baseado em assinaturas, visando mostrar para os clientes com SLA definidos não só os distúrbios mas também o tipo de ataque que cada instância sofreu. Finalmente, outros tipos de ataques e sistemas operacionais do *guest* precisam ser estudados para verificar a abrangência do método e sua robustez a ruídos, além de permitir uma análise em tempo real.

Referências

Avallone, S., Guadagno, S., Emma, D., Pescape, A., and Ventre, G. (2004). D-ITG distributed internet traffic generator. In *First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings.*, pages 316–317.

Avritzer, A., Tanikella, R., James, K., Cole, R. G., and Weyuker, E. (2010). Monitoring for security intrusion using performance signatures. In *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, WOSP/SIPEW '10, pages 93–104, New York, NY, USA. ACM.

Barford, P. and Crovella, M. (1998). Generating representative web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGME-*

TRICS Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '98/PERFORMANCE '98, pages 151–160, New York, NY, USA. ACM.

Benoudjit, N., Archambeau, C., Lendasse, A., Lee, J. A., Verleysen, M., et al. (2002). Width optimization of the gaussian kernels in radial basis function networks. In *ESANN*, volume 2, pages 425–432.

Cheng, H., Tan, P. N., Potter, C., and Klooster, S. (2008). A robust graph-based algorithm for detection and characterization of anomalies in noisy multivariate time series. In *2008 IEEE International Conference on Data Mining Workshops*, pages 349–358.

Ferreira, A. S. (2013). Uma arquitetura para monitoramento de segurança baseada em acordos de níveis de serviço para nuvens de infraestrutura. Master's thesis, Universidade Estadual de Campinas, Instituto de Computação, Campinas, SP, Brasil.

Li, J.-B., Wang, Y.-H., Chu, S.-C., and Roddick, J. F. (2014). Kernel self-optimization learning for kernel-based feature extraction and recognition. *Information Sciences*, 257:70–80.

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., and Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24.

Mah, B. A. (1997). An empirical model of http network traffic. In *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, volume 2, pages 592–600.

Megyesi, P., Szabó, G., and Molnár, S. (2015). User behavior based traffic emulator: A framework for generating test data for dpi tools. *Computer Networks*, 92:41–54.

Modi, C., Patel, D., Borisaniya, B., Patel, H., Patel, A., and Rajarajan, M. (2013). A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, 36(1):42–57.

Nikolai, J. and Wang, Y. (2014). Hypervisor-based cloud intrusion detection system. In *Computing, Networking and Communications (ICNC), 2014 International Conference on*, pages 989–993.

Patel, A., Taghavi, M., Bakhtiyari, K., and Júnior, J. C. (2013). An intrusion detection and prevention system in cloud computing: A systematic review. *Journal of Network and Computer Applications*, 36(1):25–41.

Vishwanath, K. V. and Vahdat, A. (2009). Swing: Realistic and responsive network traffic generation. *IEEE/ACM Trans. Netw.*, 17(3):712–725.

Wu, K.-P. and Wang, S.-D. (2009). Choosing the kernel parameters for support vector machines by the inter-cluster distance in the feature space. *Pattern Recognition*, 42(5):710–717.

Zhu, Y. and Zheng, Y. (2008). Research on intrusion detection system based on pattern recognition. In *Networked Computing and Advanced Information Management, 2008. NCM '08. Fourth International Conference on*, volume 1, pages 609–612.