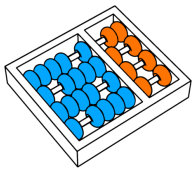
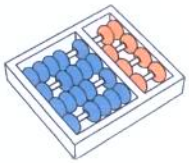


Anderson Soares Ferreira

“Uma arquitetura para monitoramento de segurança baseada em acordos de níveis de serviço para nuvens de infraestrutura”

CAMPINAS
2013



Universidade Estadual de Campinas
Instituto de Computação

Anderson Soares Ferreira

"Uma arquitetura para monitoramento de segurança baseada em acordos de níveis de serviço para nuvens de infraestrutura"

Orientador(a): **Prof. Dr. Paulo Lício de Geus**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Ciência da Computação.

Este exemplar corresponde à versão final da Dissertação defendida por Anderson Soares Ferreira, sob orientação de Prof. Dr. Paulo Lício de Geus.

Assinatura do Orientador(a)

CAMPINAS
2013

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

Ferreira, Anderson Soares, 1973-
F413a Uma arquitetura para monitoramento de segurança baseada em acordos de níveis de serviço para nuvens de infraestrutura / Anderson Soares Ferreira. – Campinas, SP : [s.n.], 2013.

Orientador: Paulo Lício de Geus.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

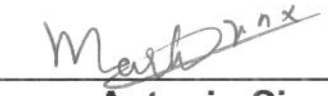
1. Computação em nuvem. 2. Sistemas de segurança. 3. Virtualização. I. Geus, Paulo Lício de,1956-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

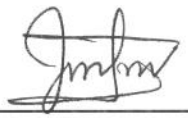
Título em outro idioma: An architecture for security monitoring based on service level agreements for cloud infrastructure services
Palavras-chave em inglês:
Cloud computing
Security systems
Virtualization
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Paulo Lício de Geus [Orientador]
Marcos Antônio Simplício Júnior
Julio Cezar López Hernández
Data de defesa: 16-09-2013
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 16 de setembro de 2013, pela Banca examinadora composta pelos Professores Doutores:



Prof. Dr. Marcos Antonio Simplicio Junior
LARC / USP



Prof. Dr. Julio César López Hernández
IC / UNICAMP

Prof. Dr. Paulo Lício de Geus
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Uma arquitetura para monitoramento de segurança baseada em acordos de níveis de serviço para nuvens de infraestrutura

Anderson Soares Ferreira

16 de setembro de 2013

Banca Examinadora:

- Prof. Dr. Paulo Lício de Geus (*Orientador*)
- Prof. Dr. Marcos Antônio Simplício Júnior
  Laboratório de Arquitetura e Redes de Computadores - USP
- Prof. Dr. Julio Cesar López Hernández
  Instituto de Computação - UNICAMP
- Prof. Dr. Antônio Marinho Pilla Barcellos
  Instituto de Informática - UFRGS (*Suplente*)
- Prof. Dr. Ricardo Dahab
  Instituto de Computação - UNICAMP (*Suplente*)

Resumo

Apesar do extensivo uso da computação em nuvens na atualidade, ainda há um grande número de organizações que optam por utilizar arquiteturas computacionais tradicionais por considerarem esta tecnologia não confiável, devido a problemas não resolvidos relacionados a segurança e privacidade. A garantia da segurança em ambientes de nuvens é atualmente perseguida através de acordos de níveis de serviço de segurança; apesar disto, o monitoramento destes acordos até agora tem sido dificultada por obstáculos técnicos relacionados a virtualização, compartilhamento de recursos e multi-locação

Visando o acompanhamento destes acordos e consequentemente a melhoria da segurança em nuvens de serviços de infraestrutura, este trabalho apresenta uma solução de monitoramento baseada em mecanismos seguros para coleta de informações. Utilizando-se técnicas como monitoramento caixa preta e introspecção, elimina-se a necessidade de instalação de ferramentas de monitoramento na máquina virtual. Através de informações de desempenho coletadas através da solução de monitoramento, apresenta-se também um estudo sobre a identificação de ataques a máquinas virtuais em ambientes de nuvens, através da detecção de anomalias de segurança.

Abstract

Despite the extensive use of cloud computing today, there is a large number of organizations that choose to stick to traditional architectures, since this technology is considered unreliable due to as yet unsolved problems related to security and privacy. Ensuring security in cloud environments is currently being pursued through security-minded service level agreements. However, monitoring such agreements has so far been hampered by technical obstacles related to virtualization, resource sharing and multi-tenancy.

Aiming at monitoring these agreements and consequently improving security in cloud infrastructure services, this work presents a monitoring architecture based on external mechanisms for collecting information. Through the use of techniques such as black-box monitoring and introspection, one eliminates the need to install monitoring tools on the virtual machine. Also, by using performance information collected by the monitoring solution, this work presents a study about the identification of attacks to virtual machines in cloud environments through anomaly detection.

Agradecimentos

Gostaria de agradecer primeiramente ao meu orientador, Professor Paulo Lício de Geus pelo apoio e aconselhamento dados durante estes anos.

Agradeço a Empresa Brasileira de Pesquisa Agropecuária - Embrapa, meu empregador, pelo investimento na minha formação científica e profissional.

Agradeço a todos os colegas de trabalhos e em especial a Ana Lúcia, Eduardo e Jaime, pelo incentivo durante este período em que estive afastado de minhas atividades profissionais.

Finalmente agradeço a minha família: aos meus pais Archimedes e Maria da Glória, minhas irmãs Cristiana e Patrícia e em especial a minha esposa Silvania pelo carinho, atenção e paciência (muita paciência) durante esta jornada.

"(hey you) Get off of my cloud!"
The Rolling Stones

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A computação em nuvens[1] é uma tecnologia que fornece aos usuários capacidade de processamento, armazenamento de dados e aplicações via rede, através de um modelo de computação utilitária [5], onde recursos computacionais e *software* são oferecidos como serviços e pagos por uso (*pay per use*). Ela fornece aos seus consumidores acesso a recursos computacionais que não estariam disponíveis em arquiteturas tradicionais, devido à complexidade técnica e aos altos custos envolvidos [54].

Da mesma forma que a utilização de serviços em nuvens é crescente entre usuários domésticos, os quais utilizam tais serviços para comunicação pessoal (redes sociais e correio eletrônico), armazenamento e edição de arquivos etc., também é grande a sua adoção por parte de empresas e governos. Segundo [58], estima-se que a utilização da computação em nuvens pelo governo dos Estados Unidos tenha um crescimento de 16%, com investimentos de US$ 47 bilhões, no período entre 2013 e 2018. De modo semelhante, o crescimento mundial do mercado de computação em nuvens no último ano foi de 18%, com valor estimado em US$ 9 bilhões e previsão de gastos de US$ 610 bilhões até 2016 [37].

Apesar do crescimento deste mercado e dos maciços investimentos, ainda é grande a preocupação com a segurança nestes ambientes. Segundo [31], as questões relacionadas à segurança são consideradas como principal fator impeditivo para a migração de serviços para ambientes de nuvem.

A crescente preocupação e insatisfação com a segurança em serviços de nuvens é resultado de uma combinação de diversos fatores, dentre os quais podem ser citados: a falta de conhecimento das características técnicas e riscos existentes em ambientes de nuvem [16, 88]; a falta de padrões de interoperabilidade bem definidos [43]; a perda de controle de dados e aplicações [54]; as falhas ocorridas em nuvens computacionais que resultaram

[1]Devido à diversidade de termos utilizados para designar a tecnologia, este trabalho utiliza a seguinte convenção: o termo "computação em nuvens" refere-se à tecnologia de modo geral, enquanto que "nuvem computacional" ou simplesmente "nuvem" referem-se a um serviço de nuvem específico ou a implementações da tecnologia.

em indisponibilidade de serviços, perda de dados e vazamento de informações [88]; e a falta de garantias relacionadas ao nível de segurança [25].

Impulsionados tanto pela demanda crescente na utilização de serviços de computação em nuvens, quanto pelo grande número de questões relacionadas à segurança nestes ambientes, diversas instituições trabalham na tipificação de serviços e especificação de padrões de interoperabilidade e de segurança em nuvens computacionais. Embora tais ações sejam um passo significativo para a criação de ambientes seguros, a maioria destes esforços ainda se encontra em estágio preliminar e necessitará de tempo considerável para amadurecimento e adoção, visto que o uso de tais padrões está fortemente associado ao interesse de provedores e da pressão dos consumidores dos serviços.

1.1 Descrição do Problema

Diante do cenário atual, percebe-se que é grande a demanda não apenas por padrões, mas também por mecanismos de especificação, monitoramento e gestão da segurança em nuvens computacionais.

A especificação de controles de segurança, voltados a atender as necessidades do usuário do serviço, pode ser alcançada através do uso de acordos de níveis de serviço de segurança (*Security-SLA*), e são apontados como uma importante ferramenta para a gestão da segurança em nuvens computacionais [22, 49, 66]. Apesar disto, observa-se na literatura a falta de trabalhos que tratam de forma concreta a especificação e o monitoramento destes acordos. Quando tratados, as soluções pouco descrevem sobre a representação de tais acordos e adotam sistemas de monitoramento desenvolvidos para arquiteturas computacionais tradicionais.

Analisando tais soluções, pode-se observar que estas não estão totalmente preparadas para o monitoramento de ambientes de nuvens, visto que:

- As ferramentas existentes possuem pouco ou nenhum suporte ao acompanhamento de SLAs, a padrões de representação de acordos e gerenciamento dos níveis especificados;
- As informações utilizadas para acompanhamento de acordos dependem de mecanismos de coleta que são executados na própria máquina em monitoramento. Em serviços de nuvens de infraestrutura, as máquinas virtuais são controladas pelo usuário, o que significa que a instalação de tais mecanismos, além de depender da colaboração do usuário, ainda está sujeita a incompatibilidades causadas por diferenças em sistemas operacionais, bibliotecas etc. Além disso, são sensíveis a adulteração em casos onde o usuário é mal intencionado;

- As ferramentas existentes não consideram eventos que ocorrem em ambientes de nuvem, como criação, suspensão, migração e término de execução da máquina virtual.

1.2 Objetivos e Contribuições

Visando a garantia da segurança através de mecanismos para acompanhamento de acordos de segurança, esta dissertação tem como objetivo principal apresentar uma solução para o monitoramento de máquinas virtuais em serviços de nuvem de infraestrutura. Nesta solução, além do foco no monitoramento dos parâmetros de segurança do *Security-SLA*, também foram tratadas questões relacionadas a: representação dos acordos; garantia da integridade dos dados a partir do uso de técnicas que eliminam a necessidade de instalação de ferramentas na máquina virtual; e integração da arquitetura à solução de gerenciamento da infraestrutura da nuvem, permitindo melhor controle das fases do ciclo de vida da máquina virtual.

Conjuntamente ao sistema de monitoramento, foram desenvolvidos mecanismos para detecção de anomalias de segurança, baseados em técnicas de aprendizado de máquina, e que utilizam dados de desempenho coletados durante a execução das máquinas virtuais em monitoramento.

De forma resumida, as contribuições presentes neste trabalho são:

- Uma linguagem para representação de acordos de níveis de serviço capaz de expressar políticas de segurança;
- Uma solução de monitoramento distribuída para o acompanhamento de acordos de níveis de serviço de segurança (*Security-SLA*) em serviços de infraestrutura;
- Um estudo sobre utilização de dados de monitoramento de desempenho coletados a partir da execução de máquinas virtuais para a detecção de anomalias de segurança;
- Uma arquitetura baseada em informações de desempenho e aprendizado de máquina para detecção de anomalias de segurança.

1.3 Organização da Dissertação

O restante desta dissertação está organizada da seguinte forma:

- **Capítulo 2:** apresenta os conceitos básicos sobre nuvens computacionais, suas características essencias e modelos de serviço, bem como as principais questões relacionadas a segurança, riscos e padrões de segurança existentes neste ambiente.

- **Capítulo 3**: discute o uso de acordos de níveis de serviço na especificação das garantias de qualidade de serviços tercerizados como telecomunicações, tecnologia da informação e segurança;
- **Capítulo 4**: apresenta os principais trabalhos relacionados ao tema desta dissertação.
- **Capítulo 5**: apresenta a arquitetura de monitoramento proposta, seus componentes e interação do sistema com as máquinas virtuais em execução no ambiente de nuvens.
- **Capítulo 6**: apresenta um estudo sobre a detecção de anomalias de segurança a partir de informações de desempenho coletadas através da solução de monitoramento proposta. Baseado nos resultados obtidos neste estudo, o capítulo propõe uma arquitetura voltada à detecção destes eventos.
- **Capítulo 7**: apresenta as conclusões e considerações finais, bem como os trabalhos futuros que podem ser derivados deste.

Capítulo 2

Computação em Nuvens

A computação em nuvens é um modelo tecnológico onde os recursos computacionais são oferecidos aos usuários na forma de serviço, garantindo grande capacidade de processamento e flexibilidade. Apesar do grande interesse nesta tecnologia, ainda é grande a desinformação sobre suas características, potenciais e riscos. Neste capítulo são apresentados os conceitos básicos e modelos de serviços da computação em nuvem. Por fim, são discutidas questões relacionadas à segurança neste ambiente.

2.1 Definição e Características Essenciais

A computação em nuvens está associada a um novo paradigma para aprovisionamento de infraestrutura computacional [91], onde capacidade de processamento e aplicações são oferecidas como serviços. Apesar de ser considerada uma nova tecnologia, suas origens remontam a década de 1960 quando o cientista da computação John McCarthy previu que a computação seria organizada como um serviço utilitário público [35]. Desde então, a intensificação do uso da Internet e o surgimento de novas tecnologias, como as citadas na Tabela 2.1, contribuíram para o surgimento da computação em nuvens.

Nuvens computacionais podem oferecer uma vasta gama de serviços, entre eles: a virtualização de recursos computacionais, o desenvolvimento de *software* e a locação de aplicações de diversos tipos. Esta variedade contribui para a existência de diferentes definições do que é a computação em nuvens, cada uma com enfoque em características específicas.

A Tabela 2.2 apresenta exemplos de definições para o termo "computação em nuvens" encontrados na literatura. Pode-se observar que inicialmente as características descritas focavam isoladamente aspectos como a computação como serviço e a computação distribuída. Com o passar dos anos, as definições tornaram-se mais completas, evoluindo para a definição proposta pelo NIST [61], que descreve aspectos como o compartilhamento de

Tabela 2.1: Tecnologias que contribuíram para o surgimento da computação em nuvens

Computação utilitária	Distribuição de serviços computacionais para usuários que apenas pagam pelos recursos que foram utilizados. De forma semelhante ao que ocorre com serviços como água, luz ou gás.
Grade computacional	A utilização de capacidade de processamento de múltiplos recursos computacionais em rede para a resolução de um problema específico.
Computação autônoma	Funcionamento de um sistema computacional sem um controle externo.
Virtualização	Particionamento lógico de recursos computacionais físicos em múltiplos ambientes de execução.
Arquiteturas orientadas a serviço	Serviços que se comunicam através de interfaces bem definidas.

recursos virtualizados e auto serviço.

A partir da definição proposta pelo NIST, foram especificadas as características essenciais que devem estar presentes em qualquer serviço de nuvem, conforme descritas a seguir:

1. **Auto serviço por demanda**: permite que os usuários utilizem os recursos computacionais da nuvem sem a necessidade de interação humana entre o usuário e o provedor da nuvem;

2. **Acesso via rede**: para que a computação em nuvens seja uma alternativa às soluções internas de *datacenters*, é necessário que a nuvem ofereça seus serviços através de uma conexão de rede com grande capacidade de banda, capaz de atender as necessidades dos usuários;

3. **Conjunto de recursos independentes do local**: a nuvem deve possuir um vasto conjunto de recursos para atender a demanda dos consumidores. Estes recursos podem estar fisicamente localizados em diferentes regiões geográficas;

4. **Elasticidade**: é a habilidade que permite a nuvem expandir ou reduzir os recursos alocados de forma rápida e eficiente de modo a atender o requisito de auto serviço da nuvem;

5. **Serviço medido**: devido a sua característica de orientação a serviço, a quantidade de recursos utilizados por um usuário pode ser automaticamente alocada e monitorada de forma que o usuário possa ser cobrado apenas pelos recursos utilizados.

Tabela 2.2: Definições para computação em nuvens

Referência	**Ano**	**Definição**
Foster et al. [31]	2008	Um paradigma de computação distribuída de larga escala, no qual um conjunto abstrato de recursos é oferecido sob demanda via Internet.
Vaquero et al. [91]	2008	Um grande conjunto de recursos virtualizados, facilmente utilizáveis e acessíveis, que podem ser reconfigurados dinamicamente para se ajustar a uma carga variável (escala). Este conjunto de recursos é normalmente explorado por um modelo de pagamento por uso, onde garantias sobre o serviço são oferecidos pelo provedor por meio de acordos de níveis de serviço (SLA) personalizados.
Buyya et al. [12]	2009	Um sistema de computação paralela e distribuída composta por um conjunto de computadores interligados e virtualizados, que são dinamicamente aprovisionados e apresentados como um ou mais recursos de computação unificados com base em SLAs estabelecidos entre o provedor do serviços e os usuários.
ENISA[2] [25]	2009	Um modelo sob demanda para aprovisionamento de recursos de tecnologia da informação (TI), geralmente baseado em virtualização e sistemas distribuídos.
Furht and Escalante [33]	2010	Um novo estilo de computação em que os recursos dinamicamente escaláveis e muitas vezes virtualizados são fornecidos como um serviço através da Internet.
NIST[3] [61]	2011	Um modelo que permite acesso em rede, por demanda, a um conjunto compartilhado e confiável de recursos computacionais que podem ser rapidamente aprovisionados e liberados com o mínimo de esforço de gerenciamento por parte do usuário ou interação pelo provedor do serviço.

[2]European Network and Information Security Agency
[3]U.S. National Institute of Standards and Technology

2.2 Tipos de Nuvens Computacionais

Existem diferentes modos de se classificar um serviço de computação em nuvens. As principais baseiam-se no tipo de serviço oferecido e na forma como a nuvem é implantada. Nesta seção, serão apresentas as principais características destes dois modelos.

2.2.1 Modelos de Serviço

De acordo com o modelo de disponibilização de serviço, as nuvens computacionais podem ser classificadas da seguinte forma:

- **Software como serviço** (*Software as a Service* - SaaS): serviços SaaS disponibilizam aplicativos através da *web* em um modelo onde o usuário aluga aplicações que são executadas através da Internet, a partir de um navegador *web*, sem a necessidade de instalação de *software* localmente. As aplicações em nuvens SaaS são multi-locadas, ou seja, são utilizadas por diversos clientes simultaneamente.

- **Plataforma como serviço** (*Platform as a Service* - PaaS): através desta categoria de serviço, desenvolvedores de *software* podem criar aplicações para *web* e instalá-las na nuvem, sem a necessidade de instalar ferramentas de desenvolvimento locais.

  Em serviços PaaS, o provedor oferece um ambiente de desenvolvimento e execução de aplicações para nuvens. Tais ambientes são compostos por linguagens de programação, interfaces de programação de aplicações (*Application Programming Interfaces* - API), ferramentas de gerenciamento de configuração, implantação e execução de aplicações.

- **Infraestrutura como serviço** (*Infrastructure as a Service* - IaaS): o serviço IaaS fornece acesso a recursos de infraestrutura computacional básica (capacidade de processamento, armazenamento e rede), garantindo um nível de escalabilidade que pode responder rapidamente à demanda de recursos dos usuários.

  Serviços de infraestrutura de computação permitem aos usuários adquirirem recursos computacionais sob a forma de máquinas virtuais (*virtual machines* - VM). Similarmente, serviços de armazenamento fornecem capacidade de armazenamento de baixo nível (blocos) ou alto nível (arquivos), onde se paga pelo espaço consumido e pelo tráfego gerado para leitura e/ou gravação destes dados na nuvem.

Interdependência dos Modelos de Serviço

Uma importante característica da arquitetura de serviços da computação em nuvens é a interdependência entre os diferentes modelos de serviço, como mostrado na Figura 2.1.

Pode-se observar que os serviços de uma nuvem SaaS dependem diretamente dos recursos fornecidos pela camada PaaS. Da mesma forma que a camada PaaS apoia-se nos recursos oferecidos pela camada IaaS,

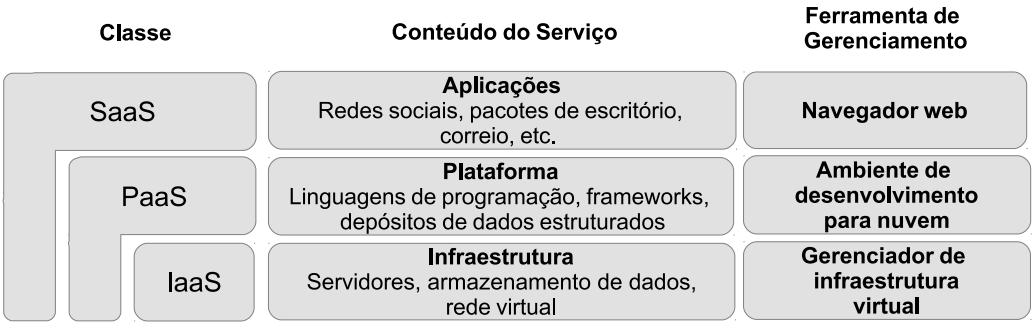


Figura 2.1: Pilha de serviços de nuvens

2.2.2 Modelos de implantação

No modelo de implantação, as nuvens são classificadas de acordo com o seu modo de gestão e da relação entre os usuários do serviço. Neste modelo, as nuvens podem ser divididas da seguinte forma:

- **Nuvens públicas**: a infraestrutura da nuvem pertence a uma organização que vende os serviços entre diferentes clientes. Todos os clientes compartilham o conjunto de recursos providos pela infraestrutura;
- **Nuvens privadas**: a infraestrutura da nuvem é operada somente para uma organização, podendo ser gerenciada pela própria organização ou por um terceiro;
- **Nuvens comunitárias**: a infraestrutura da nuvem é compartilhada por um grupo de organizações com objetivos comuns;
- **Nuvens híbridas**: a infraestrutura da nuvem é composta por duas ou mais nuvens que apesar de manterem suas identidades, estão interligadas através de tecnologias padronizadas ou proprietárias, garantindo assim a portabilidade entre dados e aplicações.

2.3 Portabilidade de Serviços em Nuvens

A portabilidade dos serviços de nuvem entre provedores é uma importante questão que deve ser considerada pelos usuários do serviço. Os provedores comerciais geralmente

utilizam soluções proprietárias e não padronizadas em suas infraestruturas, diminuindo a possibilidade de migração do serviço e fazendo com que o usuário fique "preso" (*vendor lock-in*) [25, 2]. Esta dependência pode ocorrer de diferentes formas dependendo do modelo de disponibilização do serviço. Em serviços IaaS, um usuário pode ficar dependente da tecnologia de virtualização ou das APIs utilizadas para acesso aos dados armazenados na nuvem. Em serviços PaaS, as aplicações desenvolvidas são dependentes do *framework* de desenvolvimento oferecido pelo provedor.

Atualmente os esforços para minimizar os efeitos da dependência do provedor baseiam-se em três padrões que definem mecanismos para gerenciamento de recursos computacionais em nuvens, acesso e manipulação de dados, e portabilidade de máquinas virtuais entre serviços.

Open Cloud Computing Interface (OCCI) [70] é um padrão de interoperabilidade proposto pelo OGF[4] voltado ao gerenciamento remoto de recursos em nuvens. O OCCI define um conjunto de APIs e um protocolo que atuam como um serviço de *front-end* para a arquitetura de gerenciamento da infraestrutura de nuvem, permitindo que aplicações sejam capazes de gerenciar e monitorar recursos de nuvens independente de conhecimento da estrutura interna ou APIs proprietárias da solução utilizada.

De modo similar, a especificação *Cloud Data Management Interface* (CDMI) [85] fornece uma interface padrão que permite que aplicações manipulem e gerenciem diferentes tipos de dados (arquivos, blocos, tabelas de bancos de dados etc.) armazenados em nuvens computacionais. Este padrão define um protocolo baseado em HTTP que fornece funcionalidades para descoberta de capacidades do serviço, gerenciamento de dados, associação de metadados aos objetos, gerenciamento de grupos e usuários, controle de acesso, e cobrança (*billing*). A especificação CDMI foi adotada como padrão internacional ISO/IEC 17826:2012

Open Virtualization Format (OVF) [23] é um padrão para intercâmbio de máquinas virtuais. Embora este não seja um padrão específico para nuvens computacionais, permite que máquinas virtuais criadas em uma solução ou provedor, sejam exportados em um formato comum e posteriormente implantados em outra solução. A especificação OVF foi adotada como padrão pela ANSI INCITS 469-2010 e ISO/IEC 17203:2011.

É importante ressaltar que a adoção destes padrões não é uniforme. Tais padrões são comumente suportadas por soluções de nuvens de código aberto e não em provedores comerciais.

[4]Open Grid Forum

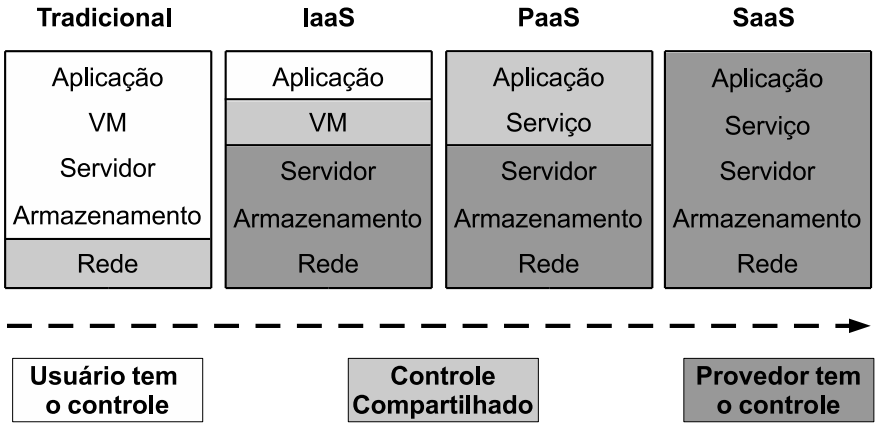


Figura 2.2: Divisão da governança nos modelos de serviços

2.4 Segurança em Nuvens

Embora a segurança e privacidade sejam questões de extrema relevância para os usuários de serviços de nuvens, as pesquisas nesta área ainda encontram-se em estágios iniciais [13]. Como resultado, a falta de padrões e mecanismos reguladores da segurança ainda causam um grande impacto no crescimento do uso de serviços em nuvens [88].

Paralelamente às questões de padronização, a migração de serviços de arquiteturas tradicionais para nuvens resulta na transferência da responsabilidade da segurança para o provedor do serviço. Isto é conhecido como **perda de governança** [59].

A perda de governança não ocorre apenas com os dados, mas com todo componente da arquitetura computacional. A Figura 2.2 [59] mostra os limites de governança existentes na arquitetura tradicional e nos diferentes modelos de serviço da computação em nuvem.

Através destes limites, pode-se compreender outro aspecto importante da segurança na computação em nuvens: o **compartilhamento da responsabilidade**. Em arquiteturas tradicionais o usuário tem controle sobre toda a infraestrutura e por isto, detém toda a responsabilidade pela segurança. Já em um ambiente de nuvem, a responsabilidade do usuário diminui de acordo com o modelo de serviço utilizado. Usuários de serviços IaaS são responsáveis pela segurança de VM e aplicações; em serviços SaaS, toda responsabilidade recai sobre o provedor do serviço.

Do ponto de vista do provedor, o compartilhamento da responsabilidade também representa um desafio, visto que em serviços IaaS o usuário tem controle sobre o sistema operacional e aplicativos em execução na VM. Isto faz com que VMs se tornem componentes vulneráveis da arquitetura da nuvem, que podem ser utilizado para comprometer outras VMs ou a própria máquina física (nó) onde a VM é executada.

2.4.1 Riscos

A utilização de qualquer modelo tecnológico está associada a riscos inerentes a este modelo [68]. Na computação em nuvens, as diferentes tecnologias que se integram para compor a arquitetura da nuvem (*hardware*, sistemas operacionais, tecnologias de virtualização etc.) introduzem riscos bem conhecidos. Ao mesmo tempo, a combinação destas tecnologias cria riscos até então não existentes. Por exemplo, aplicações e VMs em nuvem estão sujeitas a ataques como qualquer outro serviço na Internet. Contudo, a combinação da virtualização e do compartilhamento de recursos faz com que a infraestrutura da nuvem seja vulnerável a ataques iniciados por outras VMs em execução na própria nuvem.

Na literatura, diversos trabalhos abordam os riscos de segurança da computação em nuvens. Tais riscos vão além de questões técnicas, e podem ser divididos em três categorias [25]:

1. **Organizacionais e de políticas**: são riscos relacionados a perda de governança, quebra de observância a políticas organizacionais e padrões reguladores, dependência de provedores de serviço etc.
2. **Técnicos**: relacionados a questões sobre a segurança das tecnologias empregadas pelo provedor, comunicação de rede, gerenciamento do serviço, etc.
3. **Legais**: dizem respeito a quebra da proteção de dados devido a diferenças na legislação entre países que hospedam os servidores da nuvem.

Baseado nesta classificação, a Tabela 2.3 apresenta uma compilação dos principais riscos encontrados na computação em nuvens. É importante ressaltar que o impacto causado por estes riscos e a responsabilidade pelo seu controle varia de acordo com o tipo de serviço. Por exemplo, para minimizar os riscos de um ataque de injeção de SQL em uma aplicação, é de responsabilidade do provedor de PaaS oferecer mecanismos de controle em seu *framework* de desenvolvimento. Ao mesmo tempo, é responsabilidade do usuário, desenvolver suas aplicações seguindo padrões e recomendações de segurança que evitem este tipo de ataque.

2.4.2 Padrões de Segurança em Nuvens

A definição de padrões de segurança para a computação em nuvem ainda encontra-se em seu estágio inicial. Documentos como [54] e [19], fornecem recomendações e modelos de mapeamento dos padrões de segurança da informação desenvolvidos para arquiteturas tradicionais, para um contexto de serviço de nuvens.

Paralelamente, diversas instituições trabalham para a definição de padrões de segurança específicos para a computação em nuvens. Apesar destes esforços, ainda não existe nenhum padrão de segurança e privacidade que regule os serviços de nuvens [77].

Tabela 2.3: Principais riscos na computação em nuvens

Risco	Tipo de Risco	Descrição
Interoperabilidade [25, 59, 88]	Organizacional	A falta de padrões de interoperabilidade pode fazer com que um usuário torne-se dependente do serviço de um provedor (*vendor lock-in*).
Governança [25, 59]	Organizacional	Ao migrar dados para um serviço de nuvem, o usuário passa a ser dependente dos controles de segurança oferecidos pelo provedor para proteger seus dados.
Padrões e auditoria [25, 59, 54]	Organizacional	A utilização de serviços de nuvens pode impactar na observância de padrões e dificultar processos de auditorias e de certificações.
Disponibilidade [25, 54, 88]	Organizacional	A interrupção de serviços seja ela causada por falha técnica ou desastres naturais pode resultar em prejuízos causados por perda de dados ou interrupção do negócio.
Intimação [25, 59, 54]	Legal	Intimações legais que levem a apreensão de equipamentos de um provedor podem levar a liberação de dados sensíveis de usuários do serviço.
Jurisdições [25, 59, 54]	Legal	As leis de proteção a informações variam de acordo com os países. É importante conhecer a legislação do país onde o provedor mantêm seu *datacenter*.
Interceptação de dados [25, 59, 54, 88]	Técnico	Dados podem ser capturados indevidamente durante a comunicação entre nós da nuvem ou entre o usuário e a nuvem.
Autenticação e autorização [25, 59, 54, 88]	Técnico	Problemas de autenticação e autorização podem permitir que terceiros comprometam dados e serviços.
Backup e exclusão de dados [25, 59, 54, 88]	Técnico	A falta de proteção de cópias de segurança, bem como a exclusão não segura de dados e máquinas virtuais podem levar a acessos indevidos por parte de terceiros.
Gerenciamento de chaves [25, 54, 88]	Técnico	A perda de chaves criptográficas pode permitir a autorização indevida de usuários garantindo acesso a dados sensíveis.
Isolamento [25, 59, 54, 88]	Técnico	Falhas de virtualização e de armazenamento podem resultar em acessos não autorizados a dados ou utilização indevida de recursos.
Uso indevido de privilégios [25, 59, 54]	Técnico	Funcionários do provedor com alto nível de privilégios podem utilizar estes privilégios para realizar atividades maliciosas (*malicious insider*).
Sistema de gerenciamento [25, 59, 88]	Técnico	O acesso indevido ou comprometimento de ferramentas de gerenciamento do serviço pode levar a: perda de controle de recursos, interrupção de serviços e perda de dados.
Ataques e probes [25, 59, 54]	Técnico	Assim como em arquiteturas tradicionais, tanto as aplicações quanto a infraestrutura da nuvem estão sujeitos a ataques como: negação de serviço (*Denial of Service* - DoS), SQL Injection, scans, etc.
Escalamento de privilégios [25, 59, 54]	Técnico	Serviços comprometidos em uma VM podem ser usados para atacar outras VMs ou o próprio host.

A seguir são apresentados os principais esforços desenvolvidos para padronização da segurança na computação em nuvens.

CSA Security Guidance

Em *"Security Guidance for Critical Areas of Focus in Cloud Computing"*[2], é apresentado um conjunto de recomendações de procedimentos e aspectos a serem observados por empresas que tenham interesse em adotar e operar serviços em nuvens. O documento é organizado em domínios que abordam a segurança em dois níveis distintos:

- **Governança em nuvens**: são tratados os domínios relacionadas ao gerenciamento de riscos, questões legais, observância a padrões de segurança existente, auditoria, gerenciamento de informações e segurança de dados;
- **Operação em nuvens**: trata do uso de modelos de segurança tradicionais, recuperação de desastres, resposta a incidentes, criptografia, gerenciamento de identidade, controle de acesso, virtualização e segurança como serviço.

NIST Cloud Computing Security Reference Architecture

"*NIST Cloud Computing Security Reference Architeture*" [69] é um guia que orienta as agências do governo dos Estados Unidos no processos de migração de serviços para o ambiente de nuven, mantendo os mesmos níveis de segurança e observância a padrões definidos pelo governo americano para arquiteturas de computação tradicionais. O documento utiliza uma metodologia baseada no modelo de gerenciamento de riscos do NIST [67], que permite que as agências sejam capazes de: descrever o serviço a ser migrado; tipo de serviço de nuvem mais adequado; características de segurança necessárias no serviço de nuvem; e atores e riscos envolvidos.

ISO/IEC-27017 e ISO/IEC-27018

Dois padrões internacionais de segurança para nuvem encontram-se em desenvolvimento pela ISO e estenderão a série ISO/IEC-27000 [47]. O padrão ISO/IEC-27017 definirá um código de práticas de controle de segurança da informação em nuvens baseado nos controles já existentes no padrão ISO 27002 [48] e em novos controles, específicos para nuvens que serão voltados tanto para uso de provedores de serviço quanto para usuários. O padrão ISO/IEC-27018 tem como finalidade, prover orientações sobre aspectos e elementos de privacidade para o uso de serviços em nuvens públicas.

Ambos os padrões encontram-se em desenvolvimento (*draft*) e contam com as contribuições de instituições de diversos países e também da CSA. É esperado que o padrão ISO/IEC-27018 seja publicado em 2013 e o ISO/IEC-27017 em 2014.

2.5 Conclusão

Neste capítulo foram apresentados os principais conceitos relacionados à computação em nuvens. Foram descritas as tecnologias que compõem a arquitetura de nuvens, suas características essenciais, modelo de serviço e modelo de implantação. Em seguida foram discutidos os aspectos relacionados à segurança: a mudança de paradigma causado pela perda do controle de dados e compartilhamento da responsabilidade sobre a segurança. Por fim, foram apresentados os principais riscos existentes na computação em nuvem, os padrões e recomendações de segurança existentes.

Capítulo 3

Acordos de Níveis de Serviço

A especificação de parâmetros de garantia que assegurem a qualidade de serviços é um mecanismo essencial em ambientes onde a terceirização (*outsourcing*) é utilizada. Este capítulo discute a importância de acordos de níveis de serviço como forma de especificar tais garantias, seu uso em serviços de tecnologia da informação e segurança e as formas para representação de tais acordos.

3.1 Definição

Os acordos de níveis de serviço (*Service Level Agreement* - SLA) são parte do contrato de prestação de serviço firmados entre o provedor e o usuário e descrevem as qualidades de serviço (QoS) desejadas [9]. Um SLA por si só não garante que as qualidades especificadas sejam cumpridas, mas fornece os mecanismos necessários para o monitoramento, aponta as responsabilidades e define as punições e compensações caso o acordo não seja cumprido.

O SLA era utilizado originalmente em serviços de telecomunicação para especificar características técnicas e garantias oferecidas ao usuário como: largura de banda, disponibilidade, taxas de erros etc. Neste contexto, Muller [64] afirma que tais acordos deveriam possuir um conjunto mínimo de informações, conforme descrito na Tabela 3.1.

Em serviços de tecnologia de informação (TI), a utilização de SLAs adota uma abordagem diferente dos serviços de telecomunicação. O acordo passa a representar tanto as expectativas do usuário quanto as do provedor. Com isto, as obrigações podem ser especificadas para qualquer uma das partes envolvidas. As informações contidas no acordo também são diferenciadas. Segundo Bianco et al. [11], um SLA de serviços de TI deve tratar as seguintes informações:

- A descrição dos serviços;
- As partes envolvidas;

Tabela 3.1: Informações que compõem um SLA de telecomunicação

Contexto	Apresenta um conjunto de informações que permitem que um leitor, mesmo que não técnico, compreenda o serviço e suas garantias.
Partes	Identifica as partes envolvidas no acordo.
Serviço	Especifica o volume do serviço que será fornecido.
Pontualidade	Fornece uma medida qualitativa dos prazos para execução de solicitações do usuário.
Disponibilidade	Descreve os períodos onde o serviço estará disponível.
Limitações	Descreve os limites do serviço fornecido, quando este se encontrar em condições de pico de uso, causado por grandes demandas de utilização.
Compensações	Apresenta os mecanismos utilizados para a compensação do usuário em situações onde uma violação dos níveis de serviço contratados ocorrer.
Medições	Descreve o processo de monitoramento dos níveis de serviço.
Renegociação	Define as situações nas quais o SLA deverá ser modificado.

- As obrigações e/ou níveis de serviço desejados;
- As métricas utilizadas para o monitoramento do serviço;
- Os responsáveis pelo monitoramento;
- Ações e penalidades que serão tomadas quando as obrigações especificadas não forem atendidas;
- Mecanismos de evolução do SLA.

É importante destacar que no contexto de um SLA, o acompanhamento dos níveis de serviço é tão importante quanto a especificação destes níveis. Para isso, são utilizadas métricas que permitem avaliar o cumprimento das qualidades de serviço desejadas. A forma como tais métricas são medidas depende do tipo de serviço e do tipo de característica de qualidade que se deseja aferir.

De modo geral, as qualidades especificadas em um SLA podem ser classificadas em mensuráveis e imensuráveis. As qualidades mensuráveis são aquelas que podem ser medidas de forma automática através de métricas. Já as qualidades imensuráveis não permitem uma medição automática, pois não podem ser avaliadas através de um método que resulte em um valor único. Nestes casos são utilizados conjuntos de métricas secundárias que aferem aspectos específicos da qualidade imensurável. A Tabela 3.2 [11] apresenta uma relação de qualidades mensuráveis e imensuráveis encontradas em serviços de TI.

Tabela 3.2: Exemplos de qualidades em serviço de TI

Qualidades mensuráveis	
Precisão	Limite da taxa de erro para o serviço durante um determinado período de tempo.
Disponibilidade	Probabilidade do serviço estar disponível quando necessário.
Capacidade	Número de solicitações concorrentes suportadas pelo serviço.
Custo	Custo do serviço.
Latência	O tempo máximo entre a chegada da solicitação e o tempo para finalizar a solicitação.
Tempo de provisionamento	Tempo necessário para que o serviço se torne operacional.
Confiabilidade das mensagens	Garantia de entrega das mensagens.
Escalabilidade	Habilidade do serviço de aumentar o número de operações executadas com sucesso em um período de tempo.
Qualidades imensuráveis	
Interoperabilidade	Habilidade de intercomunicação com outros serviços.
Modificabilidade	Frequência de modificações (interface ou implementação) que podem ocorrer em um serviço.
Segurança	Habilidade do serviço de resistir a uso não autorizado enquanto provê serviço a clientes legítimos.

3.2 Ciclo de vida do SLA

O SLA não é um documento estático e a sua correta utilização é resultado da execução de diversas atividades conduzidas nos diferentes estágios de seu vida. Segundo [89], o ciclo de vida de um SLA é composto por seis fases:

1. **Definição**: Esta fase é voltada à identificação do serviço e suas características, bem como a definição dos parâmetros de qualidade que serão fornecidos aos usuários;
2. **Negociação**: Nesta fase são definidos os valores para os parâmetros do serviço, o custo para o usuário e as penalidades caso o SLA seja violado;
3. **Implementação**: O serviço é preparado para consumo pelo usuário;
4. **Execução**: É a fase de operação e monitoramento do serviço. Nesta fase os parâmetros de qualidade especificados são avaliados para verificação do cumprimento do SLA;
5. **Avaliação**: Nesta fase o provedor avalia a qualidade do serviço fornecido;

6. **Término**: Trata as questões de finalização do serviço, seja por questões de expiração do contrato ou violação do SLA.

3.3 Security-SLA

A crescente utilização de serviços terceirizados de TI ocasiona uma preocupação cada vez maior com questões que envolvem a segurança e privacidade [49]. Com isso, é natural que tais questões sejam abordadas em SLAs, que permitem que o usuário especifique os níveis de segurança que devem ser garantidos pelo serviço. Contudo, a especificação de SLAs que envolvem características de segurança (*Security-SLA*) apresenta desafios que envolvem a especificação dos níveis de segurança, a representação destes acordos e finalmente o seu monitoramento.

3.3.1 Especificação de Parâmetros de Segurança para SLA

Na literatura, a definição de parâmetros de segurança pode ser feita de duas formas: (i) através de políticas de segurança ou (ii) a partir de métricas de segurança.

A especificação de parâmetros de segurança através de políticas, como proposto em [15], considera o *Security-SLA* como um conjunto de políticas expressas em linguagem padrão (ex. *WS-Policy* [96]). Embora esta abordagem seja capaz de especificar claramente os níveis de segurança desejados, a utilização de políticas falha na especificação dos mecanismos de monitoramento e ignora a representação de diversas informações integrantes de um SLA.

Já a especificação a partir de métricas de segurança é um método comumente empregado e permite especificar não apenas os parâmetros de segurança, mas também forma de monitoramento. Ao contrário da especificação por políticas, a especificação por métricas baseia-se em um conjunto de métricas de segurança que medem se um determinado objetivo (controle) está sendo cumprido.

3.3.2 Métricas de Segurança

Embora a segurança seja uma qualidade imensurável [11, 53], é comum a utilização de métricas de segurança para a avaliação do estado de segurança em um ambiente.

Esta aparente discrepância ocorre devido ao incorreto entendimento de que um único valor seja capaz de representar o nível de segurança de um sistema. A segurança como qualidade imensurável parte da premissa de que não existe um modelo não ambíguo e amplamente aceito que permita dizer que um sistema é mais seguro que outro [53].

Por outro lado, as métricas de segurança são ferramentas que fornecem informações corretas e atualizadas sobre o estado de segurança de um ambiente, permitindo avaliar operações e controles de segurança no ambiente onde são aferidos [81].

Segundo [17], as métricas de segurança podem ser classificadas da seguinte forma:

- **Implementação**: São voltadas a mostrar a evolução da implementação de programas, controles, procedimentos e políticas de segurança;
- **Efetividade e eficiência**: Monitoram a correta implementação de controles e processos de segurança;
- **Impacto**: Medem o efeito da segurança da informação na missão da organização.

Paralelamente, as métricas de segurança também são classificadas de acordo com a sua audiência [30], e podem ser divididas em três categorias: (i) **gestão**, fornece informações sobre o desempenho das funções de negócio e o impacto na organização; (ii) **operacionais**, usadas para compreender e otimizar as atividades das funções de negócio; e (iii) **técnicas**, fornecem detalhes técnicos e formam a base para as outras categorias de métricas. A Tabela 3.3 [30] apresenta exemplos de métricas de segurança relacionadas ao gerenciamento de incidentes, classificadas de acordo com as categorias apresentadas.

Tabela 3.3: Exemplos de métricas de segurança e suas categorias

Categoria	Métrica
Gestão	• Custo dos incidentes • Custo médio dos incidentes • Porcentagem de sistemas sem vulnerabilidades severas • Observância à política de atualização • Porcentagem de observância de configuração • Porcentagem de gastos com segurança do orçamento de TI
Operacional	• Tempo médio para descoberta de incidentes • Tempo médio entre incidentes • Tempo médio para recuperação de incidentes • Tempo médio para atualização • Tempo médio para mitigação de vulnerabilidades
Técnica	• Número de incidentes • Número de vulnerabilidades conhecidas • Cobertura da verificação de vulnerabilidades • Observância do gerenciamento de configuração • Observância do gerenciamento de atualização

Métricas utilizadas em *Security-SLA* são métricas de efetividade/eficiência e podem pertencer a qualquer uma das categorias de audiência. Apesar disto, nem todas as métricas encontradas em uma categoria podem ser utilizadas em *Security-SLA*. Por exemplo, na Tabela 3.3, a métrica "*Porcentagem do orçamento de TI gasto com segurança*" ou "*Número de vulnerabilidades conhecidas*" são métricas que não são negociáveis no contexto de um acordo *Security-SLA*.

Embora o uso criterioso de métricas de segurança promova transparência, apoie a tomada de decisão, a previsibilidade e o planejamento pró-ativo [41], o processo de definição destas métricas não é uma tarefa simples, visto que características de segurança não são facilmente quantificadas [42, 15].

Atualmente, padrões [47, 17], guias [30, 81] e livros [83, 41] sobre mensuramento da segurança oferecem um conjunto rico de métricas que podem ser utilizadas em diversos cenários. Apesar disto, tais documentos não são capazes de fornecer uma cobertura completa que atenda totalmente às necessidades dos usuários. Nestes casos, torna-se necessário empregar metodologias voltadas à especificação de tais métricas. A seguir, são apresentadas as principais metodologias utilizadas na especificação de métricas para *Security-SLA*.

Especificação de Métricas Baseada em Políticas de Segurança

A especificação de métricas para uso em acordos *Security-SLA* a partir de políticas de segurança baseia-se em uma metodologia proposta por Henning [42]. Esta metodologia utiliza políticas, normas e padrões de segurança como ponto de partida para a derivação das métricas. O processo de derivação é composto por três etapas: (i) análise de políticas, voltada a especificar o conjunto básico de métricas que serão trabalhadas; (ii) análise da arquitetura, que tem o objetivo de validar os requisitos encontrados na etapa anterior quanto a sua aplicação na arquitetura computacional existente; (iii) entrevista com os usuários do serviço, que são utilizadas para indicar quais são as preocupações com a segurança segundo o ponto de vista do usuário.

Trabalhos mais recentes tratam estendem esta metodologia e tratam aspectos não abordados originalmente. Em [82] é proposto um método de validação de métricas e valores de parâmetros baseado em análises de medições feitas na infraestrutura do serviço. Já Chaves et al. [22] discutem a utilização desta metodologia em nuvens computacionais com foco nas particularidades desta arquitetura e também propõem uma nova etapa no processo, voltada à negociação do acordo entre o usuário e o provedor do serviço.

Especificação de Métricas Baseada na Metodologia GQM

A metodologia GQM (*Goal, Question, Metric*) [7] é um processo proposto para medição empírica em testes de *software*, baseada em metas e objetivos bem definidos. A metodologia emprega um modelo de medição composto por três níveis: (i) conceitual (*goal*), que representa o alvo da medição; (ii) operacional (*question*), no qual o alvo é refinado em um conjunto de questões operacionais; (iii) métrica (*metric*), que é um conjunto de métricas que respondem as questões especificadas no nível superior de forma quantitativa.

Em segurança, esta metodologia é aplicada em diversos trabalhos. Em [81] o modelo GQM é utilizado juntamente com o *framework* COBIT [46] para a especificação de métricas para *Security-SLA* em nuvens computacionais. Em [20], o modelo GQM é empregado na construção de uma hierarquia de métricas utilizada para a geração de um índice geral da segurança em nuvens computacionais. A metodologia GQM também faz parte de guias para a definição de métricas de segurança, como o proposto por Payne [78].

3.4 Representação de SLA

A representação da acordos SLA pode ser feita através de um vasto conjunto de formatos que vão desde a representação textual, onde os níveis de serviço são descritos em linguagem natural [11], até representações que utilizam linguagens voltadas para fins específicos. Embora o uso de linguagens naturais para a representação de acordos garanta maior flexibilidade e facilidade de entendimento destes acordos, o seu uso é inadequado quando são consideradas questões relacionados ao monitoramento automatizado, portabilidade e manutenção destes documentos.

O uso de acordos expressos através de linguagens padronizadas que permitem sua utilização direta em sistemas computacionais apresenta as seguintes vantagens [11]:

- Suportar ferramentas de negociação automática;
- Permitir o controle automático dos custos do serviço;
- Permitir a utilização de controles automatizados de acompanhamento do cumprimento do SLA;
- Permitir a especificação de mecanismos automatizados que executam ações ou disparam notificações em resposta a eventos ou violação do SLA.

Diferentes linguagens para representação de acordos SLA foram propostas, cada uma suportando diferentes tipos de serviços e contextos. A seguir, são relacionados os principais padrões de linguagens utilizados para a representação de SLAs. A relação apresen-

tada é um levantamento não exaustivo do tema e concentra-se em linguagens baseadas em XML, visando a portabilidade dos documentos produzidos.

SLAng [55] é uma linguagem orientada a definição de SLAs complexos envolvendo diferentes tipos de serviços (rede, armazenamento, aplicação etc.). A linguagem baseia-se em um modelo de aprovisionamento com três domínios (aplicação, camada intermediária e recursos subjacentes), seis tipos de partes (aplicação, *web server*, componente, *container*, armazenamento, rede), e sete tipos de acordos (aplicação, hospedagem, persistência, comunicação, serviço, *container*, rede). SLAng também adota uma abordagem onde os acordos podem ser classificados de duas formas distintas: horizontal e vertical. Em SLAs horizontais, acordos são estabelecidos entre duas partes no mesmo nível arquitetural e que provêm serviços similares. Por exemplo, um provedor de componentes contrata os serviços de outro provedor. Em SLAs verticais, são descritos acordos entre partes em diferentes níveis da pilha de arquitetura. Por exemplo, um acordo entre um provedor de componentes e um provedor de rede.

Web Service Level Agreement language (WSLA) [57] é uma especificação proposta pela IBM para a criação de acordos para *web services* que permitem descrever o serviço e as qualidades que devem ser garantidas. Um documento WSLA é composto por três seções: (i) a identificação das partes envolvidas no acordo; (ii) a definição do serviço, onde são descritas as características dos serviços; (iii) os objetivos, que descrevem as obrigações do acordo, seus responsáveis e ações a serem executadas em caso de violação. A estrutura do WSLA permite ainda a representação de métricas complexas a partir de expressões que relacionam outras métricas. A linguagem também conta com um rico dialeto composto por operadores aritméticos e lógicos, e funções de manipulação de séries temporais, utilizados para especificação de expressões lógicas para o monitoramento de níveis de serviço.

WS-Agreement [4] é uma linguagem e um protocolo proposto pelo OGF[1] para publicação de capacidades dos provedores de serviço, negociação, representação de acordos e monitoramento em tempo de execução. WS-Agreement é uma linguagem que pode ser utilizada para representar acordos em quaisquer domínios de serviço. Para isso, ela utiliza outras linguagens para representação do domínio a ser tratado, por exemplo, *WebService Definition Language* (WSDL), *Job Submission Description Language* (JSDL) etc. Um acordo expresso em WS-Agreement é composto por três seções: (i) a identificação, uma seção opcional que identifica o acordo; (ii) o contexto, seção mandatória que especifica as partes e a duração do acordo; (iii) os termos, que descreve o serviço oferecido e as suas garantias.

SecAgreement [39] é uma extensão voltada a representação de acordos de segurança utilizando a linguagem *WS-Agreement*. Esta extensão inclui novos objetos semânticos

[1] Open Grid Forum

e operações que permitem expressar políticas de segurança nos termos de descrição do serviço e nos níveis garantidos pelo serviço.

WS-Policy [96] é uma especificação que provê mecanismos necessários para permitir que aplicações baseadas em *web services* especifiquem suas capacidades, requisitos e características gerais das entidades através de declarações XML. *WS-Policy* possui uma gramática flexível e extensível que permite a especificação de diferentes tipos de políticas utilizando extensões. Por exemplo, a representação de políticas de segurança é feita através da extensão *WS-SecurityPolicy* [73]. Embora não seja linguagem para representação de SLA, este *framework* é utilizado na representação de *Security-SLAs* em [15].

3.5 Conclusão

Este capítulo abordou o tema dos acordos de níveis de serviço, descrevendo sua origem, objetivos e principais componentes. Foi discutida a importância desses acordos no crescente mercado de terceirização de serviços de tecnologia da informação, não apenas para a especificação de características de desempenho, mas também como mecanismo para controle dos níveis de segurança providos pelo serviço. Por fim, foram apresentadas as características encontradas nas principais linguagens utilizadas para a representação desses acordos.

Capítulo 4

Trabalhos Relacionados

Este capítulo apresenta os principais trabalhos relacionados ao tema desta dissertação, organizados em dois grupos principais: soluções de monitoramento e sistemas de detecção de intrusão.

4.1 Monitoramento em Nuvens

Comercialmente, vários provedores oferecem suas próprias soluções para monitoramento do serviço de nuvem. A plataforma AWS da Amazon oferece o CloudWatch [3], um sistema de monitoramento oferecido como serviço para o controle de recursos e serviços de aplicações. Microsoft Windows Azure [62] possui o *Azure Fabric Controller*, responsável por monitorar e gerenciar servidores e coordenar recursos para as aplicações. O *Google App Engine* [38] oferece um conjunto de APIs que permitem a utilização de soluções de monitoramento como o *CloudStatus* [44].

Soluções de nuvens de código aberto como *Eucalyptus* [26], *OpenNebula* [74] e *OpenStack* [75] também oferecem recursos integrados para monitoramento. Estas soluções permitem apenas o monitoramento de informações básicas como carga de CPU, uso de espaço de armazenamento e tráfego de rede.

Emeakaroha et al. [24] apresentam a solução de monitoramento LoM2HiS que faz parte da arquitetura para detecção de violações em SLA. A solução LoM2HiS é composta por um conjunto de agentes, baseados na ferramenta *Ganglia* [34], responsáveis por coletar métricas de baixo nível e enviá-las a monitores, os quais têm a função de agregá-los em métricas de alto nível, que serão utilizadas pelo módulo de detecção de violação do acordo SLA. Embora a arquitetura LoM2HiS compartilhe características com a solução proposta nesta dissertação como a arquitetura distribuída, uso de agentes para coleta de dados e o monitoramento a partir de informações de SLA; seu uso é voltado ao monitoramento de parâmetros de desempenho em serviços SaaS. A arquitetura não emprega mecanismos para

coleta segura visto que todos os componentes da infraestrutura da nuvem são controladas pelo provedor.

Alhamazani et al. [1] propõem uma arquitetura para gerenciamento e monitoramento de QoS de aplicações de transmissão de mídia. Para isso, a solução monitora parâmetros de QoS em diferentes camadas do serviço de nuvem como servidor de mídia, *web services*, processamento e armazenamento etc. A escalabilidade da arquitetura é tratada através do uso de tabelas de *hash* distribuídos (*Distributed Hash Table* - DHT) e técnicas de indexação para distribuição de dados. A solução emprega agentes para o monitoramento das diferentes camadas da arquitetura da nuvem (IaaS, PaaS e SaaS). As informações coletadas por esses agentes são então disponibilizada através do protocolo SNMP e se tornam acessíveis ao sistema de gerenciamento da solução. Embora o sistema seja voltada para o monitoramento de parâmetros de QoS, não existe nenhuma relação formal com acordos de SLA para tais parâmetros.

Em [56], é apresentado um sistema de monitoramento para nuvens que utiliza um modelo robusto de distribuição de dados. A arquitetura da solução emprega um modelo de dados com três categorias de informação: (i) representação da entidade a ser monitorada; (ii) informações de tempo de execução da entidade; (iii) eventos que ocorrem durante a execução da entidade. A solução utiliza um modelo de publicador/assinante, onde os publicadores são representados pelos agentes de monitoramento e os assinantes são aplicações que consomem os dados. Apesar da flexibilidade da solução, sua arquitetura não oferece suporte ao gerenciamento de acordos SLA, nem suporte nativo a coleta segura de informações, necessária ao monitoramento de serviços IaaS.

Em [21] os autores apresentam o sistema PCMONS, voltado ao monitoramento de nuvens IaaS privadas. O sistema baseia-se em uma arquitetura conceitual composta por três camadas: (i) infraestrutura; (ii) integração; (iii) apresentação. A camada de infraestrutura é formada por recursos básicos como hardware, rede, sistemas operacionais, aplicações etc. A camada de integração fornece uma interface comum de acesso a diferentes tipos de recursos, de modo a permitir que as solicitações vindas dos usuários sejam corretamente executadas na infraestrutura. Finalmente a camada de visão é responsável pela apresentação das informações aos usuários. O sistema PCMONS é composto por diferentes módulos voltados a coleta e integração de informações dos nós, grupos de nós (*clusters*) e VMs; gerenciamento de configuração; armazenamento de informações; e apresentação ao usuário. Diferente da solução proposta nesta dissertação, a solução PCMONS faz o monitoramento apenas de informações de desempenho através monitores instalados na VM. Também não há suporte nativo para monitoramento a partir de informações de SLA, visto que estes acordos apenas são tratados na camada de apresentação.

CloudSec [45] é uma arquitetura voltada a garantir a segurança na execução de VM utilizando mecanismos de monitoramento e verificação de ações de gravação da memó-

ria da VM a ser monitorada. A arquitetura é composta por dois componentes que se baseiam em introspecção de VM: o *VMI Back-end* e o *VMI Front-end*. No componente *VMI Front-end*, um conjunto de APIs permite a coleta de informações sobre a VM em execução, garante o controle de acesso a memória física e registradores da CPU, e também fornece mecanismos para instalação de gatilhos (*triggers*) em regiões de memória de interesse. O componente *VMI Back-end* utiliza o controle do *hypervisor* para suspender a execução da VM para permitir que a arquitetura execute verificações de segurança em regiões de memória onde foram definidos gatilhos. Estes gatilhos permitem que a arquitetura monitore regiões de memória utilizadas por aplicações maliciosas para instalar ganchos (*hooks*) para código malicioso ou ocultar aplicações. Embora a solução seja apresentada como uma arquitetura de monitoramento, ela se comporta como um mecanismo de proteção, visto que ela é capaz de interromper a execução de VMs comprometidas.

4.2 Detecção de Anomalias de Segurança

A detecção de anomalias de segurança está fortemente associada a sistemas de detecção de intrusão (*Intrusion Detection System* - IDS). De modo geral, existem poucos trabalhos que tratam da detecção de anomalias ou intrusões para ambientes de nuvens, sendo que as abordagens utilizadas consideram o monitoramento da rede ou então dos nós da nuvem.

Em [92], os autores apresentam uma arquitetura para detecção de intrusão em grades e nuvens baseada em uma abordagem híbrida que utiliza análise de comportamento e análise baseada em banco de conhecimento. O sistema monitora de forma independente cada nó; para isso, cada nó possui três componentes básicos: um auditor de eventos, o sistema de IDS e o serviço de gerenciamento da base de comportamento e da base de conhecimento. Mecanismos de sincronização e comunicação entre nós permitem a troca de informações ou notificações sobre ataques ocorridos. O sistema de detecção proposto utiliza um conjunto de componentes que devem ser executados em cada nó da nuvem, atuando desta forma como um IDS de *host*. Embora esta abordagem seja capaz de detectar eventos que seriam transparentes a um IDS de rede, a sua aplicação em VMs é limitada devido a possíveis incompatibilidades da solução como o sistema operacional ou com as aplicações em execução na VM.

Mazzariello et al. [60] apresentam um arquitetura de IDS de redes para nuvens IaaS baseada no sistema *Snort* [87] para proteção de VMs em execução em nuvens na solução *Eucalyptus*. Na abordagem adotada pela solução, um único IDS *Snort* é conectado ao *front-end* de rede da infraestrutura da nuvem. Com isto, o IDS tem acesso a todo tráfego destinado às VMs em execução nos nós. A sobrecarga do IDS devido ao grande tráfego presente no *frond-end* é resolvida através da utilização de múltiplas instâncias de IDS conectadas aos nós.

Em [40], os autores apresentam uma solução de IDS de rede distribuída para nuvens denominada *Cloud Intrusion Detection Service* (CIDS), baseada no IDS *Snort*. Diferentemente da solução proposta em [60], CIDS foi concebido para ser oferecido aos usuários através de um modelo de serviço baseado em assinaturas. Ao assinar o serviço, o usuário pode selecionar quais categorias de ataques serão tratados pelo IDS, garantindo deste modo que o serviço atenda às suas necessidades.

As soluções baseadas no sistema *Snort* atuam como um IDS de rede e baseiam-se em assinaturas de tráfego para detectar possíveis ataques. Eventos ocorridos nos hosts ou VMs não são tratados.

Em [28], os autores apresentam uma arquitetura para detecção de intrusão para nuvens SaaS voltada ao acompanhamento de SLAs. Utilizando o *framework* mOSAIC [79], a arquitetura utiliza agentes instalados em diferentes camadas de serviços de nuvens para coletar informações que indicarão a existência de violações de SLA. A arquitetura ainda conta com um mecanismo que avalia se a violação ocorrida se deve a uso indevido, proveniente de um ataque de negação de serviço, ou se é resultado de um pico de utilização. A identificação de ataques baseia-se em um processador de eventos complexos que utiliza correlações temporais e lógicas de eventos medidos nos componentes da nuvem, pemitindo detectar não apenas ataques distribuídos complexos, mas também cenários onde um atacante busca descobrir vulnerabilidades do serviço. Embora a solução utilize informações de desenpenho para detectar tentativas de intrusão, esta abordage baseia-se apenas em ataques de negação de serviço que resultem em violação de parâmetros de QoS do acordo de SLA. Também não são empregados mecanismos que eliminem a necessidade de execução de utilitários na VM, visto que todos os componentes da infraestrutura estão sob controle do provedor.

Modi et al. [63] fazem um levantamento dos diferentes tipos de IDS disponíveis para nuvens. Os resultados obtidos nesse levantamento mostram que a grande maioria dos trabalhos propõem soluções de IDS de rede. As soluções IDS de *host*, quando utilizados para proteção da VM requerem a instalação do mecanismo de detecção na própria VM.

4.3 Conclusão

Este capítulo apresentou diversos esforços propostos nas áreas de monitoramento e detecção de intrusão para nuvens computacionais. Embora algunas das soluções apresentadas campartilham características com o trabalho proposto nesta dissertação, nenhuma das arquiteturas de monitoramento descritas tem seu foco no controle de segurança a partir de acordos *Security-SLA*. De modo semelhante, as soluções para detecção de intrusão apresentadas fazem o monitoramento da rede e de processos em execução nas máquinas utilizando arquiteturas de IDS de rede e *host*. Nenhum dos trabalhos utiliza mecanismos

baseados em informações de desempenho da VM.

Capítulo 5

Solução de Monitoramento

Neste capítulo é apresentada a solução para monitoramento de acordos *Security-SLA*. Serão discutidas as características básicas da arquitetura, seus componentes, funcionalidades e o mecanismo de integração com a arquitetura da nuvem. O capítulo descreve ainda a abordagem utilizada para representação dos acordos de segurança visando seu acompanhamento automatizado. Por fim, serão apresentados os testes realizados para a validação da solução proposta.

5.1 Principais Características

A solução de monitoramento proposta tem o objetivo de acompanhar o cumprimento de acordos *Security-SLA* para máquinas virtuais em execução em nuvens IaaS. Para isto, a arquitetura desta solução foi projetada de forma distribuída e integrada aos componentes da infraestrutura da nuvem. De modo geral, as principais características da arquitetura são:

- **Monitoramento baseado em *Security-SLA***: além do acompanhamento dos parâmetros de segurança que devem ser garantidos pelo serviço, a solução proposta baseia-se no próprio acordo como fonte de informações sobre métricas e os intervalos entre coletas de dados etc. Desta forma garante-se um alto nível de automatização do processo de acompanhamento do acordo.
- **Sistema multi-agente**: a coleta das informações que serão utilizadas na validação do *Security-SLA* é feita através de programas especializados. Cada programa é escrito para coletar a informação desejada a partir de componentes existentes na infraestrutura da nuvem.

- **Monitoramento caixa-preta**: através da técnica de caixa-preta [97], os dados de monitoramento são coletados a partir de componentes do nó da nuvem (sistema operacional, arquivos de log, *hypervisor* e *firewall*). Com isto, não há a necessidade de instalação de ferramentas de monitoramento na VM, que está sob o controle do usuário do serviço.

- **Introspecção de máquina virtual**: a coleta de informações sobre processo em execução na VM ou a utilização de memória não estão disponíveis através da técnica de monitoramento caixa-preta. Para estes casos, a solução oferece suporte a agentes que utilizam introspecção de máquinas virtuais [80] para a inspeção da memória utilizada pela VM.

- **Baixo consumo de recursos**: uma constante preocupação em ferramentas de monitoramento é o consumo de recursos computacionais pelo sistema de monitoramento [56]. Como forma de reduzir o consumo de recursos, o monitoramento é realizado apenas durante o estado de execução da VM e utiliza apenas o conjunto de agentes necessários para a coleta de informações utilizadas para avaliação do acordo.

- **Arquitetura multiplataforma**: a solução proposta suporta o monitoramento em nós com configurações distintas. Isto permite seu uso em diferentes soluções de infraestrutura de nuvens e diferentes tecnologias de virtualização.

5.2 Representação de *Security-SLA*

A forma de representação dos acordos *Security-SLA* é um aspecto importante na solução de monitoramento, visto que todo o processo de acompanhamento do acordo é feito de forma automatizada.

Por isto, a busca por uma linguagem adequada à representação destes acordos considerou diferentes padrões existentes, os quais, a princípio, não eram voltados à representação de acordos de segurança.

Diferente de acordos SLA tradicionais que tem seus parâmetros expressos como valores numéricos [64] (ex. utilização de CPU, tráfego de rede etc), acordos *Security-SLA* possuem uma relação muito forte com políticas de segurança. Apesar desta relação, a representação e avaliação de políticas para verificação do cumprimento de acordos é desnecessária, uma vez que as métricas existentes no acordo devem ser suficientes para indicar se uma política está sendo cumprida.

Por outro lado, a representação destas políticas contribui para a centralização das informações de segurança em um único documento, permitindo que acordos *Security-SLA*

sejam utilizados como fonte de informações para controles de segurança como *firewall*, mecanismos de controle de acesso a recursos da nuvem etc.

Deste modo, foi adotada uma abordagem híbrida para a representação dos acordos, na qual métricas numéricas são utilizadas para a avaliação do cumprimento do acordo, e políticas de segurança podem ser especificadas opcionalmente para uso em mecanismos de controle de segurança.

Diante dos padrões de representação de acordos SLA existentes, optou-se pela utilização da linguagem WSLA, capaz de representar métricas complexas através de fórmulas relacionando outras métricas, descrever controles de tempo (*schedule*) para coleta de dados e descrever expressões condicionais e ações para acompanhamento dos objetivos do acordo.

Para a representação das políticas, observou-se que não existe uma linguagem capaz de representar todos os tipos de políticas de segurança existentes. Diferentes padrões são especializados na representação de tipos de políticas específicas. Por exemplo, o *framework WS-SecurityPolicy* [73] é voltado à especificação de controles de integridade e confidencialidade em *web services*. Já o padrão XACML [71] é voltado a representação de políticas de controle de acesso. Com isto, a adoção de um padrão único para representação de políticas restringiria a utilização da linguagem.

Para tratar a representação de políticas de segurança nos acordos, foi desenvolvida uma extensão da linguagem WSLA. Nesta extensão, foram criados dois novos tipos de objetos da linguagem: (i) o objeto "*Policy*", que fornece um compartimento para representação das políticas de segurança; (ii) o objeto "*SecurityObjective*, que é declarado na seção de obrigações do acordo, e que faz a associação das políticas aos parâmetros que fazem sua avaliação.

Devido ao tamanho dos arquivos, o esquema com a extensão da linguagem WSLA e um exemplo de uso para a representação de acordos *Security-SLA* são apresentados nos Apêndices A e B, respectivamente.

5.3 Componentes da Arquitetura

A infraestrutura de uma nuvem é constituída por diferentes componentes de *hardware* e *software* que operam coordenadamente visando o fornecimento do serviço. Em nuvens IaaS, os principais componentes encontrados são: nós (*nodes*), que compõem o conjunto de recursos oferecidos aos usuários; servidores de gerenciamento, que permitem que os usuários gerenciem suas VMs; sistemas de armazenamento (*storage*) e dispositivos de rede e segurança (*switches, firewalls, routers,* IPS etc).

Para interagir com a infraestrutura de nuvens, a solução de monitoramento é composta por diferentes componentes de *software* que são executados nos nós e nos servidores

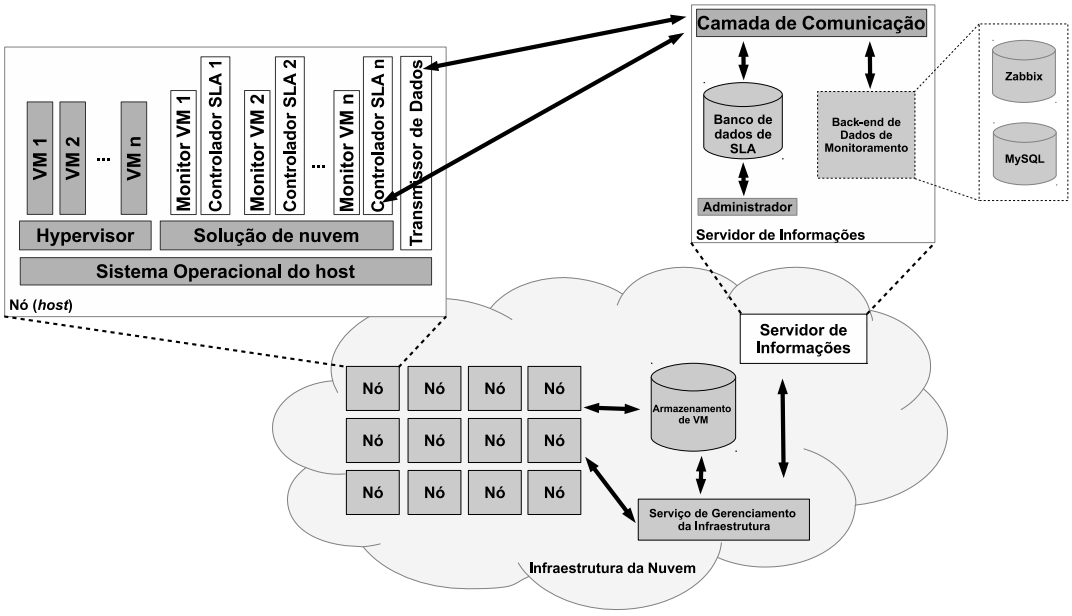


Figura 5.1: Arquitetura da solução de monitoramento

de gerenciamento. A arquitetura da solução também define um componente denominado **servidor de informações** que é responsável pelo gerenciamento de todas as informações relacionadas aos acordos *Security-SLA* e dados de monitoramento coletados. Embora representado como uma única entidade, o servidor de informações pode ser implementado através de vários servidores, como forma de garantir a escalabilidade e redundância dos componentes da solução. A Figura 5.1 apresenta os componentes da arquitetura de monitoramento e a relação destes componentes com a infraestrutura da nuvem.

5.3.1 Camada de Comunicação

A camada de comunicação é o componente que fornece acesso às informações do banco de dados de *Security-SLA* para os demais componentes da arquitetura. Para oferecer um mecanismo de comunicação seguro e padronizado, este componente foi implementado como um servidor de *web services* que abstrai as entidades do banco de dados e fornece um conjunto de operações para cada entidade através do padrão SOAP (*Simple Object Access Protocol*) [95].

A escolha da especificação SOAP sobre outros padrões como REST (*Representational State Transfer*) [29] se deve ao fato do padrão SOAP fornecer mecanismos para controles de transações e suporte para comunicação segura através do padrão *WS-Security* [72].

5.3.2 *Back-ends* de Dados de Monitoramento

Visando a integração da solução de monitoramento a outros sistemas já existentes, o armazenamento dos dados coletados é feito através de módulos que permitem o uso de diferentes repositórios de dados, de modo transparente para o restante da solução.

Cada módulo atua em dois modos diferentes: na preparação do ambiente e na transferência dos dados de monitoramento. No modo de preparação, o módulo do *back-end* é responsável por criar entidades necessárias ao sistema integrado e também gerar identificadores únicos que relacionam os dados nas na solução de monitoramento e no sistema onde os dados serão armazenados. No modo de transferência de dados, além da transferência propriamente dita, é responsabilidade do módulo garantir a integridade dos identificadores de relacionamento e fazer a conversão de tipos de dados quando necessário.

Como forma de avaliar o mecanismo de *back-end* da solução, foram implementados dois módulos de *back-end*. O primeiro tem a função de armazenar as informações de monitoramento em tabelas no próprio banco de dados de *Security-SLA*. Nesta implementação poucos controles foram necessários, uma vez que os dados armazenados utilizavam os mesmos identificadores já existentes na arquitetura e por não ser necessário fazer nenhum tipo de conversão entre os dados. O segundo módulo é responsável pela integração da solução proposta com a ferramenta Zabbix [84] e será descrito em maiores detalhes a seguir.

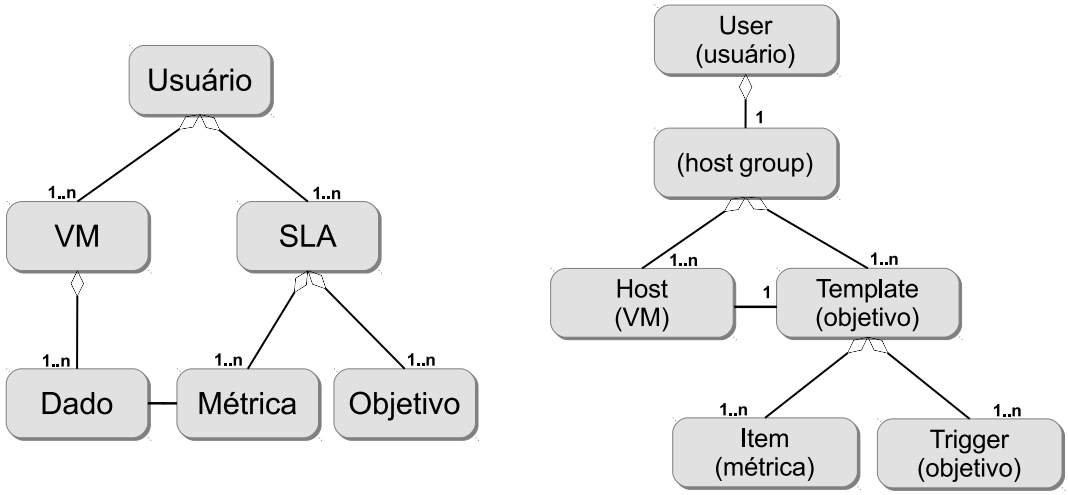
Módulo para o Sistema de Monitoramento Zabbix

Zabbix [84] é uma solução multiplataforma de código aberto para monitoramento de disponibilidade e desempenho de servidores e outros dispositivos em rede. Sua arquitetura oferece mecanismos de descoberta de dispositivos, monitoramento distribuído baseado em agentes e alerta de eventos. A administração dos recursos monitorados é feita através de um sistema *web* multiusuário, que fornece informações textuais e gráficas de acordo com a necessidade do usuário do sistema.

Para reutilizar as facilidades oferecidas pelo sistema de gerenciamento *web* e os mecanismos de alerta e notificação de eventos do Zabbix, foi desenvolvido um módulo de *back-end* responsável por replicar as entidades do modelo de dado da solução de monitoramento para o modelo de dados do Zabbix.

A Figura 5.2, apresenta o mapeamento entre as entidades das duas arquiteturas. Pode-se observar que além da diferença na hierarquia de entidades, nem todas as entidades possuem um equivalente na arquitetura parceira. Como exemplo, as entidades "Dado" na solução de monitoramento, e "*Host group*" no Zabbix não possuem entidades equivalentes.

Outro aspecto importante que foi tratado diz respeito aos identificadores responsáveis por manter o relacionamento entre entidades. No Zabbix, identificadores são controlados



(a) Arquitetura de Monitoramento (b) Zabbix

Figura 5.2: Modelos de relacionamento das entidades das arquiteturas

internamente e não podem ser alterados. Para garantir o relacionamento entre as arquiteturas, os nomes das entidades no Zabbix foram construídos de modo exclusivo, a partir dos identificadores na solução de monitoramento.

Para utilizar o sistema de notificação e alertas do Zabbix no acompanhamento do acordo, o módulo também tem a responsabilidade de converter as expressões existentes nos objetivos do acordo em *triggers* para ações de notificação na arquitetura Zabbix.

A manipulação das entidades de dados do Zabbix foi implementada de forma independente do modelo de seu banco de dados. Para isto, foram utilizadas abordagens distintas para a criação de entidades e a transferência dos dados. A criação e gerenciamento de entidades foi implementada através da interface Zabbix-API, baseado na especificação JSON-RPC [50]. A transferência de dados de monitoramento foi implementada utilizando o utilitário "*zabbix_sender*", que pertence à solução *Zabbix*.

5.3.3 Controlador SLA

O serviço controlador SLA (*sla-manager*) é executado nos servidores de gerenciamento e em cada nó da infraestrutura da nuvem. Sua função é preparar o ambiente e também controlar a execução do monitor de VMs. Para isto, o controlador se comunica com o Servidor de Informações e com a infraestrutura da nuvem, o que permite recuperar informações referentes aos acordos *Security-SLA* e acompanhar as alterações do estado de execução da VM.

A comunicação entre a infraestrutura da nuvem e o controlador é feita através de

hooks na solução de gerenciamento da nuvem. Um *hook* permite que ações sejam tomadas quando eventos específicos ocorrem na infraestrutura da nuvem. No caso da solução de monitoramento, foram tratados eventos relacionados a criação, inicialização, suspensão, migração e término de execução da VM. Em cada um destes eventos uma nova instância do controlador é iniciada, as ações são executadas de acordo com o estado da VM e o controlador é finalizado.

A comunicação entre o controlador SLA e o monitor da VM é feita através de um canal de comunicação exclusiva baseada em *sockets*. Através deste canal, o controlador pode solicitar o início ou suspensão do monitoramento e também o término da execução do monitor, em situações onde a VM é finalizada ou migrada para outro nó.

5.3.4 Monitor de Máquinas Virtuais

O monitor da máquina virtual (*vm-monitor*) é um utilitário *multithreaded* executado nos nós da nuvem que é responsável pelo controle do monitoramento da VM. Cada VM alocada para execução em um nó possui um monitor associado, que permanece em execução durante todo o ciclo de vida da VM. O monitor é finalizado apenas quando a VM termina a sua execução ou quando é migrada para outro nó. Apesar de permanecer residente, o monitor apenas faz a coleta de informações durante o período em que a VM estiver em execução, reduzindo assim a utilização de recursos do nó.

Cada monitor é composto por três tipos distintos de *threads*: o **principal**, responsável pela comunicação com o controlador SLA; o **agendador**, responsável pela controle da execução das tarefas de monitoramento; e os *threads* de **tarefas** que controlam a execução de agentes.

Para coordenar a execução dos agentes, o agendador utiliza as informações de *schedule* existentes no próprio *Security-SLA*. Quando chega o momento de execução da tarefa, o agendador cria o *thread* da tarefa e o arquivo que será utilizado para armazenar as medições coletadas.

O formato do arquivo gerado durante a execução da tarefa segue o exemplo mostrado na Figura 5.3. A primeira linha do arquivo é um cabeçalho que contém o identificador da VM e um registro de data e hora (*timestamp*) da coleta de informações. Nas linhas restantes, são relacionadas as métricas e medições coletadas pelos agentes.

Após a execução de todos os agentes, o *thread* da tarefa é finalizado e o arquivo contendo os dados coletados é movido para a área de transferência (*spool*), onde aguardará o seu processamento pelo transmissor de dados.

```
BEGIN-DATA VM:525 TIMESTAMP:1376414018
proc.threads::46
mib.tcp_currestab::0
net.traffic_out:if0:0
net.traffic_in:if0:0
io.read_requests::11230
io.write_requests::590
io.read_bytes::142392320
io.write_bytes::7517184
io.flush_ops::26
io.rdreq_persec::0
io.wrreq_persec::0
mem.phys_perc_usage::23.09
mem.swap_perc_usage::0.00
mem.pflt_minor::365987
mem.pflt_major::640
mem.pfmajor_psec::0.00
mem.pfminor_psec::0.00
cpu.count::2
cpu.utilization::.7234
netfilter.tcp_reset_in::0
netfilter.tcp_reset_out::0
```

Figura 5.3: Exemplo de um arquivo de dados coletados pelo monitor

5.3.5 Transmissor de Dados

O transmissor de dados (*data-dispatcher*) é o processo responsável pela transferência dos arquivos de medições coletados pelos monitores de VMs para o servidor de informações. O mecanismo utilizado para a transferência baseia-se em uma área compartilhada (*spool*), que é monitorada periodicamente pelo transmissor em busca de novos dados. Ao encontrar novos arquivos o transmissor utiliza o serviço *Data*, presente na camada de comunicação, para enviar os dados ao servidor de informações. Durante o processo de transferência, mecanismos de controle de transações são utilizados para garantir que as informações cheguem corretamente no banco de dados.

5.3.6 Agentes

Os agentes formam a base de toda a arquitetura proposta e são os responsáveis pelas medições das métricas especificadas nos acordos *Security-SLA*. Um agente é um utilitário executável que pode ter formato binário ou ser um *script*, criado em alguma linguagem que seja suportada pelo nó.

A comunicação entre um agente e o monitor da VM é simples. Quando um agente é executado, ele recebe do monitor os parâmetros: identificador da máquina virtual que será monitorada, parâmetros de execução e um conjunto de métricas que serão aferidas. Os resultados obtidos são enviados ao monitor através da saída padrão (*stdout*). O sucesso ou a falha na execução do agente é reportado através do código de retorno ao sistema

operacional e possíveis mensagens de erro são enviadas através do dispositivo de erro padrão (*stderr*).

Durante sua execução, o agente conta com um mecanismo fornecido pela solução que fornece facilidades para persistência temporária de dados e obtenção de informações adicionais como: tipo de solução de nuvem utilizada, *hypervisor*, interfaces de rede da VM e caminhos para diretórios importantes.

Técnicas de Coleta de Informações

A arquitetura da solução de monitoramento não especifica um método rígido para a coleta de medições e permite que diferentes mecanismos sejam utilizados para a obtenção de informações. Agentes podem ser desenvolvidos para coletar informações através protocolos de SNMP, HTTP, ou ainda através do acesso a bancos de dados ou outros serviços externos.

Contudo, por se tratar de uma solução para a coleta segura de informações, todo o ambiente desenvolvido é voltado ao monitoramento de VMs livre de interferências por parte do usuário ou de aplicativos em execução na VM. Para isto, a solução fornece suporte a duas técnicas para coleta. A principal é o monitoramento caixa-preta [97], onde as medições são realizadas externamente à VM, a partir de componentes do sistema operacional do nó como *firewall*, arquivos de log, *hypervisor* e interfaces de rede virtuais.

Embora esta abordagem se mostre efetiva para a coleta de um grande conjunto de informações, dados sobre a utilização da memória ou processos em execução na VM não estão disponíveis através desta técnica, uma vez que tais informações estão sobre o controle do sistema operacional da VM.

Para casos onde o monitoramento caixa-preta não é um mecanismo viável, a solução de monitoramento oferece suporte à introspecção de máquina virtual (*Virtual Machine Introspection - VMI*) [36] a partir da biblioteca LibVMI [93]. A introspecção permite o acesso à memória da VM a partir do *hypervisor*, o que possibilita a consulta de informações das estruturas de dados do *kernel* do sistema operacional da VM. Para que a consulta a estas estruturas seja possível, é necessário primeiramente resolver a chamada lacuna semântica (*semantic gap*) [65, 80], ou seja, atribuir um significado para as regiões de memória da VM. A resolução destas lacunas é o maior desafio para a utilização de introspecção e é resolvida através do uso de algum conhecimento sob o sistema operacional em execução na VM.

Para resolver a lacuna semântica a solução fornece dois tipos de informações aos agentes baseados em introspecção: um mapa de símbolos do *kernel* do sistema operacional e um conjunto de endereços de deslocamento (*offsets*) para campos de interesse dentro dos símbolos do *kernel*. A recuperação destas informações depende de mecanismos de busca por arquivos *system.map* em imagens de discos virtuais e da injeção de *drivers* de

kernel no sistema operacional da VM para a recuperação dos endereços de deslocamento. A implementação de mecanismos mais robustos para resolução de lacunas semânticas a partir de *frameworks* como o *Volatily* [94] é um dos trabalhos futuros para melhoria da solução.

5.3.7 Administrador de Solução

O administrador (*sla-admin*) é um utilitário de linha de comando que permite que gestores da solução de monitoramento façam o gerenciamento das informações existentes no banco de dados. Através dele é possível fazer o gerenciamento de informações de usuários, importação de acordos *Security-SLA*, cadastramento e manutenção de métricas e agentes de coletas de informações.

5.4 Integração com a Infraestrutura da Nuvem

Existem diversas soluções de gerenciamento de nuvens de infraestrutura e embora todas apresentem um conjunto comum de funcionalidades, cada solução apresenta um características distintas em suas arquiteturas. De modo semelhante, diferentes tecnologias de virtualização podem ser suportadas em cada solução, o que dificulta a utilização de uma mesma ferramentas de monitoramento em diferentes soluções de nuvens.

No sistema de monitoramento proposto, o principal mecanismo de integração baseia-se em ganhos (*hooks*) fornecidos pela solução de gerenciamento. Através destes ganchos, presentes nas soluções OpenStack [75], OpenNebula [74] e Eucalyptus [26], a infraestrutura da nuvem faz chamadas ao controlador (*sla-manager*) em eventos de criação, execução, suspensão, migração e finalização das VMs.

Além do mecanismo para recebimento de notificações da infraestrutura, a solução de monitoramento também utiliza uma camada de abstração que permite que os componentes da solução obtenham informações sobre as VMs, o usuário e sobre a própria nuvem. Esta camada baseia-se em um módulo escrito especificamente para a solução de gerenciamento em uso, o que permite que novas soluções de nuvens sejam integradas ao sistema de monitoramento.

Finalmente, o último mecanismo de integração trata o sistema operacional e a tecnologia de virtualização utilizada no nó. O controlador, durante o processo de preparação do ambiente para execucão do monitor, solicita agentes específicos para o sistema operacional e para o *hypervisor* em uso no nó, o que permite que a solução monitore ambientes heterogênios com nós com diferentes configurações.

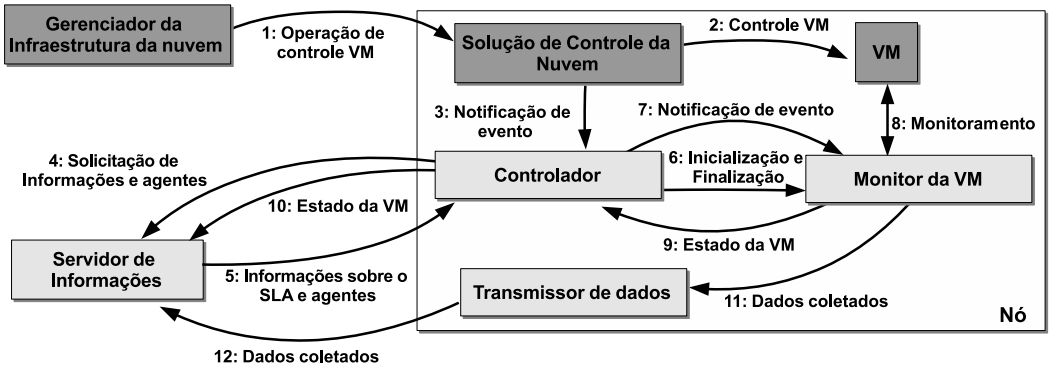


Figura 5.4: Fluxo de execução da solução de monitoramento

5.5 Fluxo de Execução

O fluxo de execução da solução de monitoramento, conforme mostrado na Figura 5.4, está integrada às operações executadas nas VMs pela solução de gerenciamento da nuvem.

Quando um usuário executa uma operação em uma VM através da ferramenta de gerenciamento da nuvem, esta operação é enviada para execução no nó através do controle da solução da nuvem (fluxos 1 e 2). Neste ponto, a infraestrutura da nuvem notifica o controlador da solução de monitoramento sobre o evento ocorrido (fluxo 3), utilizando um mecanismo de *hook*.

De acordo com o tipo de operação solicitada pelo usuário, o controlador pode executar diferentes atividades na solução de monitoramento. Estas atividades envolvem: a obtenção de informações sobre SLA e a transferência de agentes necessários para o monitoramento da VM (fluxo 4); a execução ou finalização do monitor de VM (fluxo 6); o envio de notificações sobre a mudança de estado da VM para o monitor (fluxo 7); e a atualização do estado da VM no banco de dados a partir das informações recebidas pelo monitor (fluxos 9 e 10).

Durante a execução do monitor, as informações recebidas do controlador determinam o início ou fim de ciclos de coleta de informações. Quando a coleta de informações está ativa (a VM está em execução), o monitor executa as tarefas de monitoramento e recebe as medições proveniente dos agentes (fluxo 8). Ao final da execução de cada tarefa, o monitor envia os dados ao transmissor (fluxo 11), que por sua vez, transfere estes dados ao servidor de informações para armazenamento e avaliação de cumprimento através do *back-end* de dados de monitoramento (fluxo 12).

5.6 Validação da Arquitetura

Para validar a arquitetura proposta para a solução de monitoramento, um protótipo foi desenvolvido e testado em um pequeno ambiente de nuvem baseado na solução *OpenNebula* [74], em uma máquina com processador Intel i7 Quad core 2.8 Ghz e 4GB RAM executando o sistema operacional Gentoo Linux e *hypervisor* KVM.

Os testes realizados durante a validação tiveram o objetivo de determinar o consumo de CPU e o volume de tráfego de rede produzida pela solução. Para os testes de utilização de CPU, uma série de experimentos foram realizadas utilizando diferentes intervalos de monitoramento e com diferentes números de máquinas virtuais. Cada experimento teve duração de doze horas e foram coletadas medições do nó e do Servidor de Informações. As Tabelas 5.1 e 5.2 apresentam os valores percentuais de utilização de CPU no servidor de informações e no nó, respectivamente.

Tabela 5.1: Utilização de CPU do servidor de informações

Intervalo de Coleta	**Uso de CPU**		
	1 VM	**2 VMs**	**3 VMs**
30 seg	0.04%	0.09%	0.11%
1 min	0.02%	0.05%	0.09%
5 min	0.01%	0.02%	0.03%

Tabela 5.2: Utilização de CPU no nó

Intervalo de Coleta	**Uso de CPU**		
	1 VM	**2 VMs**	**3 VMs**
30 sec	0.12%	0.24%	0.36%
1 min	0.10%	0.20%	0.30%
5 min	0.10%	0.20%	0.30%

De modo geral, os resultados obtidos mostraram que a utilização média de CPU pelos componentes da arquitetura representaram menos de 1% do tempo total de CPU, no nó e no servidor de informações, mesmo quando múltiplas instâncias de VMs foram monitoradas simultaneamente. Como esperado, a utilização mais significativa de CPU ocorreu no nó, onde são executas as tarefas de gerenciamento do monitoramento e coleta de dados através de agentes.

É importante ressaltar que os testes conduzidos tinham o objetivo de avaliar o custo da execução dos componentes estáticos da arquitetura (controlador, monitor, módulos de *back-end* e transmissor de dados), por este motivo, os agentes empregados apresentavam baixo consumo de recursos. Em ambientes reais, a execução dos agentes para a coleta de

Tabela 5.3: Tamanho em bytes das mensagens SOAP utilizadas na arquitetura

Operação	**Tamanho**	
	Solicitação	**Resposta**
getCustomerID	456	482
getAgreementID	510	665
getAgreement	466	478
checkAgreement	482	475
getProviders	492	504
getAgent	492	550
getVM	499	453
setState	558	463
sendData	465	469

informações pode resultar no uso intensivo de CPU dependendo do tipo de informação coletada e da técnica de coleta utilizada. Um estudo mais detalhado sobre os custos de execução de tais agentes é um dos trabalhos futuros relacionados a esta dissertação.

Nos testes de utilização de rede foi medido o tráfego gerado pela comunicação entre os processos em execução no nó da nuvem (controlador e transmissor) e o servidor de informações. Este tráfego é estritamente composto por mensagens SOAP representando operações de solicitação de informações sobre acordos, transferência de agentes e dados, e notificações de mudança de estado da VM.

A chamada a uma operação SOAP é composta por uma solicitação e uma resposta. Cada uma por sua vez é formada por um envelope XML e um conjunto de dados que serão transportados (*payload*). Embora o tamanho do envelope da mensagem seja fixo, o tamanho do *payload* pode variar de poucos *bytes* até vários *kbytes* em operações que envolvem a transferência de acordos e agentes, tornando difícil estimar o tamanho total das mensagens. Por esta razão, a Tabela 5.3 apresenta apenas o tamanho dos envelopes de mensagens SOAP de solicitação e resposta das operações disponibilizadas pela camada de comunicação.

Os resultados obtidos mostram que apesar do padrão SOAP ser considerado pesado por fazer toda a comunicação através de mensagens XML, o tamanho dos envelopes das mensagens não geram tráfego expressivo. Durante os experimentos observou-se que a maior intensidade de tráfego ocorre no momento da criação da VM, onde é necessário fazer a transferência do acordo e dos agentes de monitoramento utilizados pelo monitor. Após esta fase, o tráfego gerado é proveniente de operações de transferência de dados (*sendData*), cujo intervalo depende dos *schedules* existentes no acordo.

5.7 Conclusão

Este capítulo apresentou a arquitetura da solução de monitoramento proposta para o acompanhamento de acordos *Security-SLA*. Foram descritas as principais características da solução, seus componentes, e técnicas suportadas para a coleta segura de informações. Também foi discutida a abordagem utilizada para a representação de acordos *Security-SLA*, o fluxo de execução da solução e sua interação com os componentes da nuvem. Finalmente, foram apresentados os resultados dos testes de utilização de CPU e tráfego de rede gerados pela solução.

Capítulo 6

Detecção de Anomalias de Segurança

A detecção de ataques a máquinas virtuais em ambientes de nuvens utiliza sistemas de detecção de intrusão (*Intrusion Detecion System* - IDS) instalados na infraestrutura de rede da nuvem, nos nós e nas próprias VMs. Visando minimizar a dependência de mecanismos instalados na VM, este capítulo apresenta um estudo sobre a detecção de ataques a partir de informações de desempenho. Será discutida a metodologia utilizada, bem como os resultados obtidos. Por fim, será apresentada uma arquitetura para a detecção de anomalias que pode ser utilizada como uma alternativa a mecanismos de detecção convencionais, ou seja, que exijam a instalação de ferramentas na máquina virtual.

6.1 Metodologia

O estudo sobre o uso de informações de desempenho para a detecção de anomalias de segurança baseia-se no trabalho conduzido por Avritzer et al. [6]. Nesse trabalho, foram utilizados dados de desempenho de máquinas reais para a identificação de eventos de ataques de negação de serviço (*Denial of Service* - DoS), estouro de *buffer*, *sql-injection* e *man-in-the-middle*.

Diferentemente da metodologia utilizada em [6] que utilizava máquinas reais com sistema operacional *Windows*, este estudo utilizou máquinas virtuais com sistema operacional *Linux*. O mecanismo usado para a coleta da informação também difere nas duas abordagens. Nas máquinas reais, foram empregados mecanismos baseados na infraestrutura de instrumentação nativa do sistema operacional (*Windows Management Instrumentation API*) da máquina sob ataque. Para a coleta dos dados em VMs, foi empregada a solução de monitoramento proposta, onde a priori, não havia conhecimento sobre quais processos estavam em execução.

6.1.1 Cenários de Testes

Com o objetivo de aproximar os cenários de testes ao tipo de uso das VMs em execução em nuvens, novos cenários foram considerados durante a execução deste estudo. Nos testes conduzidos em máquinas reais, os ataques realizados baseavam-se apenas no serviço HTTP e o papel ocupado pela máquina em estudo variou de acordo com o teste. Nos testes com VMs, todos os cenários baseiam-se no uso da VM como um servidor. Foram conduzidos testes de ataques a serviços de correio SMTP e também testes considerando diferentes configurações e implementações do mesmo serviço. A seguir serão descritos em mais detalhes todos os testes realizados durante o estudo.

Ataques de Negação de Serviço

O objetivo de ataques DoS é fazer com que um serviço deixe de estar disponível para seus usuários. Para isto, os ataques utilizam técnicas que causam o consumo excessivo de recursos computacionais (memória, CPU, rede), fazendo com que o serviço se torne inoperante.

Nos testes realizados em VMs, foram executados dois tipos de ataques, o *HTTP Flood* nos serviços *web* e o *SMTP Flood* no serviço de correio eletrônico. Em ambos os casos, um número elevado de conexões são abertas para o serviço sob ataque.

Estouro de *buffer*

Em ataques de estouro de *buffer*, o atacante utiliza falhas da aplicação que permitem o armazenamento de informações em memória com tamanho superior ao previsto pela aplicação. Com isto, é possível sobrescrever regiões da memória que podem levar a aplicação a um estado de execução errática ou até mesmo a permitir a execução de código malicioso.

Nos testes conduzidos durante o estudo, uma aplicação com uma vulnerabilidade em seu *buffer* de dados foi disponibilizada através de um servidor *web*, permitindo que o atacante enviasse dados maiores que os esperados através de parâmetros passados pela URL da aplicação.

Ataque de *SQL-injection*

O ataque de *SQL-injection* tem como alvo o banco de dados da aplicação. Neste ataque é utilizada uma vulnerabilidade da aplicação que permite que sequências de comando SQL seja enviado ao servidor juntamente com os dados válidos. Quando bem sucedido, ataques *SQL-injection* podem resultar no vazamento de informações, ou ainda permitir a exclusão de dados ou objetos do banco de dados.

Tabela 6.1: Organização do teste

Fase \ Etapa	Pré-ataque (5 minutos)	Ataque (1 minuto)	Pós-ataque (4 minutos)
Baseline	Tráfego padrão	Carga extra	Tráfego padrão
Ataque	Tráfego padrão	Ataque	Tráfego padrão

O procedimento de teste utilizado para ataques *SQL-injection* baseia-se em uma aplicação *web* com uma vulnerabilidade que permite a execução de comandos SQL enviados como entrada de dados para aplicação. Durante os testes, foram executados dois tipos de ataques. No primeiro, a execução do código SQL ocorre com sucesso (*craft*) e o atacante tem acesso a informações do banco de dados. No segundo, o comando executado causa um erro na aplicação (*crash*).

Testes de Diferenças de Configuração e Implementação de Serviço

Os testes sobre diferenças de configuração têm o objetivo de analisar o efeito das alterações realizadas em parâmetros de configuração do serviço no comportamento da VM. Para isto, foi utilizado o servidor HTTP *Apache* e foram testadas modificações em parâmetros relacionados ao número de *threads* simultâneos em execução e o número de requisições atendidas por cada *thread*.

De modo semelhante, os teste sobre diferenças de implementação visam o estudo da alterações de comportamento da VM causados pelo uso de diferentes implementações do mesmo serviço. Para isto, foram utilizados os servidores HTTP *Apache*, *Nginx* e *Lighttpd*.

6.1.2 Organização dos Testes

Seguindo a metodologia original, cada teste teve duração de 20 minutos com intervalos de 5 segundos entre as coletas de dados. Os testes foram divididos em duas fases, cada uma com 3 etapas, conforme mostrado na Tabela 6.1.

Na primeira fase (*baseline*), as informações coletadas referem-se à execução normal da VM. Durante a primeira etapa (pré-ataque), com duração de 5 minutos, o serviço em teste recebe um tráfego composto por requisições válidas (tráfego padrão). Esta etapa tem a função de identificar o comportamento da VM durante a operação normal do serviço. Na segunda etapa (ataque), com duração de 1 minuto, um conjunto adicional de requisições (carga extra) é encaminhada ao serviço. Esta etapa tem objetivo de fornecer uma referência para situações onde o serviço encontra-se em operação normal, mas apresenta um volume elevado de requisições que necessitam assim mais recursos da máquina. Finalmente a etapa de (pós-ataque), o número de requisições geradas volta ao mesmo nível da

etapa pré-ataque durante os 4 últimos minutos.

Na segunda fase (ataque) é executado efetivamente o ataque ao serviço. As etapas pré e pós ataque têm as mesmas características de suas equivalentes na fase *baseline*. Já na etapa de ataque, são enviadas requisições maliciosas com o objetivo de causar a perturbação do serviço e alterar o comportamento da VM.

É importante destacar que o tráfego padrão, carga extra e o próprio ataque são dependentes do teste e da técnica utilizada. No caso do teste de ataque DoS em servidores HTTP, foi utilizado um tráfego de 100 conexões simultâneas por segundo para a carga válida e 200 conexões para a carga extra. Na etapa de ataque da segunda fase, foram abertas conexões suficientes para levar o servidor ao estado de negação do serviço.

Durante a execução dos testes, um conjunto padrão de parâmetros foi coletado. Neste conjunto, foram utilizados os dez parâmetros originalmente analisados em máquinas reais e cinco novos relacionados a rede e requisições de I/O. O conjunto completo é apresentado na Tabela 6.2.

Tabela 6.2: Parâmetros de desempenho monitorados

Tipo	**Parâmetro**	**Unidade**
CPU	Utilização da CPU	%
Memória	Uso memória física	%
	Uso swap	%
Rede	Tráfego de entrada	bytes/s
	Tráfego de saída	bytes/s
	TCP Reset OUT	resets
	Conexões TCP	conexões
I/O	Requisições de leitura	req/s
	Requisições de gravação	req/s
	Requisições de flush	requisições
Processos	Threads ativos	threads
	Major page faults	pf/s
	Minor page faults	pf/s
	Workset size	bytes
	Virtual size	bytes

6.1.3 Coleta de Dados

Para a realização dos testes em máquinas reais foram empregadas diferentes ferramentas para a geração do tráfego, ataque e para a coleta das informações. Visto que todos os testes executados baseavam-se no serviço HTTP, o trabalho original utilizou a ferramenta *DoSHTTP* [86] para a geração do tráfego válido e também e do tráfego dos ataques *HTTP*

Flood. Para os testes em VMs foram criados *scripts bash* utilizando as ferramenta *curl* e *netcat*, que além de replicarem o comportamento da ferramenta *DoSHTTP*, também permitiram o seu uso em modo não interativo, facilitando a automação dos testes.

Para a coleta de informações durante a execução dos testes, foi utilizada apenas a solução de monitoramento proposta no Capítulo 5. Para isto, foram desenvolvidos 5 agentes, um para cada tipo de parâmetro apresentado na Tabela 6.2. Também foi criado um acordo SLA contendo todas as métricas envolvidas e o intervalo entre as coletas. Finalmente, para facilitar a recuperação e análise das informações, foi utilizado o *backend* para o banco de dados da solução de monitoramento como forma de armazenamento das informações coletadas.

6.2 Análise dos Resultados Obtidos

De modo geral, os resultados obtidos nos testes realizados mostraram que é possível identificar as perturbações causadas por diferentes tipos de ataques às VMs em execução em nuvens, de forma similar aos resultados obtidos em testes com máquinas reais. Nas próximas seções, são discutidas as principais características dos dados obtidos durante a condução do estudo.

6.2.1 Ataques de Negação de Serviço

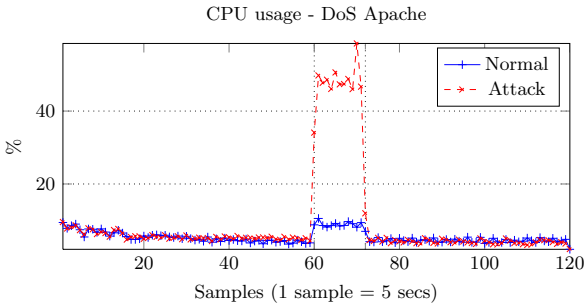
A Figura 6.1 mostra os parâmetros de desempenho coletados em um servidor *Apache HTTPD* sob um ataque DoS. Nos gráficos apresentados, pode-se observar a elevação da utilização de CPU, memória e número de *threads* em execução, durante todo o período de ataque (amostras 60 até 72). O mesmo comportamento foi verificado em outros parâmetros como requisições de leitura, tráfegos de entrada e saída.

Diferente do comportamento apresentado pelo servidor *web*, o ataque ao serviço de correio através da técnica *SMTP Flood* [14, 8], fez com que certos recursos tivessem seu uso reduzido durante o ataque (ver Figura 6.2).

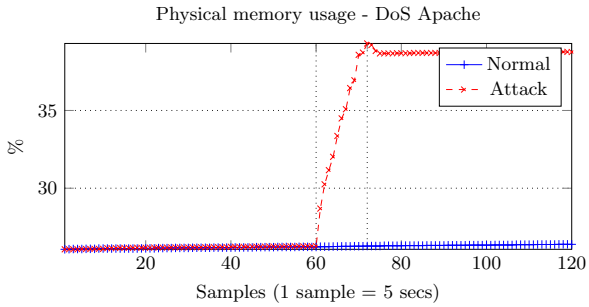
Este comportamento ocorre pois o elevado número de conexões resulta em um elevado número de processos que causam o aumento do consumo de memória. Paralelamente, como não há tráfego SMTP nas conexões abertas, os processos permanecem em um estado ocioso causando a redução do uso de CPU e requisições de I/O.

6.2.2 Diferenças de Configuração e Implementação

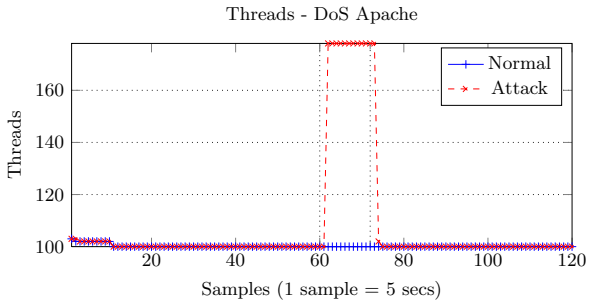
Um dos cenários abordados neste trabalho considera o efeito de mudanças de configuração de um serviço na assinatura de ataque. Para estas análises foram conduzidos novos testes



(a) Uso de CPU

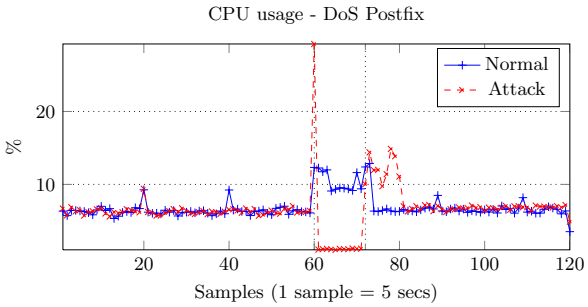


(b) Uso de memória

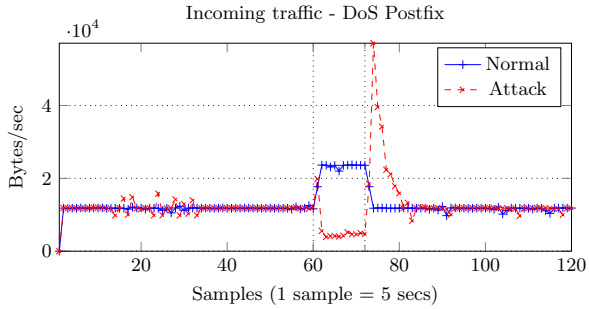


(c) Threads em execução

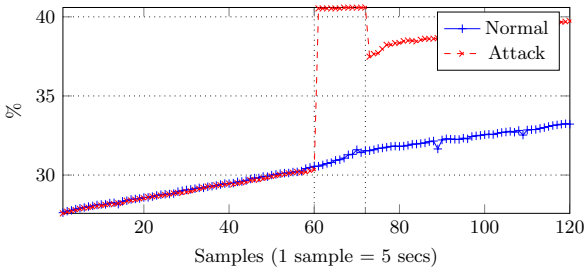
Figura 6.1: Assinatura do ataque DoS ao servidor Apache HTTPD



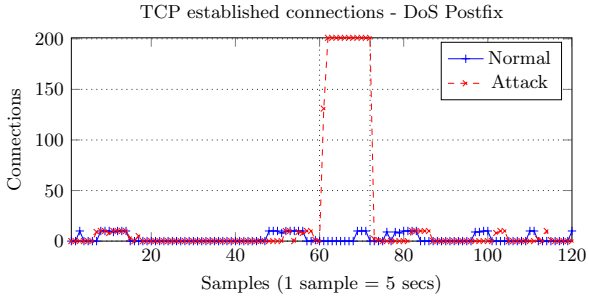
(a) Uso de CPU



(b) Tráfego de entrada

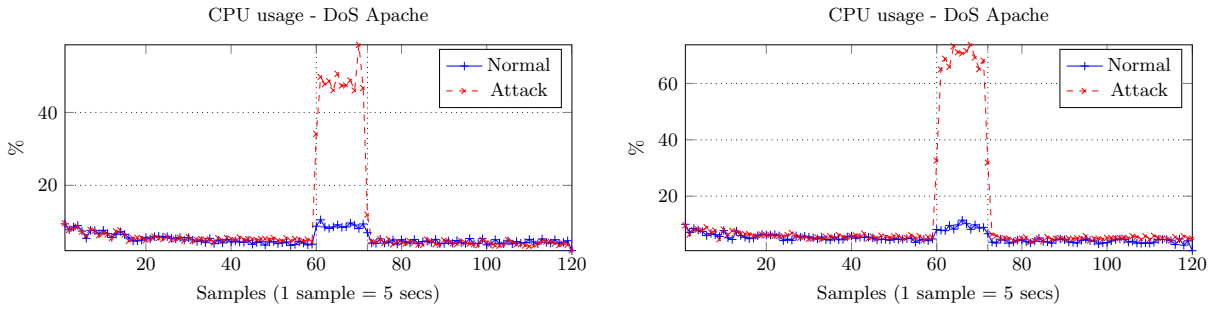


(c) Uso de memória



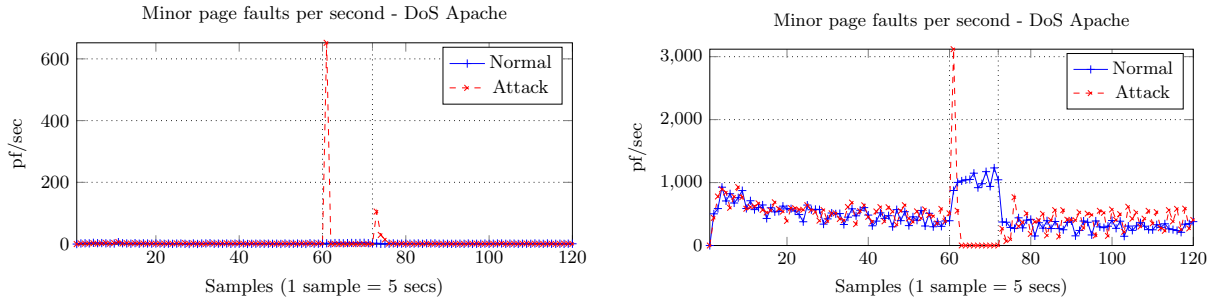
(d) Conexões TCP ativas

Figura 6.2: Assinatura do ataque DoS ao servidor *Postfix*



(a) Configuração padrão (b) Configuração modificada

Figura 6.3: Variação no padrão de utilização de CPU

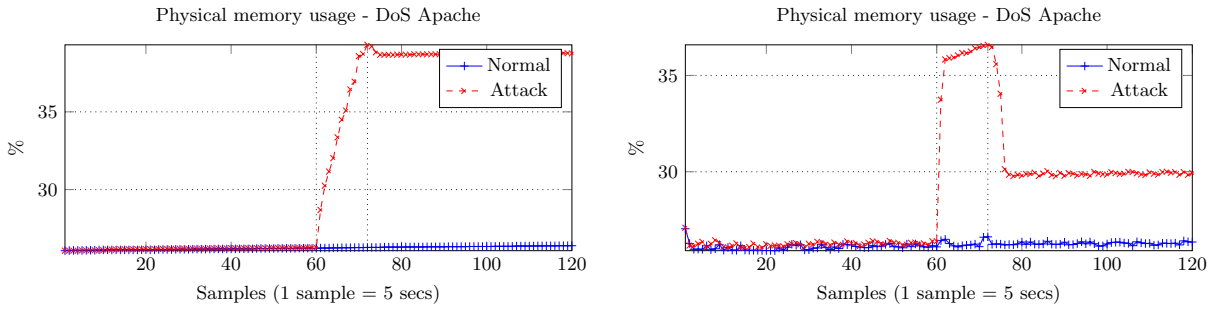


(a) Configuração padrão (b) Configuração modificada

Figura 6.4: Variação no padrão de geração de page faults

de ataque DoS no servidor *Apache HTTP*. Com a mudança do número máximo de *threads* e o número de requisições atendidos por cada *thread*. Foram observadas elevações no uso de CPU (Figura 6.3) e maior número de *page faults* (Figura 6.4), causado pelo aumento no número de *threads*. Paralelamente, a redução do número de requisições atendidas por cada *thread* criou um novo padrão de comportamento que resultou na redução da utilização de memória logo após o ataque (Figura 6.5), visto que a memória alocada para cada *thread* era liberada após o término do atendimento da requisição.

As diferentes implementações do serviço também resultam em comportamentos distintos da VM durante um ataque. Para demonstrar esta afirmação, foi avaliada uma característica distinta de cada um dos servidores testados: o uso de *threads*. O servidor *Apache* apresenta uma implementação *multi-thread*, já o *Nginx* utiliza múltiplos processos, enquanto o *Lighttpd* utiliza apenas um processo. A Figura 6.6 apresenta a utilização de threads nos diferentes servidores testados.



(a) Configuração padrão (b) Configuração modificada

Figura 6.5: Variação no padrão de utilização de memória

6.2.3 Ataques de estouro de *buffer*

Durante o teste de estouro de *buffer*, observou-se um aumento significativo do uso de CPU, conforme apresentado na Figura 6.7(a). Este aumento é explicado pelas proteções do sistema operacional e da CPU à tentativas de acessos ilegais à memória. Também foi observado o aumento do uso de memória (Figura 6.7(c)) e dos *threads* (Figura 6.7(b)) em execução, o que se deve a demora na finalização do processo causador do estouro do *buffer* e a criação de novos processos para atendimento de outras requisições.

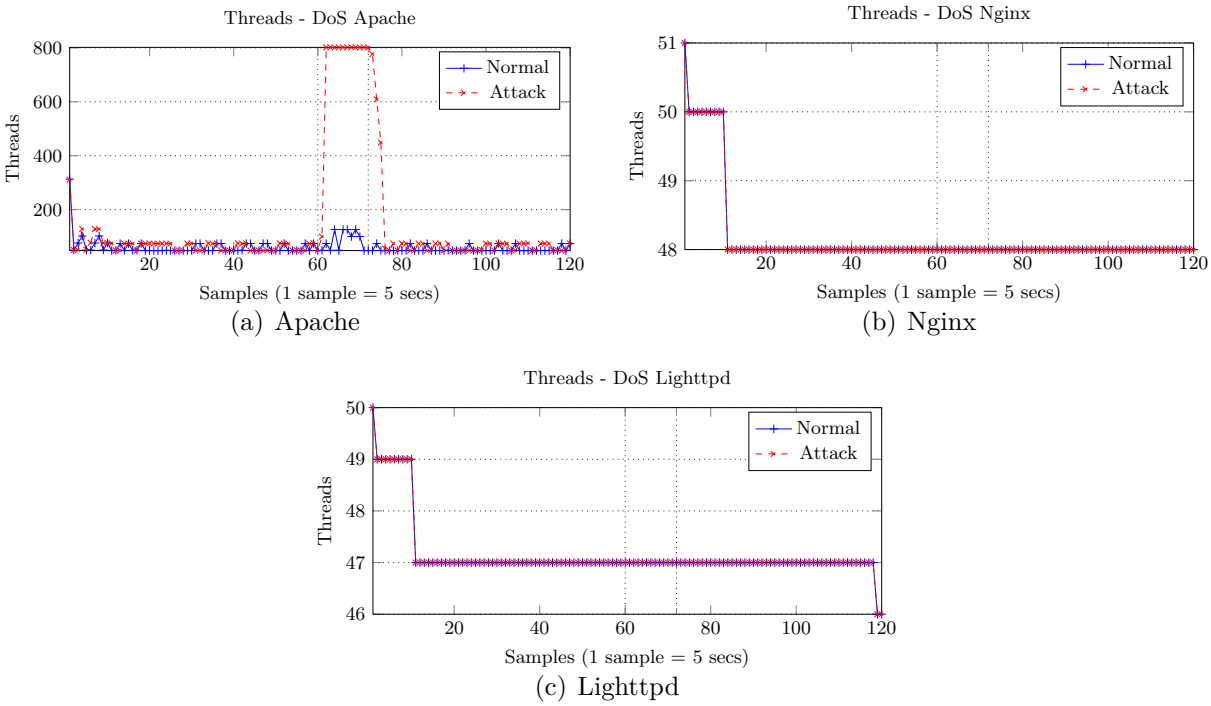
6.2.4 Ataques *SQL-injection*

Nos testes de *SQL-injection* realizados em máquinas reais, o uso do parâmetro "Tráfego de saída" foi indicado para a detecção dos ataques *craft* e *crash*. Durante os testes em VMs não foram encontradas evidências que sustentem esta afirmação. Conforme apresentado na Figura 6.8, o tráfego de saída durante o ataque ficou próximo dos níveis do teste com tráfego válido. Este comportamento também foi observada em outros parâmetros coletados.

É importante ressaltar que apenas o uso do parâmetro de tráfego de saída não é apropriado para a detecção de ataques visto que nem todos os comandos SQL resultam em aumento tráfego. Por exemplo, comandos para exclusão de registros (*Delete from*) ou exclusão de objetos do banco de dados (*Drop*) normalmente não retornam informações após a sua execução.

6.3 Arquitetura de Detecção de Anomalias

Diante dos resultados obtidos, verificou-se que apesar da efetividade da identificação de ataques através de assinaturas de desempenho, a criação de um sistema de detecção



(a) Apache (b) Nginx

(c) Lighttpd

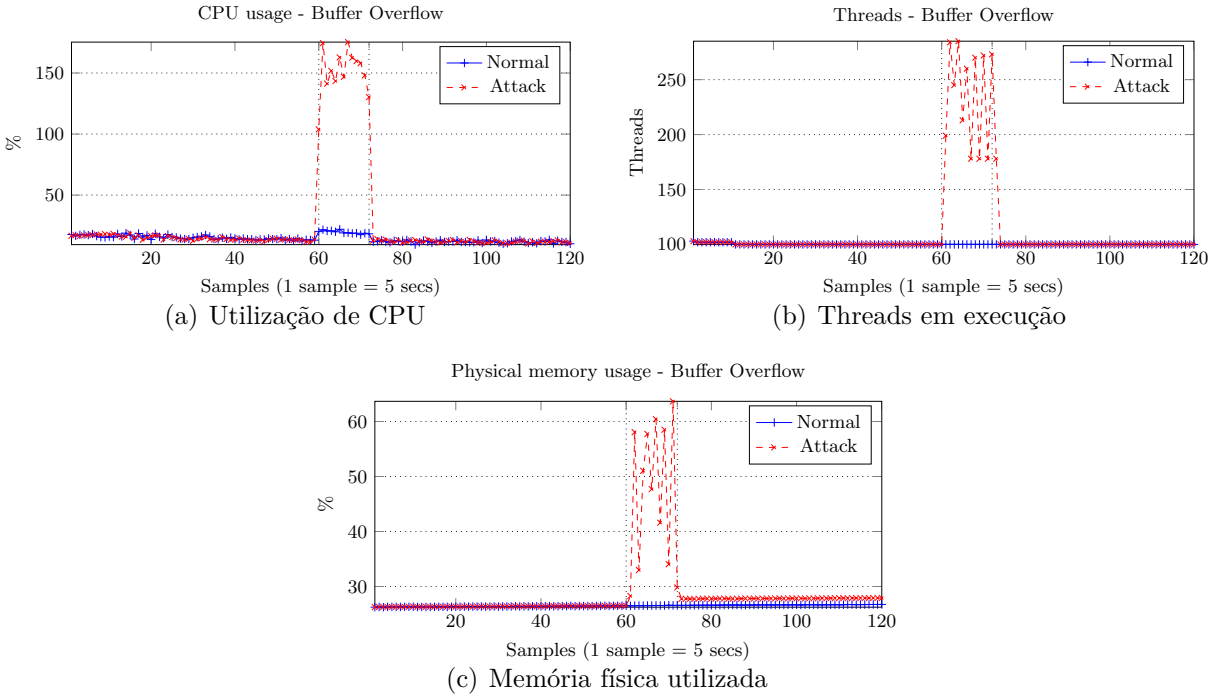
Figura 6.6: Utilização de threads das implementações do serviço HTTP

de anomalias baseadas nestas assinaturas não poderia utilizar um banco de dados de assinatura único, visto que tais assinaturas são totalmente dependentes da máquina nas quais foram geradas. Deste modo, uma arquitetura para detecção de anomalias deve ser baseada em mecanismos de aprendizado de máquina, treinada especificamente para as assinaturas geradas pela máquina sendo monitorada.

Para fazer a avaliação deste método, foram realizados novos experimentos com ataques DoS a servidores *Apache*, *Nginx* e *Lighttpd*. Foram gerados dois conjuntos de dados para cada servidor. O primeiro conjunto, utilizado para o treinamento do modelo, era composto por dados coletados durante um período de 20 minutos. O segundo conjunto, contendo os dados a serem avaliados, foram coletados durante um período de 24 horas. Em ambos os conjuntos foram gerados ataques e tráfego válidos com durações aleatórias e em diferentes grandezas.

O intervalo entre as coletas de informações foi de 5 segundos, o que resultou em conjuntos com 240 eventos (conjunto de treinamento) e 17280 eventos (conjunto de dados).

Utilizando um ambiente de análise e mineração de dados composto pelas ferramentas KNIME [52] e Weka [90], foram testados diferentes algoritmos de classificação *k-Nearest Neighbour* (KNN), *Support Vector Machines* (SVM) e *Multilayer Perceptron*, descritos em detalhes em [10], [18] e [76], respectivamente. Nestes testes, foi observado que o



(a) Utilização de CPU

(b) Threads em execução

(c) Memória física utilizada

Figura 6.7: Utilização de recursos durante ataque de estouro de *buffer*

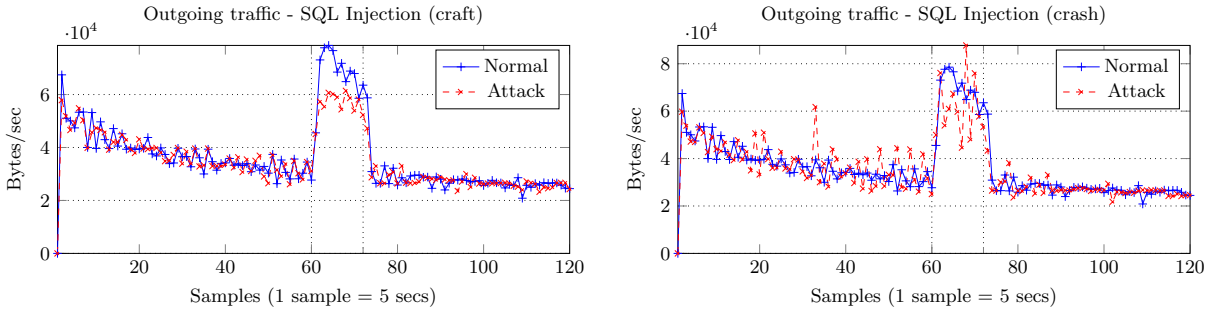
algoritmo SVM utilizando *kernel* linear, apresentou os melhores resultados no processo de classificação dos eventos.

Baseado nos trabalhos [32] e [51] foi avaliada a utilização de um mecanismo seletor de parâmetros, baseado em um algoritmo genético. Através deste mecanismo foi possível eliminar parâmetros desnecessários para a formação da assinatura do ataque e também reduzir o custo de processamento do modelo de classificação.

A Tabela 6.3 apresenta a distribuição dos valores de verdadeiros positivos (VP), falsos positivos (FP), verdadeiro negativos (VN) e falsos negativos (FN), para a detecção de ataques utilizando todos os parâmetros (coluna Todos) e utilizando apenas os parâmetros selecionados pelo algoritmo genético (coluna Filtro). Pode-se observar que a utilização do seletor contribuiu para a redução significativa do número de falsos positivos.

Tabela 6.3: Resultado da detecção de ataques

	VP		**FP**		**VN**		**FN**	
	Todos	Filtro	Todos	Filtro	Todos	Filtro	Todos	Filtro
Apache	1416	1413	1042	11	14826	15857	0	3
Nginx	1374	1377	282	6	15621	15897	9	6
Lighttpd	938	939	44	48	16301	16297	2	1



(a) Ataque Craft (b) Ataque Crash

Figura 6.8: Tráfego de saída durante ataque *SQL-Injection*

A Tabela 6.4 apresenta as métricas de avaliação do modelo: a precisão, representada pela fórmula $P = \frac{VP}{VP+FP}$, indica a porcentagem de eventos classificados corretamente como ataque dentre todos os que foram classificados como ataque; *recall*, $R = \frac{VP}{VP+FN}$, representa a porcentagem de eventos classificados corretamente como ataque dentre todos os eventos que efetivamente são ataque; *F-measure*, $F = \frac{2 \times R \times P}{R+P}$, é a média harmônica entre precisão e *recall*; a acurácia, $A = \frac{VP+VN}{VP+FP+VN+FN}$, indica a porcentagem de eventos corretamente classificados.

Nos resultados apresentados, observa-se que o uso do seletor de parâmetros aumentou a precisão da classificação do conjunto de dados do *Apache* em 42% e do *Nginx* em 16%, enquanto que a perda de precisão apresentada no conjunto de dados do *Lighttpd* foi inferior a 0,5%. De forma semelhante, a acurácia do modelo de classificação teve um aumento médio de 2,5% se comparado à classificação utilizando todos os parâmetros dos conjuntos de dados.

Tabela 6.4: Métricas de avaliação do modelo de classificação

	Precisão		**Recall**		**F-Measure**		**Acurácia**	
	Todos	Filtro	Todos	Filtro	Todos	Filtro	Todos	Filtro
Apache	0,5761	0,9923	1,0000	0,9979	0,7310	0,9951	0,9397	0,9992
Nginx	0,8316	0,9959	0,9938	0,9954	0,9050	0,9957	0,9832	0,9993
Lighttpd	0,9552	0,9512	0,9974	0,9989	0,9759	0,9745	0,9973	0,9972

Baseado nestes resultados, a Figura 6.9 apresenta uma arquitetura conceitual para detecção de anomalias de segurança baseado em assinaturas de desempenho. Entre os componentes desta arquitetura estão: o sistema de classificação SVM (preditor, classificador); o seletor de parâmetros, baseado em um algoritmo genético; um pré-processador dos dados de monitoramento, responsável pela validação e normalização dos dados; e um gerador de ataques, responsável por simular diversos tipos de ataques contra a máquina

virtual e desta forma gerar o banco de assinaturas necessário ao treinamento do modelo de classificação.

Como a configuração da VM pode mudar com frequência devido a atualização do *software*, atualizações do próprio sistema operacional e também a instalação de novos aplicativos, a geração destas assinaturas de ataques deve ser executada periodicamente ou então por solicitação do próprio usuário. Esta tarefa pode ser feita diretamente na VM, ou, em uma instância clone, em execução em ambiente controlado, evitando assim a interrupção dos serviços fornecidos pela VM.

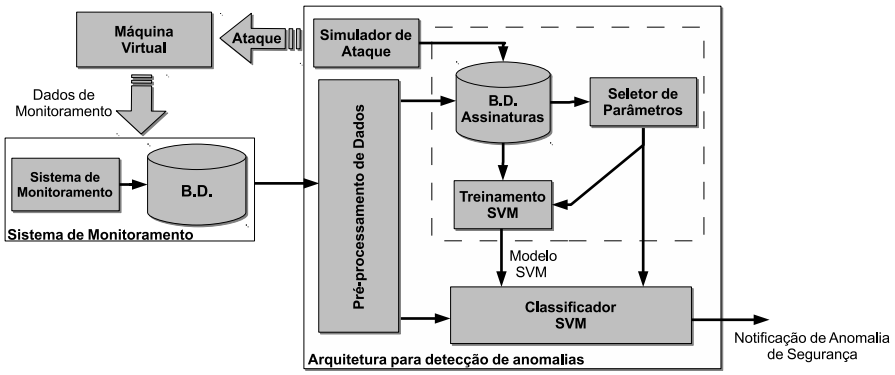


Figura 6.9: Arquitetura para detecção de anomalias de segurança

A principal característica desta arquitetura é o desacoplamento do sistema de coleta dos dados (monitoramento) e o sistema de detecção. Isto permite que esta arquitetura seja utilização como ferramenta para a melhoria da segurança do serviço fornecido por um provedor de IaaS, ou como um serviço de detecção de anomalias no modelo *Security as a Service*, que pode ser fornecido para usuários de nuvens de modo geral.

6.3.1 Integração com Acordos *Security-SLA*

A detecção de anomalias pode ser visto como um dos controles implementados na infra-estrutura da nuvem para a garantia da segurança do serviço de máquinas virtuais. Neste contexto, as informações dos acordos *Security-SLA* são utilizadas no gerenciamento da arquitetura de detecção em duas formas distintas: (i) para a ativação do serviço de detecção, o que é feito através de uma política que ativa o serviço e informa à arquitetura quais os parâmetros de desempenho serão utilizados; (ii) para especificar o conjunto de métricas responsáveis por monitorar os parâmetros de desempenho e os intervalos para a coleta de informações. Com esta abordagem, garante-se que que todos os detalhes do monitoramento de ataques sejam claramente expressos tanto para o provedor quanto para o usuário do serviço.

Outra aplicação do sistema de detecção de anomalias é o acompanhamento do cumprimento dos acordos através da validação da qualidade de mecanismos de controles de prevenção a ataques existentes na infraestrutura da nuvem. Neste caso, as informações resultantes da detecção de anomalias são utilizadas como objetivos do acordo, limitando o número de eventos de ataques em VMs que serão aceitos pelo usuário do serviço.

6.4 Conclusão

Neste capítulo foi apresentado um estudo sobre a utilização de assinaturas de desempenho geradas a partir de dados de monitoramento para a detecção de anomalias de segurança em VM. Os resultados obtidos durante o estudo indicaram que é possível detectar eventos de ataques em VM mesmo não conhecendo as particularidades dos serviços em execução. Para isto, foram empregados mecanismos baseados em aprendizado de máquina e seleção de características. Por fim, o capítulo apresentou um modelo de arquitetura que utiliza o algoritmo SVM que devido a alta taxa de precisão alcançada, pode ser utilizada como alternativa a sistemas de detecção de intrusão que dependem da instalação de ferramentas na máquina virtual.

Capítulo 7

Conclusões

A segurança de serviços e máquinas virtuais em execução em nuvens computacionais é um aspecto importante para indivíduos e organizações que desejam utilizar esta tecnologia. Apesar disto, os esforços no sentido de fornecer garantias de segurança por parte do provedor do serviço ainda são insuficientes para usuários que necessitam de níveis de segurança específicos.

Embora a literatura sugira o uso de acordos de níveis de serviço de segurança para a especificação dos níveis de proteção capazes de atender as necessidades dos usuários do serviço, o uso prático de tais acordos ainda é restrito, devido a dificuldades na especificação dos níveis de segurança e principalmente pela falta de mecanismos para o acompanhamento destes acordos.

Como forma de minimizar a lacuna entre a especificação de níveis de segurança e o monitoramento destes níveis, este trabalho apresentou um sistema de monitoramento de acordos de níveis de serviço de segurança para nuvens de serviços de infraestrutura. O sistema proposto integra-se à solução de gerenciamento da infraestrutura da nuvem e passa a coletar informações necessárias para a verificação de possíveis violações do acordo. Para isto, são utilizados agentes baseados em técnicas de monitoramento que dispensam a instalação de ferramentas na máquina virtual, sob controle do usuário do serviço.

Para permitir a utilização do acordo de segurança sendo monitorado de forma automática, o sistema utiliza uma abordagem onde os acordos são expressos através de uma linguagem XML. Esta permite a representação das métricas e parâmetros usados para a avaliação do cumprimento do acordo, e políticas que podem ser utilizadas por mecanismos de controle de segurança da infraestrutura da nuvem.

Apesar do sistema proposto ter sido projetado para o acompanhamento de acordos de segurança, sua arquitetura flexível permite a utilização como um sistema de monitoramento de uso geral. Utilizando tal característica, a solução foi empregado na coleta de informações de desempenho (CPU, tráfego de rede, utilização de memória etc) que per-

mitiram a realização de um estudo sobre o uso de assinaturas de desempenho na detecção de anomalias de segurança em máquinas virtuais.

Os resultados obtidos neste estudo mostraram que é possível detectar anomalias de segurança causadas por ataques a serviços em execução em máquinas virtuais a partir de informações de desempenho. Esta indicação permitiu projetar uma arquitetura de detecção de anomalias de segurança que utiliza um seletor de parâmetros baseado em algoritmo genético e um classificador baseado em *Support Vector Machines*. Em simulações realizadas durante o estudo, a combinação destes dois componentes mostrou-se efetiva na detecção de eventos de ataques, o que fornece evidências que o uso da arquitetura de detecção proposta aliada ao sistema de monitoramento é uma alternativa a sistemas de detecção de intrusão que dependem de ferramentas instaladas na máquina virtual.

7.1 Trabalhos Futuros

O mecanismo de *hook* utilizado para integração do sistema de monitoramento com a solução de gerenciamento da infraestrutura da nuvem, embora existente em diversas soluções, não utiliza uma interface padrão entre diferentes implementações, o que implica em adaptações do sistema de monitoramento, reduzindo a sua portabilidade. Com a crescente adoção do padrão *Open Cloud Computing Interface* entre os desenvolvedores de solução de nuvem, torna-se viável a implementação de mecanismos de integração a partir deste padrão, garantindo portabilidade e interoperabilidade entre soluções de nuvem.

A resolução de lacunas semânticas, necessária para a utilização de agentes baseados em introspecção pode ser beneficiada pela integração de *frameworks* para inspeção de memória de máquinas virtuais à solução de monitoramento. Através destes *frameworks* os agentes teriam a disposição mecanismos robustos para resolução destas lacunas em diferentes sistemas operacionais.

Diferentes técnicas para coleta de dados podem ter um impacto significativo do desempenho do nó. Por este motivo um estudo sobre tais custos e alternativas para coleta de informações pode resultar em redução dos recursos do nó utilizados pela infraestrutura de monitoramento.

O estudo sobre a detecção de anomalias de segurança a partir de assinaturas de desempenho baseia-se em um conjunto restrito de serviços e técnicas de ataque. Como forma de aprimorar a abordagem proposta, novos testes devem ser conduzidos utilizando diferentes técnicas e serviços. Nestes testes, novos parâmetros podem ser considerados como forma de melhorar a precisão do processo de classificação dos eventos. Há espaço para melhorias no modelo de classificação, a partir da utilização de técnicas de classificação não supervisionadas, que podem minimizar a necessidade da geração de ataques contra a máquina virtual para a coleta de assinaturas.

7.2 Publicações

Durante a realização deste trabalho foi publicado um artigo científico em conferência internacional e outro foi aceito para publicação em conferência nacional. Em [27], aceito para publicação, são apresentadas resumidamente a solução de monitoramento e a arquitetura de detecção de anomalias de segurança abordadas nos Capítulos 5 e 6, respectivamente. Em [20] é proposta uma metodologia para geração de um índice de segurança baseado em uma arquitetura de métricas a partir de *Security-SLA*.

Apêndice A

Esquema XSD para Extensão da Linguagem WSLA

Listagem A.1: Esquema de extensão da linguagem WSLA

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3    xmlns:wsla="http://www.ibm.com/wsla" targetNamespace="http://www.ibm.com/wsla"
4    elementFormDefault="qualified">
5
6    <xsd:include schemaLocation="WSLA.xsd" />
7
8    <xsd:complexType name="PolicyType" abstract="true">
9      <xsd:attribute name="name" type="xsd:string" use="required" />
10   </xsd:complexType>
11
12   <xsd:complexType name="PolicyObjectType">
13     <xsd:complexContent>
14       <xsd:extension base="wsla:PolicyType">
15         <xsd:sequence>
16           <xsd:element name="ObjectContainer" type="xsd:anyType" />
17         </xsd:sequence>
18       </xsd:extension>
19     </xsd:complexContent>
20   </xsd:complexType>
21
22   <xsd:complexType name="PolicyTextualType">
23     <xsd:complexContent>
24       <xsd:extension base="wsla:PolicyType">
25         <xsd:sequence>
26           <xsd:element name="Container" type="xsd:string" />
27         </xsd:sequence>
28         <xsd:attribute name="format" type="xsd:string" use="required" />
29       </xsd:extension>
30     </xsd:complexContent>
31   </xsd:complexType>
32
33   <xsd:complexType name="SecurityObjective">
34     <xsd:complexContent>
```

```
35        <xsd:extension base="wsla:ServiceLevelObjectiveType">
36          <xsd:sequence>
37            <xsd:element name="Policy" type="xsd:string" />
38            <xsd:element name="Expression" type="wsla:LogicExpressionType"
39              minOccurs="0" maxOccurs="unbounded" />
40          </xsd:sequence>
41        </xsd:extension>
42      </xsd:complexContent>
43    </xsd:complexType>
44
45    <xsd:complexType name="SecurityServiceDefinitionType">
46      <xsd:complexContent>
47        <xsd:extension base="wsla:ServiceDefinitionType">
48          <xsd:sequence>
49            <xsd:element name="Policy" type="wsla:PolicyType" />
50          </xsd:sequence>
51        </xsd:extension>
52      </xsd:complexContent>
53    </xsd:complexType>
54
55  </xsd:schema>
```

Apêndice B

Exemplo de um Acordo *Security-SLA*

A Listagem B.1 apresenta um exemplo de utilização da linguagem WSLA para representação de acordos de segurança a partir dos níveis de serviço descritos na Tabela B.1.

Tabela B.1: Parâmetros do acordo *Security-SLA*

Categoria	**Descrição**	**Nível**
Manutenção	Prazo para correção de vulnerabilidades de S.O.	3 dias
	Intervalo para confirmação de vulnerabilidade de aplicação	3 horas
	Prazo para correção de vulnerabilidades de aplicação	6 dias
	Atualização do banco de dados de vírus/-malware	12 horas
Proteção contra código malicioso	Varredura de documentos durante no envio	1 (true)
	Intervalo para varredura em lote	1 dia
	Percentual de spams por dia	$\leq 1\%$
Gerenciamento de contas	Tamanho da senha	≥ 8 caracteres
	Intervalo entre mudança de senhas	6 meses
	Prazo entre cancelamento da conta e exclusão	1 hora

Listagem B.1: Exemplo de um acordo *Security-SLA*

```
<?xml version='1.0' encoding='UTF-8'?>
<SLA name="SLA:9" xmlns="http://www.ibm.com/wsla"
  xmlns:xacml="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/wsla SecWSLASchema.xsd
                      urn:oasis:names:tc:xacml:2.0:policy:schema:os
                      access_control-xacml-2.0-policy-schema-os.xsd ">
  <Parties>
    <ServiceProvider name="The Provider" />
    <ServiceConsumer name="CUSTOMER:1" />
  </Parties>
  <ServiceDefinition name="SaaS-Application"
    xsi:type="SecurityServiceDefinitionType">
    <Schedule name="MainSchedule">
      <Period>
        <Start>2013-01-01T00:00:00-03:00</Start>
        <End>2013-12-31T23:59:59-03:00</End>
      </Period>
      <Interval>
        <Minutes>5</Minutes>
        <Seconds>0</Seconds>
      </Interval>
    </Schedule>
    <SLAParameter name="paramBatchScanInterval" type="double"
      unit="seconds">
      <Metric>metricBatchScanInterval</Metric>
    </SLAParameter>
    <SLAParameter name="paramPassowordSize" type="integer"
      unit="characters">
      <Metric>metricPasswordSize</Metric>
    </SLAParameter>
    <SLAParameter name="paramInvalidLogins" type="integer"
      unit="logins">
      <Metric>metricInvalidLogins</Metric>
    </SLAParameter>
    <Metric name="metricPasswordSize" type="double" unit="seconds">
      <Source>The Provider</Source>
      <MeasurementDirective xsi:type="Gauge">
        <ReadingSchedule>MainSchedule</ReadingSchedule>
        <MeasurementURI />
      </MeasurementDirective>
    </Metric>
    <Metric name="metricBatchScanInterval" type="double" unit="seconds">
      <Source>The Provider</Source>
      <MeasurementDirective xsi:type="Gauge">
        <ReadingSchedule>MainSchedule</ReadingSchedule>
        <MeasurementURI />
      </MeasurementDirective>
    </Metric>
    <Metric name="metricSpamRatio" type="double" unit="%">
      <Source>The Provider</Source>
      <Function resultType="double" xsi:type="Divide">
        <Operand>
          <Metric>metricSpamPerDay</Metric>
        </Operand>
        <Operand>
          <Metric>metricMessagesPerDay</Metric>
```

```
58          </Operand>
59        </Function>
60      </Metric>
61      <Metric name="metricMessagesPerDay" type="double" unit="messages">
62        <Source>The Provider</Source>
63        <Function resultType="double" xsi:type="Sum">
64          <Function resultType="TS" xsi:type="TSConstructor">
65            <Schedule>MainSchedule</Schedule>
66            <Metric>metricMessagesLastHour</Metric>
67            <Window>24</Window>
68          </Function>
69        </Function>
70      </Metric>
71      <Metric name="metricSpamPerDay" type="double" unit="messages">
72        <Source>The Provider</Source>
73        <Function resultType="double" xsi:type="Sum">
74          <Function resultType="TS" xsi:type="TSConstructor">
75            <Schedule>MainSchedule</Schedule>
76            <Metric>metricSpamLastHour</Metric>
77            <Window>24</Window>
78          </Function>
79        </Function>
80      </Metric>
81      <Metric name="metricSpamLastHour" type="double" unit="spams">
82        <Source>The Provider</Source>
83        <MeasurementDirective xsi:type="Gauge">
84          <ReadingSchedule>MainSchedule</ReadingSchedule>
85          <MeasurementURI />
86        </MeasurementDirective>
87      </Metric>
88      <Metric name="metricMessagesLastHour" type="double" unit="messages">
89        <Source>The Provider</Source>
90        <MeasurementDirective xsi:type="Gauge">
91          <ReadingSchedule>MainSchedule</ReadingSchedule>
92          <MeasurementURI />
93        </MeasurementDirective>
94      </Metric>
95      <Metric name="metricInvalidLogins" type="integer" unit="logins">
96        <Source>The Provider</Source>
97        <MeasurementDirective xsi:type="Gauge">
98          <ReadingSchedule>MainSchedule</ReadingSchedule>
99          <MeasurementURI />
100       </MeasurementDirective>
101     </Metric>
102     <Policy name="LoginControl" xsi:type="PolicyObjectType">
103       <ObjectContainer xsi:type="xacml:PolicyType"
104         PolicyId="LoginControl"
105         RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
106         <xacml:Target>
107           <xacml:Resources>
108               <xacml:Resource>
109             <xacml:ResourceMatch
110               MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
111               <xacml:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">WebMailApp
112               </xacml:AttributeValue>
113               <xacml:ResourceAttributeDesignator
114                 DataType="http://www.w3.org/2001/XMLSchema#string"
115                 AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" />
```

```
116             </xacml:ResourceMatch>
117             </xacml:Resource>
118           </xacml:Resources>
119         </xacml:Target>
120         <xacml:Rule RuleId="LoginRule" Effect="Permit">
121           <xacml:Target>
122             <xacml:Actions>
123               <xacml:Action>
124                 <xacml:ActionMatch
125                   MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
126                   <xacml:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">login
127                   </xacml:AttributeValue>
128                   <xacml:ActionAttributeDesignator
129                     DataType="http://www.w3.org/2001/XMLSchema#string"
130                     AttributeId="AplicationAction" />
131                 </xacml:ActionMatch>
132               </xacml:Action>
133             </xacml:Actions>
134           </xacml:Target>
135           <xacml:Condition>
136             <xacml:Apply
137               FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
138               <xacml:EnvironmentAttributeDesignator
139                 AttributeId="UserLogin" DataType="http://www.w3.org/2001/XMLSchema#string" />
140               <xacml:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
141                 <xacml:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Joao
142                 </xacml:AttributeValue>
143                 <xacml:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Jose
144                 </xacml:AttributeValue>
145               </xacml:Apply>
146             </xacml:Apply>
147           </xacml:Condition>
148         </xacml:Rule>
149         <xacml:Rule RuleId="DenyAll" Effect="Deny" />
150       </ObjectContainer>
151     </Policy>
152   </ServiceDefinition>
153   <Obligations>
154       <ServiceLevelObjective name="sloPasswordSize">
155         <Obliged>The Provider</Obliged>
156         <Validity>
157           <Start>2013-01-01T00:00:00.000-03:00</Start>
158           <End>2013-12-31T23:59:59.000-03:00</End>
159         </Validity>
160         <Expression>
161           <Predicate xsi:type="GreaterEqual">
162             <SLAParameter>paramPasswordSize</SLAParameter>
163             <Value>8</Value>
164           </Predicate>
165         </Expression>
166         <Schedule>MainSchedule</Schedule>
167       </ServiceLevelObjective>
168       <ServiceLevelObjective name="sloPolicy1" xsi:type="SecurityObjective">
169         <Obliged>The Provider</Obliged>
170         <Validity>
171           <Start>2013-01-01T00:00:00.000-03:00</Start>
172           <End>2013-12-31T23:59:59.000-03:00</End>
173         </Validity>
```

```
174         <Expression>
175           <Predicate xsi:type="Equal">
176             <SLAParameter>paramInvalidLogins</SLAParameter>
177             <Value>0</Value>
178           </Predicate>
179         </Expression>
180         <Schedule>MainSchedule</Schedule>
181         <Policy>LoginControl</Policy>
182       </ServiceLevelObjective>
183       <ServiceLevelObjective name="sloBatchScanInterval">
184         <Obliged>The Provider</Obliged>
185         <Validity>
186           <Start>2013-01-01T00:00:00.000-03:00</Start>
187           <End>2013-12-31T23:59:59.000-03:00</End>
188         </Validity>
189         <Expression>
190           <Predicate xsi:type="LessEqual">
191             <SLAParameter>paramBatchScanInterval</SLAParameter>
192             <Value>86400</Value>
193           </Predicate>
194         </Expression>
195         <Schedule>MainSchedule</Schedule>
196       </ServiceLevelObjective>
197       <ServiceLevelObjective name="sloSpamRatio">
198         <Obliged>The Provider</Obliged>
199         <Validity>
200           <Start>2013-01-01T00:00:00.000-03:00</Start>
201           <End>2013-12-31T23:59:59.000-03:00</End>
202         </Validity>
203         <Expression>
204           <Predicate xsi:type="LessEqual">
205             <SLAParameter>paramSpamRatio1</SLAParameter>
206             <Value>0.1</Value>
207           </Predicate>
208         </Expression>
209         <Schedule>MainSchedule</Schedule>
210       </ServiceLevelObjective>
211       <ActionGuarantee name="actionBatchScanInterval">
212         <Obliged>The Provider</Obliged>
213         <Expression>
214           <Predicate xsi:type="Violation">
215             <ServiceLevelObjective>sloBatchScanInterval
216             </ServiceLevelObjective>
217           </Predicate>
218         </Expression>
219         <EvaluationEvent>NewValue</EvaluationEvent>
220         <QualifiedAction>
221           <Party>The Provider</Party>
222           <Action actionName="notification" xsi:type="Notification">
223             <NotificationType>Violation</NotificationType>
224             <CausingGuarantee>sloBatchScanInterval</CausingGuarantee>
225             <SLAParameter>paramBatchScanInterval</SLAParameter>
226           </Action>
227         </QualifiedAction>
228         <ExecutionModality>Always</ExecutionModality>
229       </ActionGuarantee>
230     </Obligations>
231   </SLA>
```

Referências Bibliográficas

[1] Khalid Alhamazani, Rajiv Ranjan, Fethi Rabhi, Lizhe Wang, and Karan Mitra. Cloud monitoring for optimizing the qos of hosted applications. In *IEEE 4th International Conference on Cloud Computing Technology and Science, 2012*, CloudCom, 2012, pages 765 –770, December 2012.

[2] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing, 2011. Disponível em `https://downloads.cloudsecurityalliance.org/initiatives/guidance/csaguide.v3.0.pdf`. Acessado em 09 de julho de 2013.

[3] Amazon Web Services, Inc. Amazon CloudWatch. Disponível em `http://aws.amazon.com/cloudwatch`. Acessado em 09 de julho de 2013.

[4] AndrAlain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web services agreement specification (WS-Agreement), March 2007. Disponível em `http://scheme.ogf.org/documents/GFD.107.pdf`. Acessado em 09 de julho de 2013.

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, and Matei Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical report, Electrical Engineering and Computer Sciences University of California at Berkeley, 2009.

[6] Alberto Avritzer, Rajanikanth Tanikella, Kiran James, Robert G Cole, and Elaine Weyuker. Monitoring for security intrusion using performance signatures. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 93–104. ACM, 2010.

[7] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. *Encyclopedia of software engineering*, 2(1994):528–532, 1994.

[8] Boldizsár Bencsáth and Miklós Aurél Rónai. Empirical analysis of denial of service attack against smtp servers. In *International Symposium on Collaborative Technologies and Systems*, CTS'07, pages 72–79, 2007.

[9] Diana Berberova and Boyan Bontchev. Design of service level agreements for software services. In *Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, CompSysTech '09, pages 13:1–13:6, New York, NY, USA, 2009. ACM.

[10] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishman, and Uri Shaft. When is "nearest neighbor"meaningful? In *In International Conference on Database Theory*, 1999.

[11] Philip Bianco, Grace A. Lewis, and Paulo Merson. Service level agreements in service-oriented architecture environments. Technical Report CMU/SEI-2008-TN-021, Carnegie Mellon University - SEI, September 2008. Disponível em `http://www.sei.cmu.edu/reports/08tn021.pdf`. Acessado em 09 de julho de 2013.

[12] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In *High Performance Computing & Simulation, 2009. HPCS'09. International Conference on*, pages 1–11. IEEE Computer Society, 2009.

[13] Sean Carlin and Kevin Curran. Cloud computing security. *International Journal of Ambient Computing and Intelligence*, 3(1):38–46, april-june 2011.

[14] Gabriel Cartier, Jean-François Cartier, and José M. Fernandez. Next-generation dos at the higher layers: A study of smtp flooding. In *Network and System Security*, pages 149–163. Springer, 2013.

[15] Valentina Casola, Antonino Mazzeo, Nicola Mazzocca, and Massimiliano Rak. A SLA evaluation methodology in service oriented architectures. In *Quality of Protection*, volume 23 of *Advances in Information Security*, pages 119–130. Springer US, 2006.

[16] Yanpei Chen, Vern Paxson, and Randy H. Katz. What's new about cloud computing security? Technical Report UCB/EECS-2010-5, Electrical Engineering and Computer Sciences University of California at Berkeley, January 2010.

[17] Elizabeth Chew, Marianne Swanson, Kevin Stine, Nadya Bartol, Anthony Brown, and Will Robinson. Performance measurement guide for information security. Technical Report SP-800-55-rev1, National Institute of Standards and Technology, July 2008. Disponível em `http://csrc.nist.gov/publications/nistpubs/800-55-Rev1/SP800-55-rev1.pdf`. Acessado em 09 de julho de 2013.

[18] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Journal of Machine Learning*, 20(3):273–297, 1995.

[19] CSA. Cloud control matrix, March 2013. Disponível em `https://cloudsecurityalliance.org/download/cloud-controls-matrix-v1-4/`. Acessado em 09 de julho de 2013.

[20] Carlos Alberto da Silva, Anderson Soares Ferreira, and Paulo Lício de Geus. A methodology for management of cloud computing using security criteria. In *Proceedings of the IEEE Latin American Conference on Cloud Computing and Communications*, LatinCloud'12, Porto Alegre, Brazil, November 2012.

[21] Shirlei Aparecida de Chaves, Rafael Brundo Uriarte, and Carlos Becker Westphall. Toward an architecture for monitoring private clouds. *IEEE Communications Magazine*, 49(12):130–137, 2011.

[22] Shirlei Aparecida de Chaves, Carlos Becker Westphall, and Flavio Rodrigo Lamin. SLA perspective in security management for cloud computing. In *Sixth International Conference on Networking and Services*, ICNS'10, pages 212–217, 2010.

[23] DMTF. Open virtualization format specification. Technical Report DSP0243, Distributed Management Task Force, December 2012. Disponível em `http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.0.0.pdf`. Acessado em 09 de julho de 2013.

[24] Vincent C. Emeakaroha, Ivona Brandic, Michael Maurer, and Schahram Dustdar. Low level metrics to high level slas - lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments. In *International Conference on High Performance Computing and Simulation, 2010*, HPCS'10, pages 48–54. IEEE Computer Society, 2010.

[25] ENISA. Cloud computing: Benefits, risks and recommendations for information security. Technical report, European Network and Information Security Agency (ENISA), November 2009. Disponível em `http://www.enisa.europa.eu/activities/risk-management/files/deliverables/cloud-computing-risk-assessment/at_download/fullReport`. Acessado em 09 de julho de 2013.

[26] Eucalyptus System, Inc. Eucalyptus. Disponível em `http://www.eucalyptus.com/eucalyptus-cloud`. Acessado em 09 de julho de 2013.

[27] Anderson Soares Ferreira and Paulo Lício de Geus. Uma arquitetura para monitoramento e detecção de anomalias de segurança para nuvens computacionais. In *XIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, SBSEG'13, Manaus - Brazil, Novembro 2013. No prelo.

[28] Massimo Ficco, Massimiliano Rak, and Beniamino Di Martino. An intrusion detection framework for supporting SLA assessment in cloud computing. In *Fourth International Conference on Computational Aspects of Social Networks*, CASoN'12, pages 244–249. IEEE Computer Society, 2012.

[29] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000.

[30] The Center for Internet Security. The cis security metrics. Technical report, Cloud Security Alliance, November 2010. Disponível em `https://benchmarks.cisecurity.org/tools2/metrics/CIS_Security_Metrics_v1.1.0.pdf`. Acessado em 09 de julho de 2013.

[31] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008*, GCE '08, pages 1–10, November 2008.

[32] Holger Fröhlich, Olivier Chapelle, and Bernhard Schölkopf. Feature selection for support vector machines by means of genetic algorithm. In *15th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '03, pages 142–148, 2003.

[33] Borko Furht and Armando J. Escalante. *Handbook of Cloud Computing*. Springer, 2010.

[34] Ganglia Project. Ganglia monitoring system. Disponível em `http://ganglia.sourceforge.net/`. Acessado em 29 de setembro de 2013.

[35] Simson L. Garfinkel. *Architects of the Information Society: Thirty-Five Years of the Laboratory for Computer Science at MIT*. The MIT Press, 1999.

[36] Tal Garfinkel and Mendel Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *The 10th Network and Distributed System Security Symposium*, NDSS'03, pages 191–206. Internet Society, 2003.

[37] Gartner, Inc. Gartner says worldwide public cloud services market to total $131 billion, February 2013. Disponível em `http://www.gartner.com/newsroom/id/2352816`. Acessado em 09 de julho de 2013.