UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ricardo Bianchin Gomes

CLOUD AID - AUXÍLIO À PREVENÇÃO DE ATAQUES DE CANAL LATERAL NA NUVEM

Santa Maria, RS
2018

Ricardo Bianchin Gomes

CLOUD AID - AUXÍLIO À PREVENÇÃO DE ATAQUES DE CANAL LATERAL NA NUVEM

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCC), da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**

Orientadora: Profª. Drª. Roseclea Duarte Medina

Santa Maria, RS

2018

Gomes, Ricardo
Cloud Aid - Auxílio à Prevenção de Ataques de Canal Lateral na Nuvem / Ricardo Gomes.- 2018.
91 p.; 30 cm

Orientadora: Roseclea Duarte Medina
Dissertação (mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Ciência da Computação, RS, 2018

1. Computação em Nuvem 2. Segurança 3. Ataque Side Channel I. Duarte Medina, Roseclea II. Título.

Sistema de geração automática de ficha catalográfica da UFSM. Dados fornecidos pelo autor(a). Sob supervisão da Direção da Divisão de Processos Técnicos da Biblioteca Central. Bibliotecária responsável Paula Schoenfeldt Patta CRB 10/1728.

Ricardo Bianchin Gomes

CLOUD AID - AUXÍLIO À PREVENÇÃO DE ATAQUES DE CANAL LATERAL NA NUVEM

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCC), da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**

Aprovado em 26 de Fevereiro de 2018:

Roseclea Duarte Medina, Drª.
(Presidente/Orientadora)

Raul Ceretta Nunes, Dr. (UFSM)

Érico M. H. do Amaral, Dr. (UNIPAMPA) - Videoconferência

Santa Maria, RS
2018

DEDICATÓRIA

Aos amigos, familhares, professores, colegas e todos aqueles para quem este trabalho possa vir a ser proveitoso.

AGRADECIMENTOS

Este trabalho representa o encerramento de mais uma etapa em minha carreira profissional. Contudo, só foi possível chegar a este momento graças a diversas pessoas e/ou instituições presentes em minha vida, as quais certamente merecem meu sincero agradecimento. Inicialmente, agradeço a Deus pela vida, pelas coisas boas que nela acontecem, e por possibilitar que desafios sejam superados.

Em seguida, agradeço aos meus pais, Walter e Celene, por proporcionarem a mim educação, estudo e todo o apoio durante estes anos que decorreram. Agradeço também por darem suporte aos vários *kilowatts* utilizados durante toda a etapa de construção e experimentação deste trabalho.

Juntamente com os pais, vêm os irmãos, cunhados e minha amiga, parceira e atualmente namorada, Ana Paula. Certamente o primeiro contato que tive com um *datacenter* (propiciado pelo Daniel, ainda na época do ensino médio) auxiliou na minha opção em iniciar os estudos na área da computação. Uma vez escolhida a área, o apoio constante e as cobranças dos pais, irmãos (Daniela, Denise e Eduardo) e cunhados foram fundamentais para o desenvolvimento dos estudos. O incentivo e a troca de experiências da pós-graduação, bem como o amor e carinho nestes anos que se passaram desde que conheci a Ana também possuem importante papel nesta caminhada.

Devo agradecer também aos demais familiares - tios, avós e tio-avós (alguns já *in memoriam*), primos, madrinha, e também aos amigos. O afeto, os laços familiares e de amizade são, sem dúvida, fator importante na vida de qualquer pessoa. A conversa, a troca de experiências e ideias com cada um de vocês, bem como os bons conselhos fornecidos, certamente me influenciaram positivamente ao longo deste tempo. Em especial, devo agradecer ao Fernando Moro, estatístico, pelo auxílio durante o desenvolvimento deste trabalho.

Aos amigos/colegas integrantes do GRECA (*aka*. grequianos) atuais e dos anos anteriores, meu agradecimento pela amizade, parceria, pelas opiniões, pela confiança, ajuda e oportunidade de crescer junto ao grupo. São diversos nomes para listar aqui, mas a cada um de vocês, minha sincera gratulação. Agradeço especialmente à minha orientadora, Prof[a]. Rose, por me acolher junto ao grupo desde 2012 e por aceitar me orientar durante esta caminhada, onde seus ensinamentos e profissionalidade e também alguns "puxões de orelha" foram de suma importância para que pudesse chegar até aqui.

Meu reconhecimento aos professores Carlos, Giliane, Andrea e Lisandra, por compartilharem o conhecimento durente esta jornada. Ao Josmar, pelo auxílio com todos os trâmites. Aos professores que aceitaram compor a banca avaliadora, prof. Raul e prof. Érico, meu reconhecimento a vocês pelo caráter, profissionalismo e pela colaboração com este trabalho. À UFSM, pela oportunidade de acesso ao ensino superior e pós-graduação.

Meu agradecimento a todos vocês, partes integrantes desta conquista!

"O começo de todas as ciências é o espanto de as coisas serem o que são"'

(ARISTÓTELES)

RESUMO

CLOUD AID - AUXÍLIO À PREVENÇÃO DE ATAQUES DE CANAL LATERAL NA NUVEM

AUTOR: RICARDO BIANCHIN GOMES
ORIENTADORA: ROSECLEA DUARTE MEDINA

O mercado de tecnologias baseadas em *cloud computing* obteve grande expansão no decorrer da década atual. Esta propagação tem ajudado a propiciar serviços que melhor se ajustam às necessidades dos usuários, principalmente em termos de capacidade computacional e custos financeiros. Ao mesmo tempo, questões de segurança ainda não resolvidas, como ataques de *side channel* (ASC), culminam em ameaça para a segurança daqueles que fazem uso destes serviços. Neste sentido, diferentes técnicas tem sido propostas para a minização deste risco, sendo que poucas exploram a análise de lançamentos de máquinas virtuais em ambientes de *Infrastructure as a Service* (IaaS). Mesmo entre as que o fazem, a análise deste comportamento de forma adaptativa se demonstra pouco explorada. Assim, este trabalho apresenta o desenvolvimento de uma metodologia de segurança integrável a controladores de ambientes IaaS através de uma API REST. A metodologia atua na análise de padrões de lançamento de máquinas virtuais, no intuito e evitar o ataque durante a elaboração de seu pré-requisito. Para tal, faz-se uso do método de máquinas de vetores de suporte para a geração de uma modelo preditor. Testes baseados no conjunto de dados Google Cluster Trace apontam uma boa qualidade do modelo gerado e a capacidade de identificação de prováveis inícios de ASC.

Palavras-chave: Computação em nuvem. Segurança. Ataque Side Channel.

ABSTRACT

CLOUD AID - HELP TO THE PREVENTION OF SIDE-CHANNEL ATTACKS IN CLOUD

AUTHOR: RICARDO BIANCHIN GOMES
ADVISOR: ROSECLEA DUARTE MEDINA

The market of cloud-based technologies has been greatly expanded in this current decade. The propagation of this business and technology model has been providing flexibility in services to its users, mainly in financial costs and computational needs. At the same time, unsolved security breaches like Side Channel Attacks (SCA) imply security threats for the ones that make use of these services. This way, several techniques has been proposed for mitigating this risk, but few of them make use of virtual machine instantiation analysis in Infrastructure as a Service. Even among those which does, the adaptive analysis of instantiation behaviour still seem barely explored. This way, this work presents the development of a security tool with possibility of integration with IaaS controllers through a REST API. This methodology actuates analysing virtual machine instantiation patterns with the objective of preventing the attack during the elaboration of its prerequisite. In order to do that, support vector machines are used to generate a predictor model. Tests using Google Cluster Trace dataset show good quality of the generated model and the viability of detecting possible indications of SCA.

Keywords: Cloud computing. Security. Side Channel Attack.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Aplication Programming Interface
ASC	Ataque de Side Channel
CC	Cloud Computing
CPU	Central Processing Unit
CSA	Cloud Security Alliance
DDoS	Distributed Denial of Service
DNS	Domain Name System
FP	Falso Positivo
IaaS	Infrastrutcture as a Service
IDS	Intrusion Detection System
IP	Internet Protocol
IPS	Intrusion Prevention System
MA	Módulo de Análise
MBD	Módulo de Banco de Dados
MC	Módulo de Conexão
MLG	Modelo Linear Generalizado
MR	Módulo de Reação
MSE	Mean Square Error
PaaS	Platform as a Service
PCS	Preditor Categórigo Simples
REST	REpresentational State Transfer
RL	Regressão Linear
RC	Redes de Computadores
RLB	Regressão Logística Binária
SaaS	Software as a Service
SMP	Symetrical Multi Processor
SVM	Support Vector Machine
UML	Unified Modeling Language
URL	Uniform Resource Locator
VM	Virtual Machine
VP	Verdadeiro Positivo

SUMÁRIO

1 INTRODUÇÃO

Cloud Computing (CC), computação em nuvem, é um modelo de serviço e um paradigma de computação que provê acesso a recursos computacionais sob demanda (GARRISON; KIM; WAKEFIELD, 2012). Citam-se como exemplos destes recursos, redes, servidores, armazenamento, serviços, etc. As nuvens estão redefinindo o mercado de Tecnologia da Informação (TI), fornecendo grande flexibilidade para locação de serviços e recursos de computação. Essa transformação consolida um modelo de negócio conhecido como "*pay-as-you-go*", ou seja, "pague apenas pelo que consumir". Segudo SUBASHINI; KAVITHA (2011), alguns dos principais benefícios encontrados em CC são resiliência, elasticidade, redução de custo e acesso ubíquo.

Os benefícios deste paradigma têm atraído diversos novos clientes nos últimos anos. Porém, sua total adoção está sendo limitada, devido a questões relativas à segurança e privacidade, sendo estes dois os principais problemas na área (MOURATIDIS et al., 2013), (KALLONIATIS et al., 2014). Assim, é possível afirmar que tratam-se de dois requisitos essenciais para a consolidação total das nuvens como plataformas robustas e confiáveis para o uso geral (GONZALEZ et al., 2012).

Muitos dos benefícios deste paradigma estão diretamente ligados ao melhor aproveitamento do hardware do fornecedor de serviços, que é obtido pelo compartilhamento de recursos. Porém, esse mesmo compartilhamento é responsável por elevar os riscos e vulnerabilidades em CC. Uma vez que o ambiente é compartilhado, atacantes podem se favorecer da própria infraestrutura de nuvem para fins indevidos. Um exemplo disso é uma *botnet* descoberta em operação utilizando o ambiente Amazon EC2 (JANSEN, 2011), reforçando a hipótese de que a *cloud* pode ser utilizada de forma maliciosa, o que impacta negativamente a segurança de seus usuários (CARON; CORNABAS, 2014).

Segundo AINAPURE; SHAH; RAO (2018), o compartilhamento de recursos acrescenta ainda novas superfícies de ataque e novos desafios de prevenção de ameaças. Além dos riscos inerentes a uma conexão com a Internet, em um ambiente de nuvem, uma máquina virtual – do Inglês, *Virtual Machine* (VM) – maliciosa pode ser alocada no mesmo servidor em que se encontra uma VM de um usuário legítimo (APECECHEA et al., 2014).

Ataques partindo de dentro da nuvem podem ter como alvo qualquer host físico ou virtual conectado à Internet, inclusive aqueles que também estão na *cloud*. O posicionamento de uma VM maliciosa próximo a uma VM legítima torna possível uma vasta gama de ataques inter-VM, com consequências que variam desde a degradação de desempenho até um grave vazamento de informações (CARON; CORNABAS, 2014). Um tipo de ataque inter-VM existente é o ataque de canal lateral – do Inglês, *Side Channel* Attack (SCA), onde uma VM maliciosa co-alocada com uma legítima possibilita que atacantes roubem dados sensíveis da memória cache do processador. Porém, mesmo se tratando de um tipo de ameaça crescente em cloud computing, os ataques de *side channel* (ASC) ainda não recebem a atenção e tratamento adequados (IQBAL et al., 2016).

Recentemente, pesquisadores como AZAB; ELTOWEISSY (2016), AINAPURE; SHAH; RAO (2018) GODFREY; ZULKERNINE (2013), GODFREY; ZULKERNINE (2014), KONG et al. (2013), LIU et al. (2015), ZHANG; ZHANG; LEE (2016) têm desenvolvido metodologias no esforço de minimizar este tipo de ameaça. Infelizmente, ainda não existe uma solução bem difundida para este problema, tornando necessária a continuidade das pesquisas neste tópico.

Com base no exposto, este trabalho apresenta o desenvolvimento de um método de prevenção de ataques de *side channel* em ambientes de *cloud computing* do tipo *infrastructure as a service*, através do uso de técnicas de classificação e predição de dados para reconhecer padrões de lançamentos de máquinas virtuais.

1.1 MOTIVAÇÃO

Embora havendo esforços recentes para que os *side channel* attacks sejam barrados em meio às *clouds*, não chegou-se a um padrão de fato capaz de solucionar este problema por completo. Conforme novos sistemas são criados para previnir ataques, novos recursos também são criados para que a segurança seja burlada. A maioria dos trabalhos de prevenção existentes fazem uso de metodologias que possuem alto custo computacional ou requerem mudanças significativas de *hardware* e/ou *software*. Soma-se a esse problema, o fato de que as ferramentas de segurança normalmente utilizadas em ambientes de CC e.g. *firewalls*, listas de controle de acesso e sistemas de prevenção de intrusão (Intrusion Detection Systems - IDS) não são capazes

de evitar esta ameaça (ANWAR et al., 2017). Contudo, segundo encontrado na literatura, existe a possibilidade de que métodos de prevenção de corresidência possam mitigar a ocorrência da ameaça exposta. Assim, espera-se que o desenvolvimento de uma ferramenta de análise de comportamento de lançamento de máquinas virtuais consiga mitigar a ocorrência de ataques de *side channel*, através da prevenção de corresidência.

1.2 OBJETIVOS

Os objetivos deste trabalho são delineados como geral e específicos, sendo apresentados a seguir.

1.2.1 Objetivo Geral

Este trabalho tem como objetivo principal, o desenvolvimento de uma metodologia de análise de comportamento de uso e prevenção de ataques de *side channel* em ambientes de computação em nuvem. Desta forma, pretende-se mitigar esta ameaça, contribuindo para a segurança desta tecnologia, como um todo.

1.2.2 Objetivos Específicos

- Realizar um Mapeamento Sistemático sobre segurança em *cloud computing*, métodos de detecção de funcionamento de ataques entre VMs, em específico, do tipo *side channel*;
- pesquisar e compreender o funcionamento das arquiteturas de detecção e prevenção já existentes;
- identificar pontos não contemplados pelas metodologias de detecção e prevenção de ASC atuais;
- definir uma metodologia alternativa para a detecção de ataques de *side channel*;
- elaborar um protótipo de ferramenta integrada a uma plataforma de *cloud computing*, implementando a metodologia previamente definida.

2 CLOUD COMPUTING

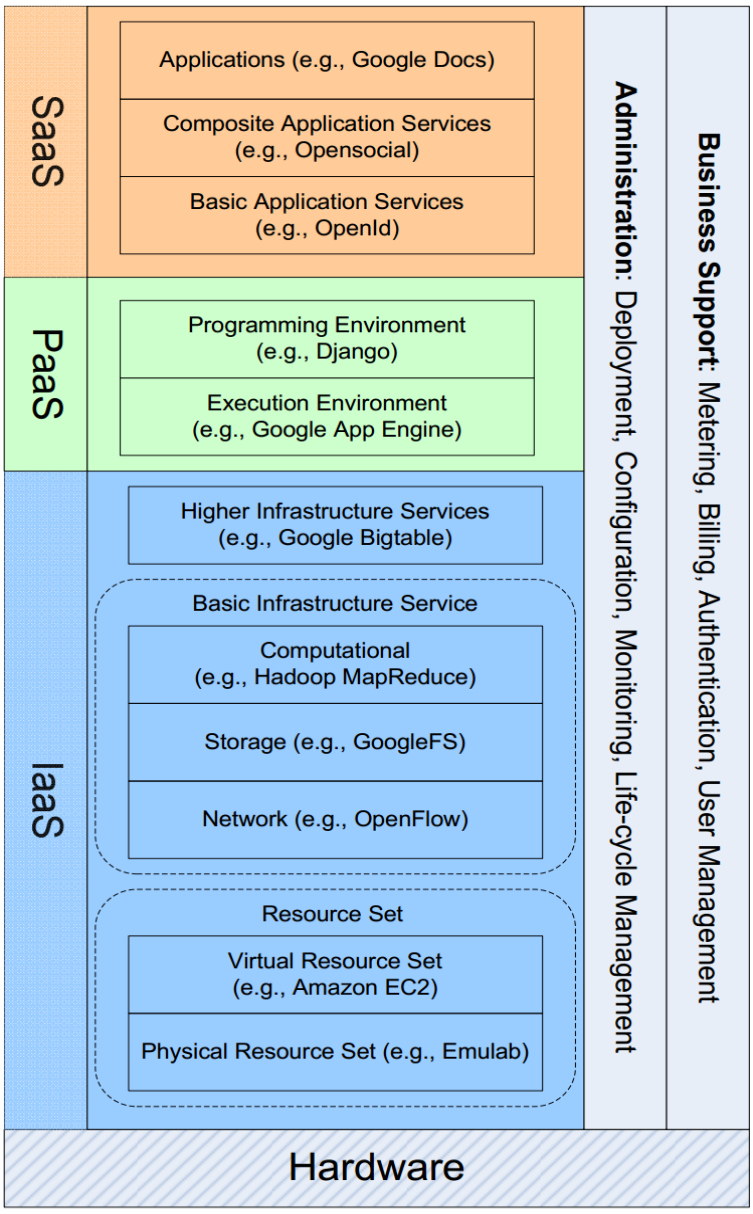
Como visto anteriormente, computação em nuvem pode ser definido como um modelo de serviço e paradigma computacional que provê acesso a recursos computacionais sob demanda. A literatura classifica as nuvens computacionais em três categorias clássicas, de acordo com o tipo de serviço oferecido (SUBASHINI; KAVITHA, 2011): Software as a Service (SaaS), Platform as a Service (PaaS) e Infrastructure as a Service (IaaS), correspondendo respectivamente a Software como Serviço, Plataforma como Serviço e Infraestrutura como Serviço. Estas três categorias compõem uma estrutura em forma de pilha (Figura 2.1) - cloud stack ou pilha da nuvem (XIAO; XIAO, 2013).

A camada superior da pilha, SaaS, consiste no fornecimento de aplicações de *software* como serviço para o cliente, sendo o Dropbox, Google Docs e Microsoft Office 365 exemplos desta modalidade. Na camada intermediária, tem-se a modalidade PaaS, onde normalmente são providas funcionalidades de desenvolvimento ao cliente, sendo exemplos típicos o Google Search API, Google App Engine e Microsoft Azure. Já na camada inferior da pilha, adjacente ao *hardware*, tem-se a modalidade IaaS, onde tipicamente são fornecidos serviços de computação, armazenamento de dados e conectividade. Exemplos desta categoria são Amazon EC2, Google FS, Microsoft Azure IaaS.

Contudo, a arquitetura das nuvens não mais se restringe apenas a estas três categorias clássicas. Com o aumento de ofertas de soluções baseadas em cloud, autores já afirmam que “tudo é um serviço” – X as a Service (XaaS). É possível contratar serviços de sensoramento, redes, telemetria, contabilização, banco de dados, mineração de dados, autenticação, saúde, telemedicina, educação, etc. (DUAN et al., 2015). Isto demonstra o grande potencial de aplicação das nuvens nas mais diversas áreas da sociedade. Esse paradigma de computação também pode ser classificado em três outras categorias, com respeito aos proprietários e usuários da infraestrutura e/ou serviços. Tratam-se de nuvens públicas, privadas ou híbridas.

A primeira modalidade, nuvens públicas, são aquelas onde a infraestrutura física é compartilhada por entre diversos usuários externos à organização que fornece o serviço de *cloud*. Tipicamente os serviços fornecidos são contratados e pagos, e

Figura 2.1: Arquitetura e categorias de serviço de CC



Fonte: XIAO; XIAO (2013)

os fornecedores são grandes empresas de TI. Exemplos desta modalidade de serviço são a Amazon EC2 e Windows Azure. Como ponto positivo, pode-se citar o fato de que normalmente os serviços ofertados possuem um preço atraente. Porém, por se tratar de um serviço público, ainda se questiona a real segurança e privacidade dos usuários neste tipo de ambiente.

Já as nuvens privadas são aquelas onde o acesso é restrito à própria empresa que é dona da infraestrutura física. Assim, não existe a locação de recursos de tercei-

ros, a própria organização é dona da arquitetura cloud. O ponto negativo que permeia esta modalidade é que o proprietário da cloud privada necessita arcar com todos os custos relacionados à sua construção e manutenção, o que envolve gastos com equipamentos e também com pessoal treinado.

Porém, apesar dos custos com equipamentos, nuvens privadas possuem aspectos positivos, os quais podem torna-las vantajosas para as corporações. O primeiro e mais notório é a possibilidade de virtualização e centralização das estações de trabalho dos usuários de uma organização através da nuvem. Assim, obtém-se melhor disponibilidade dos serviços, melhor aproveitamento de recursos computacionais e até mesmo, economia de energia (SHIKIDA et al., 2012). O segundo deve-se ao fato de que neste modelo, existe ainda a expectativa de maior privacidade dos dados e maior controle sobre os recursos, comparando-se com o modelo público, mesmo que estes recursos sejam compartilhados entre os usuários da companhia.

Por fim, existem as nuvens híbridas, que combinam características de ambas as modalidades privadas e públicas. Nesta categoria, costuma-se dispor de uma arquitetura pertencente a uma organização, a qual faz uso de sua própria infraestrutura cloud, mas uma parte dos recursos são disponibilizados a terceiros.

2.1 DESAFIOS DE SEGURANÇA EM CLOUD COMPUTING

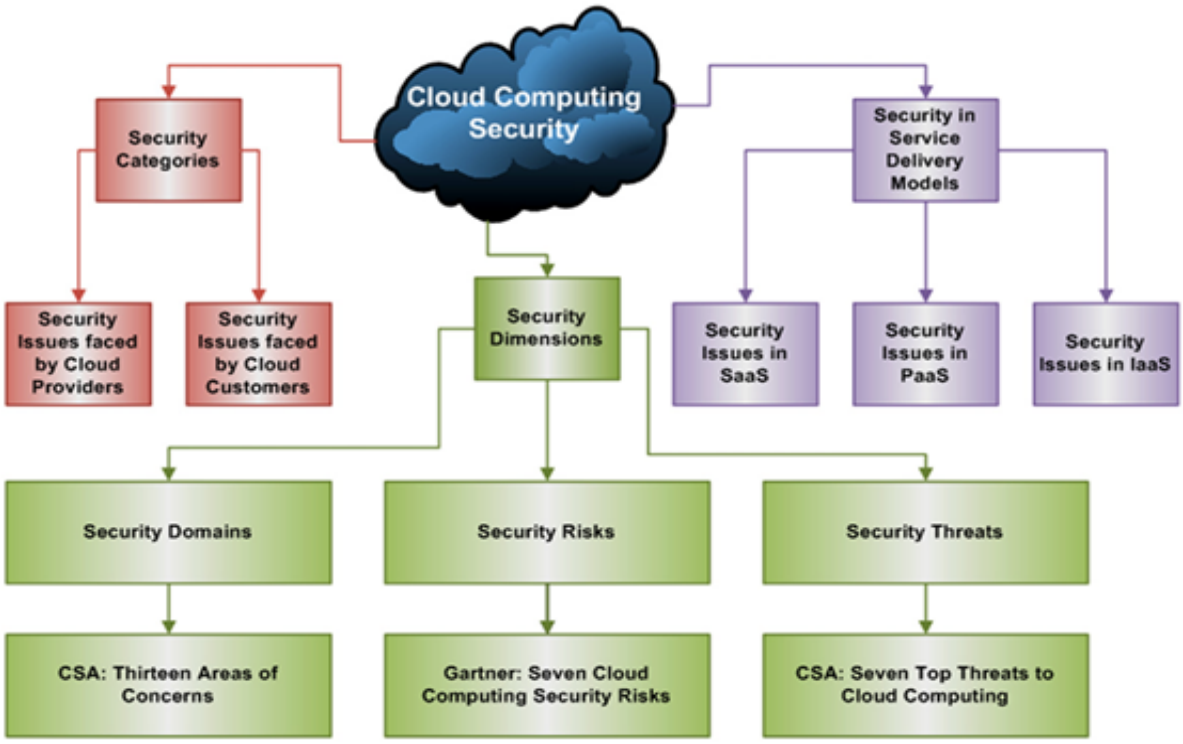
Nuvens são ambientes complexos de computação, compostos por diversas camadas de tecnologias e serviços. Desta forma, existem muitas superfícies de ataques que podem ser exploradas por agentes mal intencionados. Exemplos destas superfícies são as redes, *hypervisors*, hardware e as próprias aplicações (COPPOLINO et al., 2017).

KHAN (2016) estabelece uma classificação dos tipos de ataque em quatro categorias, baseando-se em termos de rede, máquinas virtuais, armazenamento e aplicações. Coppolino et al. (2016) estabelece uma classificação dos possíveis agentes maliciosos em três categorias: atacantes externos à *cloud*, atacantes internos à *cloud* e o próprio provedor de serviços.

KHORSHED; ALI; WASIMI (2012) propõem a classificação de falhas e ameaças de segurança em CC (Figura 2.2) de três maneiras: baseando-se em categorias de segurança, em dimensões de segurança e em modelos de fornecimento de ser-

viço. As categorias de segurança são duas: problemas de segurança enfrentados por provedores de serviço e os enfrentados pelos clientes. Já a classificação baseada em dimensões de segurança subdivide-se em três subcategorias: domínios de segurança, riscos de segurança e ameaças de segurança. Os itens compreendidos em tais subcategorias são detalhados a seguir:

Figura 2.2: Categorias de segurança em CC



Fonte: KHORSHED; ALI; WASIMI (2012)

- **Domínios de segurança:** engloba as áreas de desafios e riscos em CC, estabelecidos pela *Cloud Security Alliance* (CSA) no documento "*Security Guidance for Critical Areas of Focus in Cloud Computing*" (Cloud Security Alliance, 2017). Em sua versão 4.0, publicada em 2017, possui 14 quatorze domínios: 1) Conceitos e Arquiteturas de *Cloud Computing*; 2) Governança e Gerenciamento de Riscos Empresariais; 3) Dificuldades Legais e Descobertas Eletrônicas; 4) Gerenciamento de Auditoria e Conformidade; 5) Governança de Informações; 6) Plano de Gerenciamento e Continuidade Empresarial; 7) Segurança de Infraestrutura; 8)

Virtualização e Contêineres, 9) Resposta a Incidentes; 10) Segurança de Aplicações; 11) Segurança e Criptografia de Dados; 12) Gerenciamento de Identidade, Títulos e Acesso; 13) Segurança como Serviço e 14) Tecnologias relacionadas.

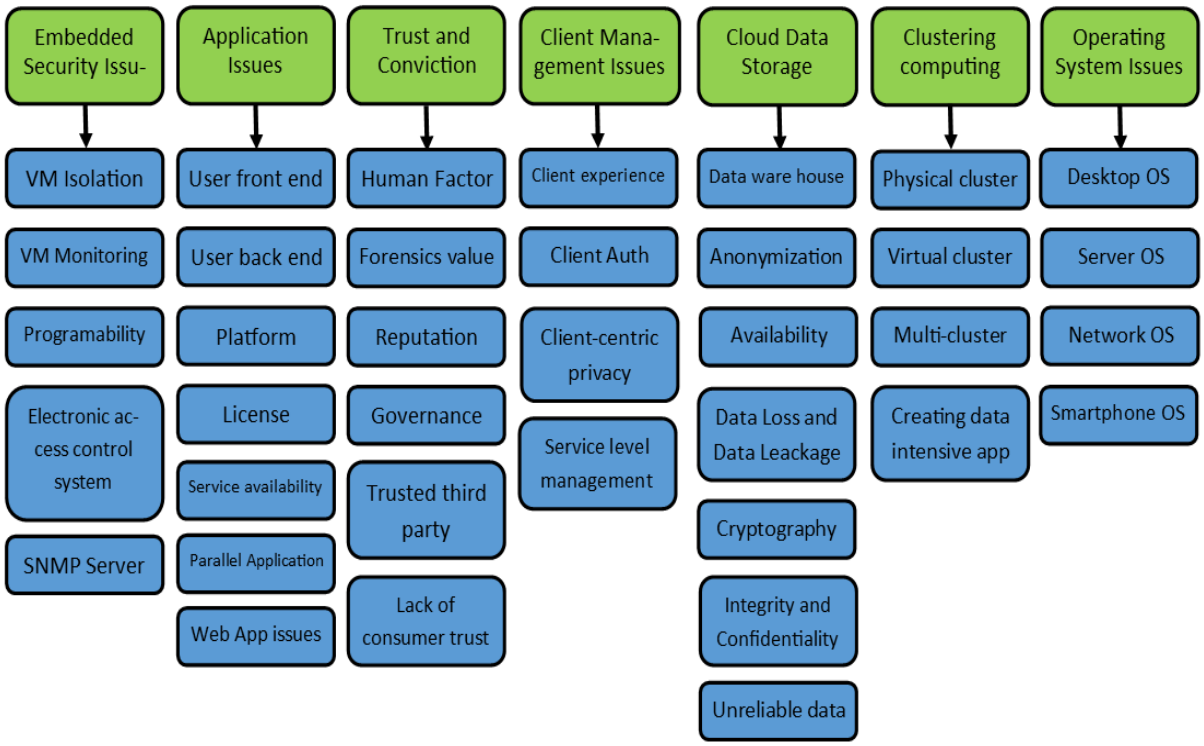
- **Riscos de segurança:** exemplos são perda da proteção dos dados, comprometimento da rede de gerenciamento da *cloud*, perda de governança.

- **Ameaças de segurança:** segundo o autor, engloba a classificação de ameaças estipulada pela *Cloud Security Alliance*. Segundo o guia *The treacherous 12 – Cloud Computing Top Threats in 2016* (Cloud Security Alliance, 2016) são elas: 1) Vazamento de dados; 2) Mau gerenciamento de identidade, credenciais e acesso; 3) APIs inseguras; 4) Vulnerabilidade de aplicações; 5) Sequestro de contas; 6) Funcionários maliciosos; 7) Ameaças persistentes; 8) Perda de dados; 9) Medidas e ações insuficientes; 10) Abuso ou uso prejudicial dos serviços; 11) Negação de serviço; e por fim, 12) Problemas de tecnologias compartilhadas.

SINGH; JEONG; PARK (2016) classificam problemas de segurança das *clouds* em sete grandes categorias (Figura2.3): problemas de segurança inerentes, problemas de aplicação, confiança e convicção, problemas de gerenciamento de clientes, armazenamento de dados na *cloud*, computação distribuída e problemas relacionados ao Sistema Operacional (SO). Segundo o autor, os problemas relacionados ao SO vão desde o SO do usuário até aquele que está rodando em uma máquina virtual dentro da nuvem, onde destacam-se riscos como trojans, *stack buffer overflow, GNU Bash Common Vulnerability*, dentre outros. Naturalmente, também estão incluídos nesta categoria de risco os sistemas operacionais dos equipamentos que se encontram no meio desta cadeia, e.g. SO de roteadores, *firewalls*, *switches*. Nestes últimos, destacam-se como fragilidades a eventual permanência das configurações padrão, e dispositivos com SO/protocolos não atualizados, podendo levar a ataques como DDOS.

Já com relação a computação paralela/distribuída, sabe-se de sua utilidade na solução de tarefas grandes, como mineração de dados e *big data*. Porém, um *cluster* vulnerável dentro da *cloud* pode ser atraente para atacantes no sentido de usar seu poder computacional para tarefas maliciosas (SINGH; JEONG; PARK, 2016). Desta forma, o cluster comprometido poderá ser usado para quebra massiva de senhas e

Figura 2.3: Classificação de problemas de segurança em CC.



Fonte: SINGH; JEONG; PARK (2016)

comprometimento de dados. Outra vulnerabidade também existente é a de utilização do cluster para ataques de força bruta.

No que tange a categoria de armazenamento de dados, existe a preocupação em garantir a disponibilidade, consistência e integridade dos dados armazenados, bem como o anonimato das informações pessoais dos cliente. Segundo o autor, existem ataques capazes de desfazer o anonimato dos dados pessoais armazenados, enquanto DDoS podem impactar seriamente na disponibilidade do serviço de armazenamento. Ataques como *Man in the Middle* e sequestro de sessões ativas podem comprometer a integridade e confidenciabilidade dos dados. *Storages* com vulnerabilidades podem ser injetados com códigos maliciosos e passar a fazer parte de *botnets*.

A área de gerenciamento de clientes enfrenta preocupações em cumprir os SLA (*Service Level Agreement*) e manter a satisfação dos contratantes dos serviços *cloud*. Também existem desafios em garantir correta autenticação, segurança e privacidade desses usuários ao acessarem os serviços através dos mais diversos dispositivos, como PCs ou *smartphones*. A área de confiança e convicção pode também impactar

na satisfação dos clientes. Nesta categoria, encontram-se desafios como o fator humano dentro de um ambiente CC, reputação, governança da *cloud*, e confiança em empresas parceiras. De fato, o ser humano pode ser um ponto vulnerável dentro do ecossistema em questão, visto que pessoas com os mais diversos níveis de acesso (funcionários em geral ou clientes) estão passíveis de cometer erros de configuração ou sofrer com golpes como o de engenharia social (SINGH; JEONG; PARK, 2016). Outra preocupação são funcionários maliciosos (KHAN, 2016).

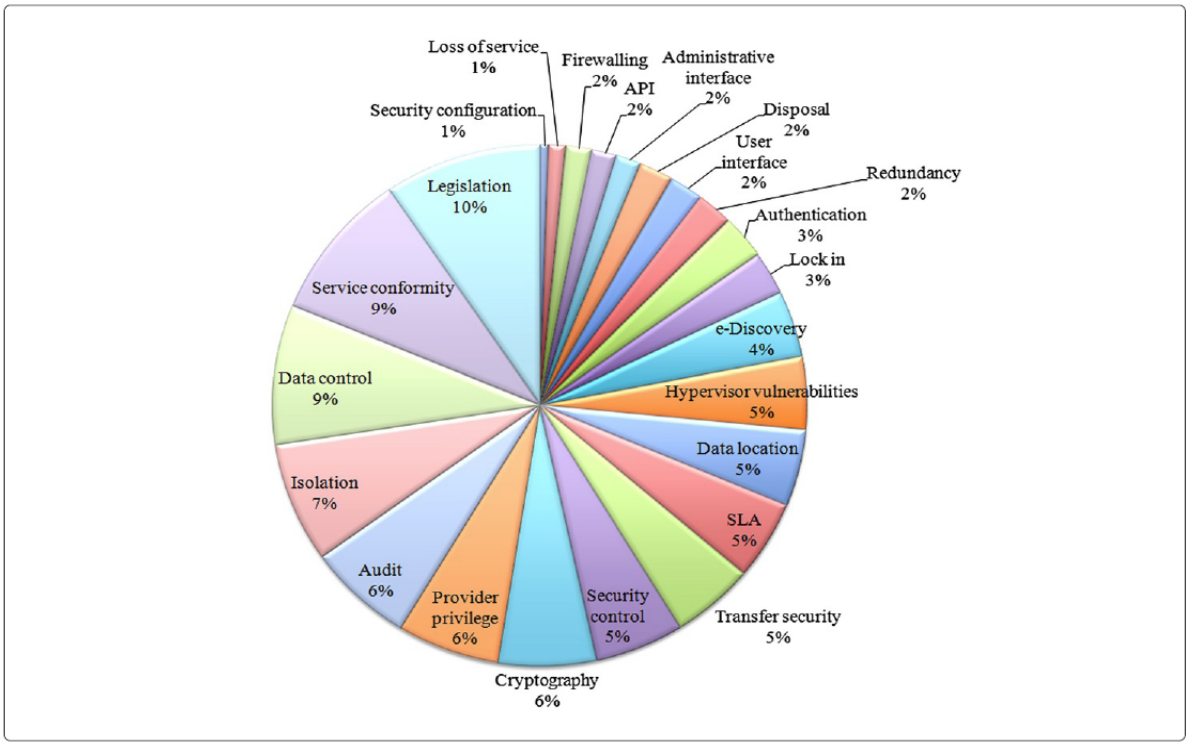
Por fim, existem as falhas em aplicações e as vulnerabilidades inerentes da arquitetura *cloud* em si. Ataques clássicos como injeção de *scripts* malignos em aplicações *web* ou injeção de SQL são possíveis de realizados, caso as medidas adequadas para desenvolvimento seguro não sejam tomadas. Servidores que implementam protocolos como DNS, SMTP e HTTP estão suscetíveis a envenenamento de cache DNS e *sniffing*, dentre outras ações maliciosas. Outro ponto em especial é a dificuldade de isolamento entre máquinas virtuais, o que pode levar a vazamentos de dados e ataques entre máquinas virtuais vizinhas do mesmo provedor (ataques inter-VM).

XIAO; XIAO (2013) realizaram um estudo sistemático sobre as vulnerabilidades de privacidade e segurança em CC, agrupando-as em cinco grande grupos: Confidencialidade, Integridade, Disponibilidade, Controle e, por fim, Preservação de Privacidade. Um estudo similar também foi realizado por (GONZALEZ et al., 2012), onde as principais ameaças encontradas podem ser visualisadas através da figura 2.4.

A vulnerabilidade que este trabalho aborda, ataques de *side channel*, pode ser classificado como um ataque baseado em hardware de computação (COPPOLINO et al., 2017), problema de confidenciabilidade (XIAO; XIAO, 2013), ameaça de vazamento de dados (KHORSHED; ALI; WASIMI, 2012) (SINGH; JEONG; PARK, 2016) e ataque entre VM (KHORSHED; ALI; WASIMI, 2012).

A seção atual visou sumarizar os principais conceitos na área de segurança em *cloud computing* (CC), os quais se relacionam com o corrente estudo. Verifica-se a grande quantidade de subáreas de segurança envolvidas em um ambiente CC, devido ao fato de que esta consiste de um agregado de diversas tecnologias. Dentre as fragilidades de segurança identificadas através deste estudo, destacam-se os ataques de *side channel* devido aos fatores: a)são relativamente fáceis de serem conduzidos (ANWAR et al., 2017); b) oferecem risco à segurança e privacidade dos usuários; e

Figura 2.4: Sumário de problemas de segurança em CC.



Fonte: GONZALEZ et al. (2012)

c) ainda não recebem a devida atenção (IQBAL et al., 2016). As seções seguintes se destinam ao aprofundamento do estudo desta vulnerabilidade e sua importância dentro de ambientes CC.

2.2 ATAQUES DE SIDE CHANNEL EM CLOUD COMPUTING

Para compreender o que são Ataques de *Side Channel* (ASC), inicialmente é necessário compreender o que são *side channels* – canais laterais. Um *side channel* é uma forma de comunicação através de um meio que não foi concebido para o propósito de comunicação de dados (GODFREY; ZULKERNINE, 2013). A partir daí, define-se que ASC são ataques que utilizam canais laterais para obter dados críticos. Normalmente, os alvos são algoritmos de criptografia, e o objetivo é roubar chaves criptográficas, o que pode levar a um comprometimento de informações sigilosas. Segundo ANWAR et al. (2017), a literatura mostra prevalência de casos de obtenção de chaves AES, havendo outros casos de sucesso envolvendo RSA, DES, ElGamal e

outros tipos de informações.

ASC são baseados em fenômenos de *hardware*, sendo classificados de acordo com tipo de canal que é utilizado para sua execução. AINAPURE; SHAH; RAO (2018) os classificam em cinco grupos, sendo eles:

1. **Ataques Baseados em Falha:** baseiam-se em falhas de computação, geradas através de mudanças de tensões dos circuitos, ou alteração no *clock*.

2. **Ataques Baseados em Análise Energética:** analisam o consumo de energia do *hardware* que executa operações criptográficas durante processamento.

3. **Ataques Eletromagnéticos:** observam as emissões eletromagnéticas dos dispositivos eletrônicos para estabelecer uma relação causal entre computação e dados.

4. **Ataques Acústicos:** exploram sons produzidos pelos computadores durante a computação de dados. Segundo o mesmo autor, a relação entre som e processamento já foi provada por (SHAMIR; TROMER, 2004).

5. **Ataques Baseados em *Cache*:** exploram o comportamento da memória *cache* de um dispositivo do computador, como CPU ou disco rígido, para inferir dados neles armazenados.

Dentre estas categorias, a mais frequentemente reportada é a de ASC baseados em cache. Embora em teoria, qualquer tipo de cache possa ser utilizado e.g., cache do disco rígido, o mais eficaz e recorrentemente utilizado é o cache de processador (KHORSHED; ALI; WASIMI, 2012).

É importante notar que esta vulnerabilidade não ocorre exclusivamente em ambientes de *cloud*. Na verdade, ataques deste tipo já haviam sido demonstrados inicialmente na década de 80, onde os computadores normalmente eram compartilhados. Com a individualização dos equipamentos, sua ocorrência foi minimizada, já que o acesso ao hardware foi dificultado. Porém, o que ocorre em CC é que a sua natureza favorece a ocorrência deste infortuno. Isso se deve ao fato de que com recursos compartilhados, o acesso aos recursos físicos torna-se mais fácil (CHANG; RAMACHANDRAN, 2016).

O primeiro ASC na nuvem foi demonstrado em 2009 por Ristenpart et al. Naquela ocasião, o autor obteve sucesso em resgatar uma mensagem de uma VM em uma outra, dentro do ambiente Amazon EC2, fazendo uso de um canal lateral. Com isso, foi definida a metodologia "*Prime + Trigger + Probe*". Outra metotologia recentemente desenvolvida para a execução de SCA é a *Flush + Reload* (YAROM; FALKNER, 2014a), a qual funciona de forma semelhante à previamente citada.

ZHANG et al. (2012) demonstrou a realização de um ASC em uma arquitetura de processador *multi-core* SMP executando o *hypervisor* Xen. IRAZOQUI et al. (2014) demonstrou a viabilidade de se realizar ASC nos viatualizadores Xen, VMware e KVM, usando a técnica de correlação de Berntein, recuperando chaves AES. Em 2015, o mesmo autor desenvolveu uma técnica capaz de determinar se uma biblioteca criptográfica específica está sendo executada em uma VM hospedada com KVM, aumentando as chances de sucesso dos ataques. GULMEZOGLU et al. (2016) demonstrou os métodos acima citados em *hypervisors* VMWare e Xen, além de reiterar suas possibilidades de execução na Amazon EC2. Outro caso de sucesso de vazamento de dados na EC2 é demonstrado por INCI et al. (2015).

Os ASC baseados em *cache* podem ainda ser classificados em três categorias, de acordo com a forma da qual a arquitetura da memória é utilizada (AINAPURE; SHAH; RAO, 2018), (RECALCATTI, 2015):

1. **Ataques Dirigido pelos Acesso**: neste tipo de ataque, o atacante obtém informações do *cache* da vítima acessando o *cache* compartilhado. Para isso, primeiramente o *cache* é preenchido com informações do atacante. Em seguida, espera-se um tempo pré-determinado e após, testa-se novamente as posições de *cache* em busca de quais dados foram substituídos e quais foram mantidos. Dados que permaneceram em *cache* normalmente são considerados parte integrante da chave criptográfica.

2. **Ataques Dirigidos por Traços**: o atacante mantém um rastro das atividades de cache da vítima. Envolve conhecimento profundo do *hardware* envolvido, e normalmente faz uso das técnicas de ASC baseado em análise de energia.

3. **Ataques Dirigidos pelo tempo**: neste tipo de ataque, o número de faltas de acesso aos dados no *cache* e, logo, o tempo de execução de códigos criptográ-

ficos revelam os dados. O tempo de execução depende do valor da variável da chave. Envolve muita análise estatística para inferir os dados.

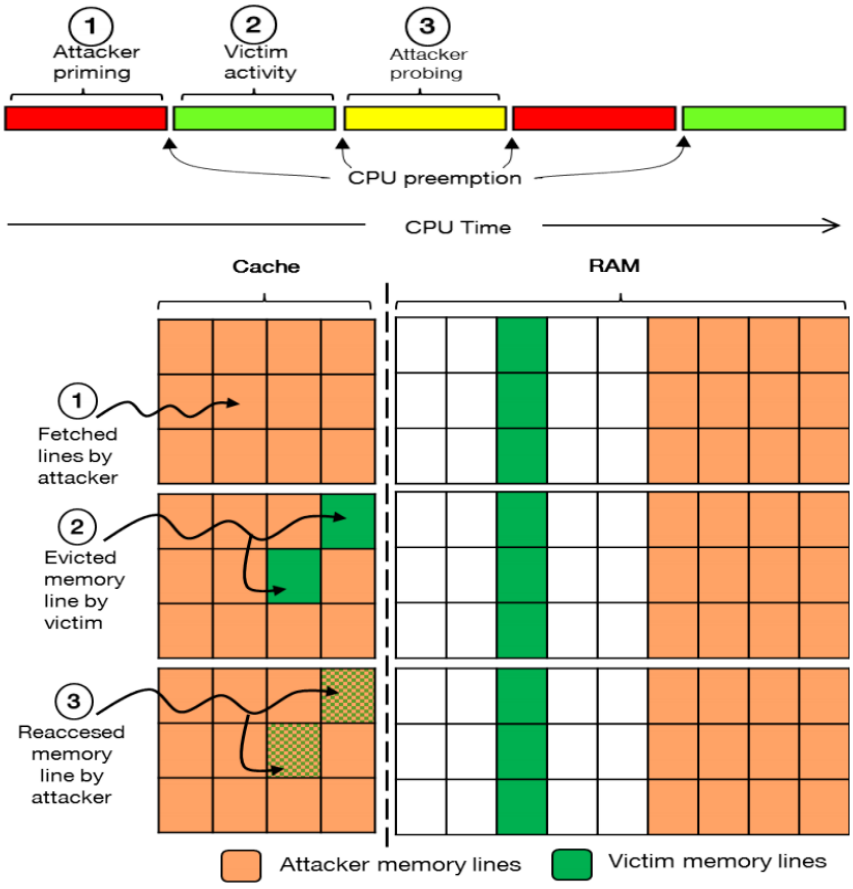
Em se tratando de ambientes de CC, os ASC possuem características semelhantes tanto aos ataques dirigidos pelo acesso quanto aos dirigidos pelo tempo, conforme afirmado por ZHANG et al. (2015). A principal metodologia, PTP (figura 2.5), se dá da seguinte forma:

1. A etapa *Prime* consiste em encher o *cache* com dados do atacante, assim como nos ataques dirigido pelo acesso. Em seguida, a CPU é liberada para que possa executar as aplicações da vítima.

2. Durante o *Trigger* (também conhecido como *Set*), a vítima armazena seus dados de memória no *cache* da CPU, substituindo parte dos dados inseridos pelo atacante.

3. Por fim, na etapa *Probe*, bits são recuperados pelo atacante através da mensuração do tempo de acesso às linhas de *cache*, de forma semelhante aos ataques dirigidos pelo tempo.

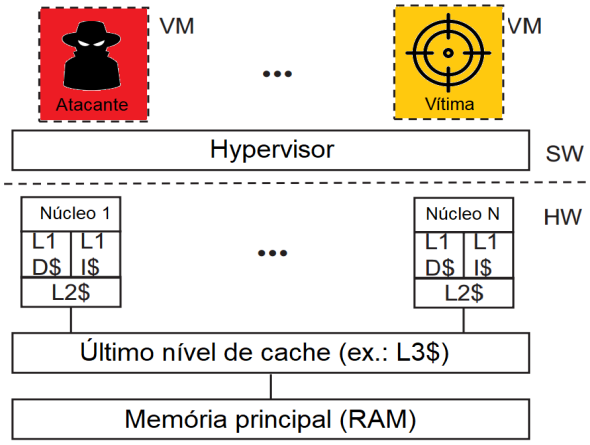
Figura 2.5: Sequência da metodologia PTP.



Fonte: BAZM et al. (2017)

Um ponto em comum entre todos os tipos de ataque de canal lateral é que o atacante precisa obter uma forma de acesso ao mesmo *hardware* em que o serviço da vítima está executando. Em termos de *cloud computing*, isto se traduz em conseguir posicionar a máquina virtual maliciosa no mesmo servidor em que se encontra a VM da vítima (Figura 2.6). Para isto, comumente ocorre o lançamento de diversas máquinas virtuais e uso de algumas ferramentas de teste de redes, como *ping* e *traceroute*, dentre outras (SECLUDIT, 2013).

Figura 2.6: Cenário de Ataque em CC



Fonte: Adaptado de LIU et al. (2015)

3 CONTRAMEDIDAS PARA SEGURANÇA EM CLOUD COMPUTING

Diversas vulnerabilidades apresentadas até o momento possuem contramedidas capazes de mitigar seus riscos associados. Tomando-se por exemplo, a categoria de vulnerabilidade "Sistemas Operacionais", apresentada na figura 2.3, uma medida de prevenção de ataques cabível é a de mantê-los atualizados com os últimos *patches* de segurança. Ataques de rede como DDoS podem ser mitigados através do emprego de ferramentas como IDS e IPS e/ou *firewalls* corretamente configurados. Problemas de acessos indevidos podem ser mitigados através de auditorias, emprego de ACL e limitação de tentativas de login (PITROPAKIS et al., 2013).

Interceptação de dados pode ser evitada através do uso de protocolos criptografados, confidencialidade e autenticidade podem ser aprimorados com uso de certificados digitais. Perda de governança e engenharia social podem ser mitigados através da capacitação profissional ou evitando operações com decisões humanas.

3.1 INTRUSION DETECTION SYSTEMS - IDS

Intrusion Detection Systems, ou sistemas de detecção de intrusão são sistemas que comumente operam em conjunto com *firewalls* e outros mecanismos, a fim de prover segurança a um sistema de redes de computadores (RC). IDS são categorizados de acordo com a forma em que operam, sendo que os dois principais tipos são IDS de Detecção de Anomalias e IDS Baseados em Assinatura, também conhecidos por "IDS de Detecção de Mau Uso" (EESA; ORMAN; BRIFCANI, 2015), (BAMAKAN et al., 2016). Um tipo ferramenta similar aos IDS, os IPS - Intrusion Prevention Systems, são capazes de, além de identificar um ataque, tomar contramedidas para cessá-lo.

O primeiro tipo constrói um modelo de padrão de atividades a partir de um conjunto de dados, o qual é utilizado para a classificação de novos eventos. Para isso, faz-se uso de métodos de aprendizado estatístico como *Support Vector Machines* (SVM) (máquinas de vetores de suporte), Redes Neurais (RN), Regressão Linear, dentre outros. Na ocorrência de um desvio do comportamento esperado, é disparado então um alarme. O ponto vantajoso deste tipo de IDS é que em geral, são capazes de se adaptar e detectar novos tipos de ataque de rede. Como desvantagem tem-se a ne-

cessidade de executar um treinamento do modelo anteriormente à sua aplicação. Um aprofundamento a respeito de métodos de aprendizado e predição são apresentados no decorrer da seção 4.

Já o segundo tipo, baseado em assinaturas, funciona comparando dados com padrões de ataques bem definidos, previamente armazenados no sistema. Este tipo de IDS não requer um treinamento inicial e ataques bem conhecidos possuem alta taxa de detecção (verdadeiro-positivo). O ponto negativo consiste na dificuldade deste método em se adaptar a novos padrões de ataques.

Diversos tipos de IDS podem ser empregados em diferentes níveis da pilha da *cloud*. Embora possuam importância no cenário de redes de computadores - e portanto, em ambientes de CC, IDS em geral são incapazes de detectar ASC (ANWAR et al., 2017). Segundo MISHRA et al. (2017), apenas IDS baseados em *VM introspection* ou *Hypervisor introspection* são capazes de detectar ASC, havendo apenas uma implementação bem-sucedida. Contudo, ela traz consigo novos riscos ao ambiente onde atua, como ataques *rootkits*.

3.2 FORMAS DE PREVENÇÃO DE ASC EM NUVENS

Ataques de *side channel* exploram o comportamento e fenômenos de *hardware* e obtém vantagem do próprio modelo de computação em nuvem, onde recursos são compartilhados entre usuários. Tal característica faz com que sua prevenção seja uma tarefa complexa. Sua detecção também costuma ser uma tarefa difícil, uma vez que trata-se de um ataque não-intrusivo (RECALCATTI, 2015), ou seja, nenhuma barreira de segurança é realmente quebrada. Mecanismos como *firewall*, criptografia e controle de acesso não conseguem barrar esta ameaça (ANWAR et al., 2017).

Através do estudo da literatura é possível identificar algumas técnicas que tem sido propostas para prevenção dessas ameaças. De acordo com ZHANG; ZHANG; LEE (2016), estas propostas são baseadas em:

- **Isolamento de *cache*:** consiste em fazer com que o *cache* seja dividido em setores isolados entre si, de forma que a comunicação entre eles não seja possível. Para isso, novas arquiteturas de *hardware* ou metodologias de *software* foram propostas. Normalmente, possui grande sobrecarga e consequente perda

de desempenho.

- **Randomização**: consiste em tornar aleatórias as medidas obtidas pelos processos espiões do atacante, fazendo com que a análise estatística se torne mais difícil. Podem ser usadas técnicas de mapeamento aleatório dos endereços de *cache*, temporizadores, dentre outras.

- **Evitar co-alocação:** busca evitar co-alocação de VM de usuários distintos no mesmo servidor. Uma forma de tornar isso viável é através da oferta de servidores exclusivos aos clientes de serviços *cloud*. Outra forma de mitigar o problema de co-alocação é através da migração frequente de VM e definição de novas políticas de alocação de VM.

Já GODFREY; ZULKERNINE (2014) classificam as iniciativas de mitigação em cinco categorias (Tabela 3.1). Como é possível de ser observado, tanto metodologias baseadas em ofuscação de correçação de dados quanto as de atraso de informações temporais dos algoritmos de criptografia implicam alterações nos códigos-fonte das aplicações ou bibliotecas criptográficas. Soluções baseadas em customização de *hardware* implicam em mudanças nas arquiteturas atuais de CPU e outros componentes. Também verifica-se que as abordagens de normalização dos estados de cache ou sua desativação, embora não impliquem alterações em *hardware* ou *software*, acarretariam grande sobrecarga (*overhead*) para os sistemas de computação, tornando-as inviáveis.

Tabela 3.1: Metodologias de Mitigação e Custo

Tipo de Alteração / Custo	**Código**	**Hardware**	**Overhead**
Tipo de Abordagem			
Ofuscação de Correlação de Dados do Cache	S[1]	N	N
Atrasar Informações Temporais	S	N	N
Normalizar o Estado do Cache	N	N	S
Customizar o Hardware	N	S	N
Desabilitar o Cache	N	N	S

Fonte: Adaptado de GODFREY; ZULKERNINE (2014)

Outra técnica de mitigação identificada no decorrer desta pesquisa é a de análise de comportamento de máquinas virtuais, nos trabalhos de SECLUDIT (2013) e

ALJAHDALI et al. (2014). Nesta técnica, costumam-se avaliar dados de lançamentos de máquinas virtuais dentro dos *clusters* da nuvem, com o intuito de identificar comportamentos suspeitos. Parâmetros tipicamente analisados são número de VM lançadas e tempo (SECLUDIT, 2013), e número de instâncias encerradas pelos usuários (ALJAHDALI et al., 2014). A ocorrência de um comportamento anômalo dentro do conjunto de observações indica um forte indício de que um determinado usuário busca o estágio de estabelecimento de corresidência, fundamental para o sucesso de ASC.

Com a identificação de tal ocorrência, o administrador da *cloud* pode adotar medidas cautelares como encerramento de instâncias ou até mesmo o bloqueio da conta do usuário, a fim de maximizar a segurança do ambiente. Vale ressaltar que a medida adotada pelo administrador é algo crítico e nenhuma das metodologias citadas determina qual ação deve ser tomada ao se identificar comportamento anômalo, depositando a responsabilidade no administrador da *cloud*.

4 ALGORITMOS DE CLASSIFICAÇÃO E PREDIÇÃO

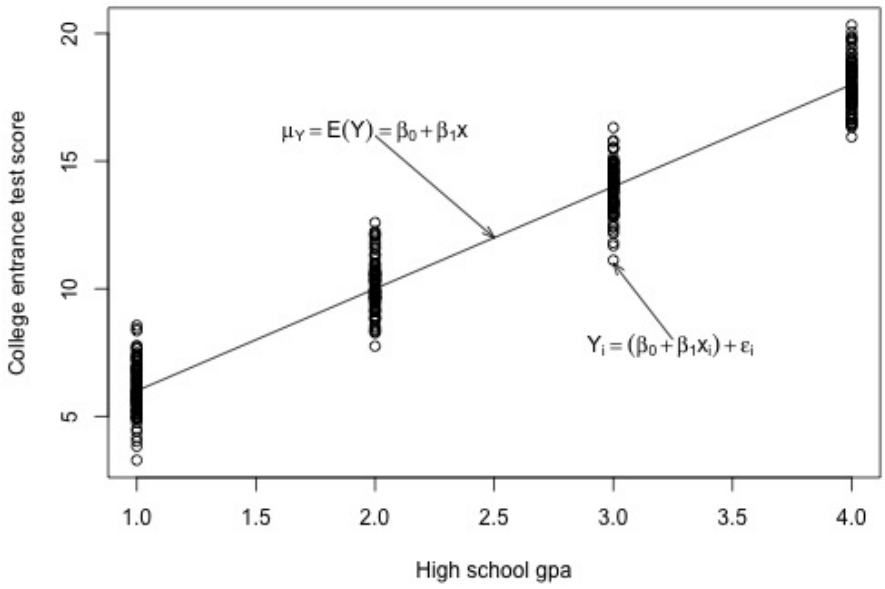
Algoritmos de predição e classificação possuem grande importância nas áreas de conhecimento que fazem uso da estatística. Tais algoritmos são aplicáveis quando se deseja classificar observações entre categorias, de acordo com variáveis explanatórias, ou quando existe a necessidade de predizer a probabilidade de um acontecimento pertencer a uma determinada categoria, dada uma *função preditora* (Eberly College of Science - The Pennsylvanya State University, 2018). Ou seja: a função básica de um algoritmo *classificador* é a de *classificar* (agrupar) observações em grupos distintos. Já um algoritmo *preditor* visa determinar se uma nova ocorrência pertence a um das classes conhecidas.

Segundo RUSSEL E NORVIG (2010), o objetivo geral destes métodos é gerar uma função classificadora/preditora h (também conhecida como hipótese) que aproxima uma função-verdade f, baseando-se em um conjunto de observações

$$(x_1, y_1), (x_2, y_2), ...(x_N, y_N) \tag{4.1}$$

onde cada y_i foi gerado por uma função desconhecida $y = f(x_i)$. Esta função hipótese pode ser gerada através de técnicas como Regressão Linear (RL). Regressão linear de uma variável consiste de uma linha reta gerada da forma $y = \beta_1 x + \beta_0 + \epsilon_i$, onde β_1 e β_0 são pesos (parâmetros) a serem estabelecidos e ϵ_i são os erros entre as observações e a reta gerada. A figura 4.1 mosta um exemplo de regressão linear utilizando uma variável, onde a reta é a função preditora e os pontos são valores previamente conhecidos e a distância entre os pontos e a reta são as medidas de erro ϵ_i do modelo.

Figura 4.1: Exemplo de regressão linear de uma variável



Fonte: Eberly College of Science - The Pennsylvanya State University (2018)

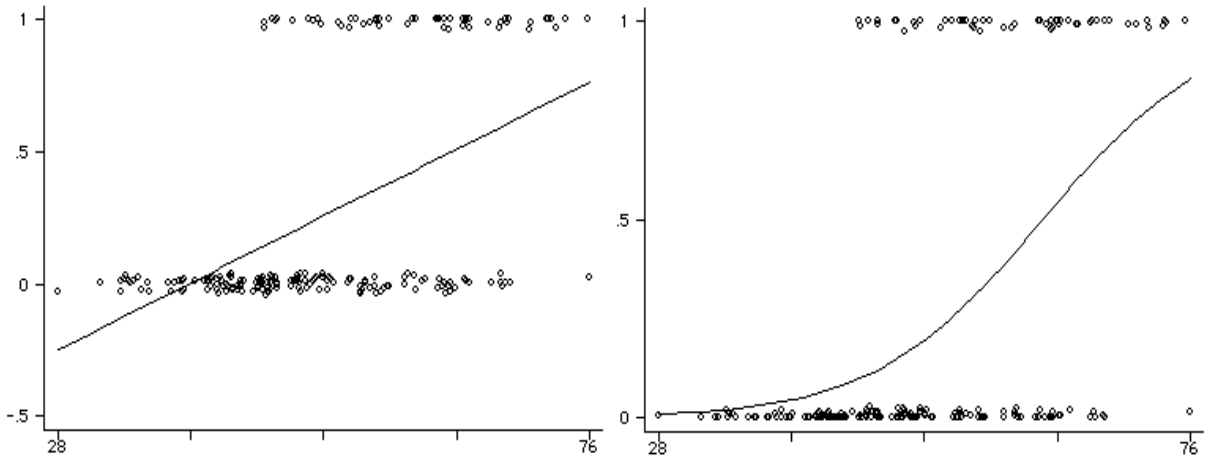
Uma variante dos modelos lineares simples são os modelos lineares generalizados (MLG), que são adequados para quando se deseja predizer um comportamento com base em mais de uma variável. Um exemplo de MLG é o modelo de Regressão Logística Binomial (RLB), o qual possui a forma geral

$$logit(\pi_i) = \log(\frac{\pi}{1-\pi}) = \beta_0 + \beta x_i + ... + \beta_0 + \beta x_k \tag{4.2}$$

onde: $log(\pi/1-\pi)$ é conhecida como função de ligação, x_i a x_k são as covariáveis β_0 e β são os pesos. A saída Y é a variável binomial aleatória, representada da forma (n, π), onde π representa a probabilidade do valor de saída ser um positivo (probabilidade de sucesso). De forma a facilitar a percepção das diferenças entre os modelos de regressão apresentados, apresenta-se a figura 4.2.

Como pode ser observado, existem dois agrupamentos de eventos em ambos os gráficos, dispostos em torno dos valores de probabilidade real 0 e 1 (eixo y). À esquerda, é representado a classificação através de uma RL simples. A reta gerada classifica os dados em zero, caso estejam abaixo dela, ou 1, caso estejam acima. Já na direita, é traçada a curva do modelo logístico. Fica visível que para o conjunto de

Figura 4.2: Diferenças entre tipos de regressão.



À esquerda: regressão linear não classifica adequadamente as observações em questão (vide grupo inferior). À direita: uma regressão logística binomial classifica com maior sucesso o mesmo conjunto. Fonte: ENDER (2018)

dados do exemplo, o modelo linear não consiste de uma boa alternativa, pois existem ocorrências próximas a 0 acima da reta. Ou seja, foram classificados erroneamente como sendo de valor 1. Por outro lado, nota-se que o modelo logístico adaptou-se melhor ao conjunto de dados em questão, com pouca ou nenhuma ocorrência de erros: todos os valores próximos a 1 estão acima da curva, enquanto que quase todos os valores próximos a 0 estão abaixo dela. Em suma, fica evidente que um modelo puramente linear não se comportará adequadamente para a classificação de dados de tendência não-linear.

Ainda segundo Eberly College of Science - The Pennsylvanya State University (2018), um preditor baseado em RLB são bons candidatos na predição variáveis binomiais categóricas Y a partir de variáveis de interesse X. Tomando-se por exemplo, o funcionamento de um IDS baseado em anomalias, a variável binomial categórica Y representa a ocorrência de um evento, podendo assumir dois valores: 0 para comportamento normal, ou 1, para comportamento suspeito, enquanto os X assumem atributos como, por exemplo largura de banda, protocolo, dentre outros.

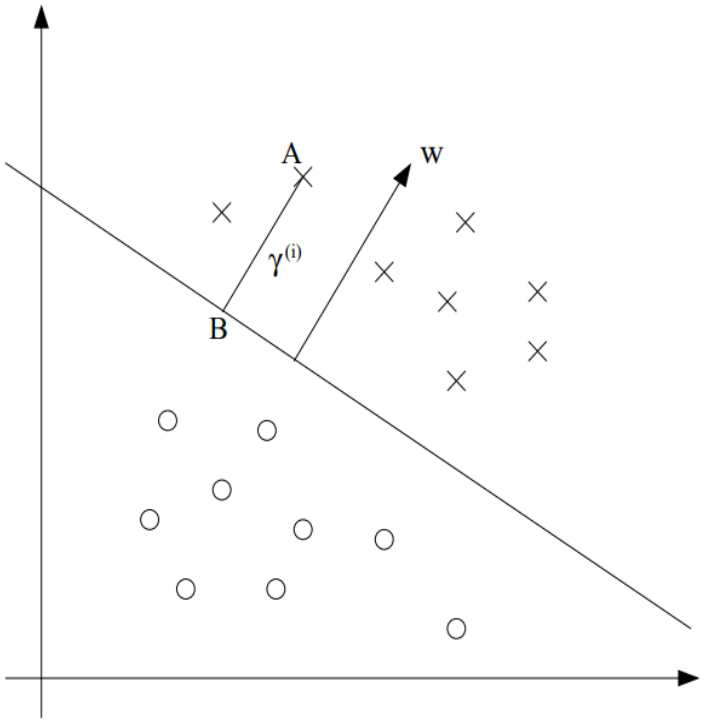
O modelo acima apresentado pode não ser uma boa opção porém, de acordo com a quantidade de *clusters* de categorias de predição. Como a saída do modelo de RLB é um valor binário (0 ou 1), o método não é adequado para predição de eventos em mais de uma classe. Nesta ocasião, existem outras técnicas de predição, como

RLB generalizada ou até mesmo, máquinas de vetores de suporte - *Support Vector Machines* (SVM).

4.1 MÁQUINAS DE VETORES DE SUPORTE

SVM são algoritmos de *predição* e aprendizado supervisionado e, segundo NG (2018), estão entre os melhores algoritmos deste tipo. Ao contrário de RLB, que classificam um conjunto de dados em duas classes, SVM são capazes de classificar dados em diversas classes, de acordo com a função *kernel* utilizada. De forma geral, SVM visam encontrar o melhor *hiperplano* que separa os conjuntos os pontos no espaço, usando para isso, vetores de suporte (figura 4.3).

Figura 4.3: Hiperplano e vetores.

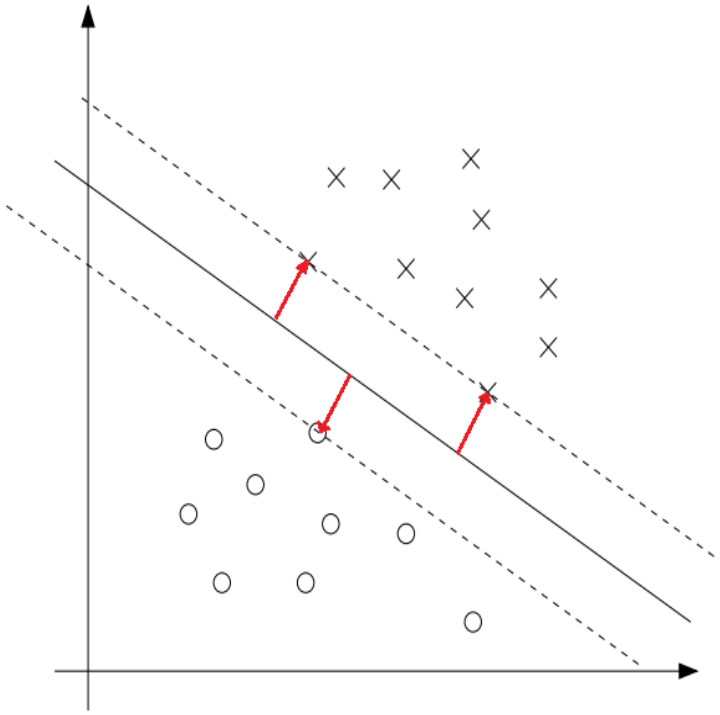


Fonte: NG (2018)

Um *hiperplano* consiste de um subspaço afim de dimenção $p-1$, sendo que p é a dimeção do espaço onde estão os dados. Por exemplo, em um espaço bidimensional, o *hiperplano* será uma reta; já em um espaço tridimencional, será um plano; para um espaço p-dimensional, será uma superfície $(p-1)$ dimensional (JAMES et al., 2013). No cenário apresentado, percebe-se um hiperplano linear e os

vetores γ^i e w, sendo o primeiro a margem funcional e o segundo, um vetor parâmetro. A estratégia para encontrar o melhor hiperplano consiste em maximizar a margem geométrica entre as observações registradas (figura 4.4).

Figura 4.4: Maximização das margens geométricas



Fonte: adaptado de NG (2018)

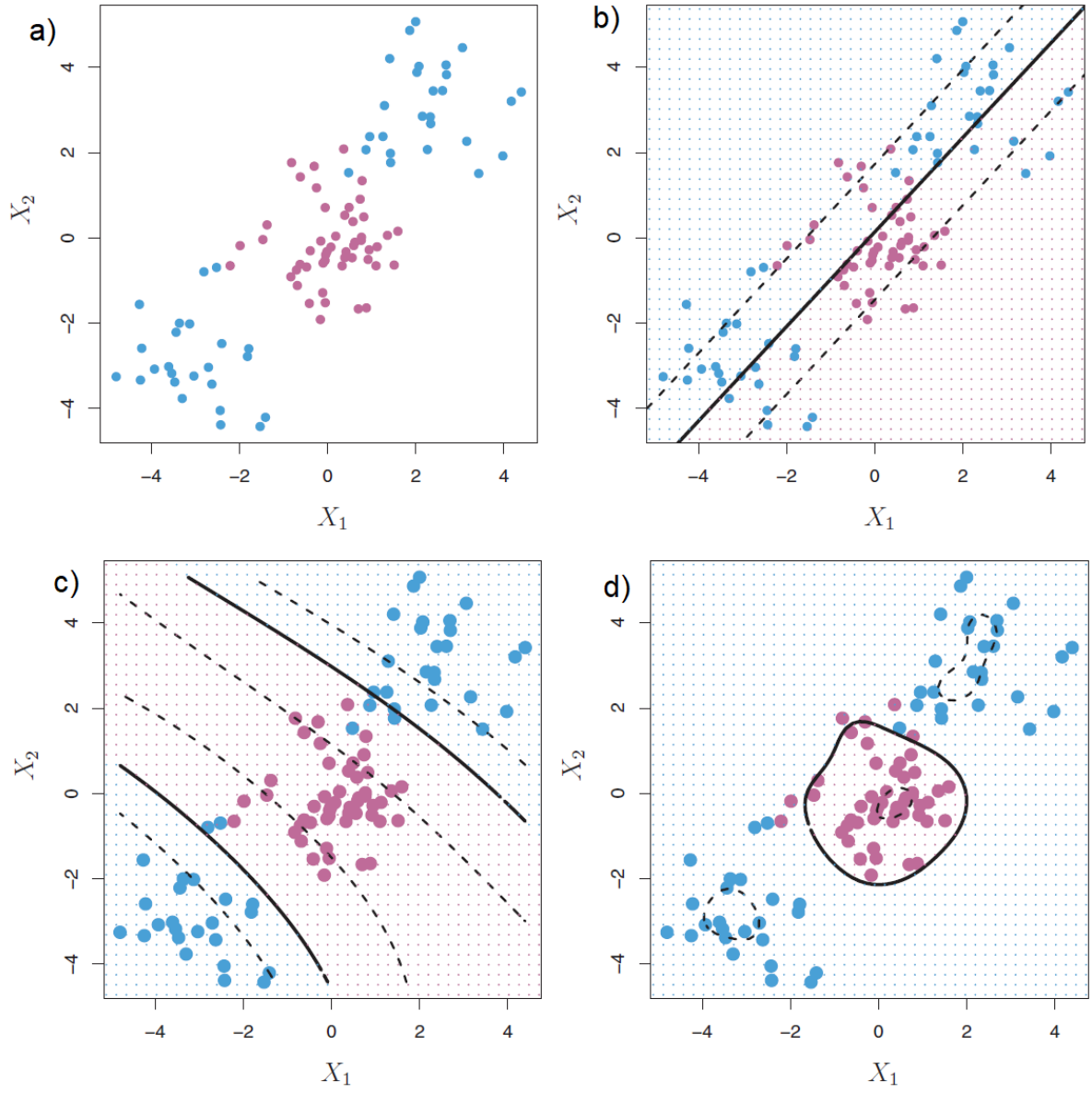
O processo de maximizar a margem geométrica, citado acima, é realizado na etapa de treinamento e é dado pelo problema de otimização *Lagrangiano*

$$\mathcal{L}(w, \alpha, b) = \frac{1}{2} \parallel w \parallel^2 - \sum_{i=1}^{m} \alpha_i [y^{(i)}(w^T x^{(i)} + b) - 1] \tag{4.3}$$

onde w, α, b são parâmetros da SVM e (x^i, y^i) são conjuntos de pontos utilizados no treinamento. Para facilitar a determinação do classificador e aprimorar a separação de conjuntos não-lineares, faz-se uso de *funções kernel* (figura 4.5), que têm o objetivo transformar os pontos para um espaço de dimensão superior, tornando sua separação possível.

É possível observar em 4.5 "a" um conjunto de dados não-lineares em duas

Figura 4.5: Maximização das margens geométricas - funções *kernel*



Em "a": exemplo de conjunto de observações. Figuras "b", "c" e "d": funções *kernel* linear, polinomial e radial, respectivamente. Fonte: adaptado de JAMES et al. (2013)

cores, azul e vermelho. O gráfico 4.5 "b" mostra uma classificação com *kernel* linear, onde fica evidente que este não é uma boa opção para o conjunto em questão. Os casos "c" e "d" representam o uso de *kernels* polinomiais e radiais (RBF), respectivamente.

Por fim, ressaltam-se as principais características das SVM (JAMES et al., 2013), sendo elas: maior robustez do modelo gerado, melhor classificação na mai-

oria dos casos, possibilidade de classificação de dados em mais de uma categoria (SVM multiclasse). Existem diversas implementações de SVM disponíveis nas mais variadas linguagens de programação, como R, Java, Python, C, etc. O Cloud Aid faz uso de SVM para a predição de comportamentos (seção 7).

4.2 ALGORITMOS PAM E K-MEANS

Diferentemente de SVM, que são usadas para classificação e predição, os algoritmos PAM e K-Means são ambos voltados apenas para a classificação de dados, o que também é conhecido como agrupamento. O algoritmos K-means visa encontrar as K médias para dividir um conjunto de dados em grupos (*clusters*) distintos e é dado pela equação (MACQUEEN et al., 1967):

$$OtimizarE = \sum_{i=1}^{k} \sum_{x \in ci} d(x, m_i) \tag{4.4}$$

onde E é a função objetivo, k é o número de *clusters* especificados pelo usuário e $d(x, m_i)$ são as distâncias euclidianas entre um ponto e o centro de um dos *clusters*. O K-means busca minimizar a distâncias entre cada ponto pertencente a cada *cluster* (KUMAR; WASAN, 2011). Por fim, os pontos são associados ao *cluster* cujo a distância entre si e o centro do cluster é a menor.

Já o algoritmo PAM busca determinar a *centroide* representante de cada *cluster* através de um processo de duas etapas (KUMAR; WASAN, 2011), (SIMOVICI, 2018):

1 Construção: conjunto de k objetos são associados para um determinado *cluster* S;

2 Troca (*swap*): o algoritmo tenta melhorar a qualidade do *cluster* trocando a seleção de objetos do cluster. Isto é, o algoritmo tenta reduzir a *dissimilaridade* entre os objetos vizinhos, encontrando o objeto mais central.

Portanto, tem-se que a diferença básica entre os algoritmos K-means e PAM é que o primeiro visa atribuir um elemento a um determinado *cluster* cujo centro está mais próximo, enquanto o PAM visa encontrar o objeto (*medoid*) mais central, de forma a representar o *cluster*.

4.3 CONSIDERAÇÕES PARCIAIS

Os capítulos deste trabalho, até o momento, apresentaram inicialmente uma sumarização dos modelos de serviços de CC que se encontram na literatura. Embora cada um dos modelos apresentados possuam suas vulnerabilidades específicas, uma delas em especial são os ataques de *side channel*, que tem sido demonstradas com sucesso particularmente em ambientes de IaaS. Desta forma, dentre as diversas apresentações deste tipo de ataque, o foco deste trabalho são os ASC entre máquinas virtuais em IaaS, baseados em *cache* do processador. Embora existindo diversas metodologias para se extrair informações sensíveis do *cache*, ao exemplo do PTP, foi retratado que existe uma etapa pré-requisito, a corresidência.

Como forma de prevenção, pode ser observado que ainda não existem medidas eficazes de barrar esta ameaça, visto que as ferramentas de segurança em RC, em geral, trabalham com base em protocolos de rede/comunicação. Firewall e IDS possuem sua devida importância dentro de RC e CC, mas, devido ao fato de seus objetos de análise serem pacotes IP, não detectam ASC (MISHRA et al., 2017).

Por fim, foram apresentados os métodos de classificação e predição estatísticos. Dentro da área de segurança de RC estes métodos são epregados, por exemplo, em IDS baseados em anomalias. Neste trabalho, tais métodos são de grande importância para a proposta desenvolvida, a qual será abordada nos próximos capítulos.

5 TRABALHOS RELACIONADOS

Através da revisão da literatura é possível identificar autores buscando contribuir com a segurança em CC, particularmente, visando mitigar ataques de canal lateral. O estudo dos trabalhos relacionados auxiliou no desenvolvimento de alguns métodos utilizados nesta dissertação. Desta forma, este capítulo tem por objetivo apresentar uma síntese do conteúdo obtido, disposto nas subseções seguintes. Ao final, é apresentada uma discussão dos métodos encontrados.

ZHANG et al. (2012), da Universidade Commonwealth da Virgínia, propôs mitigar o problema de ataques de canal lateral em *cloud computing* através do emprego de migrações temporárias de máquinas virtuais. Segundo o autor, isso tornaria mais difícil para os atacantes detectarem a localização da vítima. A proposta assume que uma chave criptográfica está dividida em um conjunto N de máquinas virtuais, abordagem que faz uso de um algoritmo conhecido como "Compartilhamento de Segredos de Shamir". Neste algoritmo, a chave criptográfica é protegida através de um polinômio, que pode ter seus pontos distribuídos.

No cenário do autor, os pontos desse polinômio ficam divididos entre as máquinas virtuais. Em seguida, o sistema calcula uma quantidade ótima *k* de máquinas virtuais para serem migradas. Por fim, essas *k* VM são periodicamente migradas de servidor, protegendo o segredo. O período de migração das VM também é calculado pelo sistema.

Já AZAB; ELTOWEISSY (2016) desenvolveram o protótipo MIGRATE, que opera de uma forma similar à metodologia apresentada na seção anterior. Trata-se de um *framework* de gerenciamento de VM e contêineres, que oferece migrações periódicas para os objetos gerenciados, entre os *hosts* físicos. O objetivo é dificultar que um usuário malicioso seja capaz de identificar a real localização de uma VM legítima, evitando a coalocação de ambos usuários no mesmo *host*. Outros trabalhos como CloudFlow (BAIG et al., 2014) e Nomad (MOON; SEKAR; REITER, 2015) também utilizam a mesma abordagem de migrações periódicas.

Os autores GODFREY; ZULKERNINE (2013) (2014) demonstraram em seus trabalhos uma abordagem de prevenção de ataques através da modificação do código-fonte do virtualizador Xen. Eles desenvolveram uma rotina de monitoramento que de-

tecta quando haverá uma troca de contexto de execução. Contexto, neste caso em específico, refere-se à propriedade das máquinas virtuais. Máquinas virtuais de usuários diferentes possuem contextos diferentes, enquanto que VM de usuários distintos possuem contextos distintos.

Caso o monitoramento detecte que haverá uma troca de contexto, ela executa um *flush* no cache do processador, ou seja, esvazia o conteúdo ali contido. Desta forma, VM co-residentes nunca conseguem inferir os dados da chave criptográfica, já que o conteúdo será sempre vazio.

Enquanto as abordagens até o momento trabalham somente a nível de *software*, KONG et al. (2013) desenvolveu uma outra abordagem para a mitigação de ataques de *side channel*, através da integração entre recursos de *hardware* e *software*. Como recursos de *hardware*, são propostos novos designs de *cache*: *Partition-Locked Cache* (PLcache) e *Random Permutation Cache* (RPcache), combinados com novas instruções do tipo *load*.

Como recursos de *software*, é sugerido o uso de um esquema de permutação para modificar a distribuição de dados dentro da memória *cache* da CPU. Segundo o autor, ambas as abordagens de *hardware* e *software* podem ser usadas isoladamente para previnir ataques, podendo também serem associadas entre si.

5.1 SECLUDIT ELASTIC DETECTOR

SecludIT é uma companhia francesa que desenvolve soluções de segurança em TI. No ano de 2015, desenvolveram um *software* chamado ElasticDetector, destinado a realizar *logs* provenientes do controlador da nuvem que está sendo monitorada. O programa busca por padrões que indicam o início de um ASC. Mais precisamente, tenta-se identificar lançamentos anormais de máquinas virtuais, o que indica a execução da fase de corresidência.

No sistema apresentado, sempre que uma máquina virtual é instanciada, é disparado um gatilho para que o evento seja analisado, sendo que as notificações são correlacionadas e armazenadas na plataforma OSSIM[2]. Se mais de nove máquinas virtuais forem instanciadas em menos de um minuto, e se as VM forem desligadas em menos de quinze minutos, o monitor as classifica como suspeitas. Vale ressaltar que

[2] https://www.alienvault.com/products/ossim

estas métricas foram definidas arbitrariamente pelo autor.

5.2 DISCUSSÃO

Uma vez estudadas as metodologias existentes nos trabalhos relacionados, é possível analisá-las em busca de suas fragilidades e dos potenciais de aprimoramento. Esta seção visa discuti-las.

Inicialmente, enquanto a migração periódica de máquinas virtuais demonstre mitigar os riscos até então apresentados, deve-se notar que migração de VM são tarefas bastante onerosas, principalmente em se tratando de recursos de rede. Ocorre também que a redistribuição de VM entre os *hosts* físicos de forma a manter o ecosistema equilibrado em termos de consumo de recursos também é uma tarefa onerosa, sendo classificado como problema NP-Difícil (XAVIER; REJIMOAN, 2016). Existem ainda ataques em cloud que exploram migrações de VM (ANWAR et al., 2017), o que abre precendentes desta prática gerar novas vulnerabilidades.

Segundo, embora a prática de esvaziar o *cache* da CPU proposta por (GODFREY; ZULKERNINE, 2014) se demonstre eficaz em impedir o acesso às informações, é necessário notar que isto tende a impactar negativamente os tempos de acesso aos dados das aplicações dos usuários legítimos. A seguir, observa-se na metodologia de (KONG et al., 2013) a necessidade de mudanças na arquiteturas de *hardware* e de sistemas operacionais que existem atualmente. Desta forma, esta solução pode não estar disponível a curto prazo.

Já a metodologia exposta apresentada por SECLUDIT (2015) se demonstra menos onerosa, porém, também existem limitações passíveis de aprimoramento. Sendo assim, demonstra bom potencial de desenvolvimento. O primeiro ponto a ser observado é que o uso de *logs* pode não ser adequado em um ambiente de tempo real, uma vez que muitos *logs* são atualizados de forma tardia. Segundo, a função classificadora não demonstra flexibilidade, já que o número de disparos de gatilho aceitáveis é constante (nove instâncias por minuto).

Sendo assim, o corrente trabalho se inspira na metodologia de prevenção expressa por SECLUDIT, onde o principal objetivo é evitar que o estágio de co-alocação seja completado. Contudo, a entrada de dados na metodologia do Cloud Aid se dá por chamadas através de uma API REST, no intuito de evitar atrasos de detecção devido

a atualizações tardias dos *logs*.

Outro diferencial entre arquitetura e as demais é que não é utilizada a técnica de migração de VM como mecanismo de defesa, uma vez que esta prática possui grande degradação de desempenho e pode levar a outras vulnerabilidades (KHAN, 2016), (SINGH; JEONG; PARK, 2016).

6 MATERIAIS E MÉTODOS

O início deste trabalho se deu através da realização de uma revisão sistemática da literatura, a fim de se obter o estado da arte na área de segurança em *cloud computing*. Para isso, seguiu-se a metodologia de pesquisa proposta por Kitchenham (2014). As questões que permeiam a pesquisa são:

- Quais são os principais desafios de segurança existentes em *cloud computing*?
- Como os ataques de *side channel* são executados em tal ambiente?
- Quais são as formas de prevenção possíveis?
- Existem formas de detecção de tais ataques?

Com base em tais questões, definiu-se que as strings de busca seria composta pelos termos "*cloud computing*", "*cloud*", "*security*", "*side channel*", "*attack*", "*prevention*", "*mitigation*"e "*detection*", incluindo operadores lógicos AND e OR. Desta forma, as strings foram definicas como:

S1 : ("cloud computing" OR cloud) AND (security OR "side channel" OR attack) OR "side channel"

S2 : ("*side channel attack*" OR "*side channel*") AND (*prevention* OR *dettection* OR *mitigation*)

Como bases de pesquisa, foram utilizadas a IEE Xplore, Elsevier, ACM, Springer e o Portal de Dissertações e Teses da CAPES. Os resultados da pesquisa foram filtrados, de forma que apenas aqueles que refletissem as questões de pesquisa permanecessem. Após a filtragem, obteve-se um total de 81 documentos.

Concluída a revisão da literatura, deu-se início à determinação do ferramental necessário para a execução do restante do trabalho. Inicialmente, estipulou-se que a solução a ser desenvolvida deveria conter uma forma de integração com um ambiente de *cloud computing*, por se tratar de uma ferramenta de prevenção de ataques naquele tipo de ambiente. Para que isso fosse possível, um ambiente de nuvem demonstrou-se necessário.

O ambiente escolhido foi uma instalação do OpenStack Newton (OPENSTACK, 2017) em um *cluster* privado, composto por máquinas heterogêneas com Sistema Operacional Debian 8 GNU/Linux, já que se tratam de uma plataforma e um SO *open-source* e bem difundidos. A opção por um ambiente próprio se deve ao fato de que os ambientes comerciais não fornecem acesso às camadas de serviço necessárias para a execução do projeto.

O *cluster* (tabela 6.1) foi composto de seis servidores, sendo composto de: 1 servidor controlador, com 2 processadores Xeon dual-core e 8GB de memória; 1 servidor de armazenamento com processador Xeon dual-core com 1GB de memória e 500GB de disco; 4 servidores de virtualização, sendo 1 Core2 Duo e 2 Core2 Quad com 4GB de memória e, por fim, 1 Xeon quad-core com 8GB de memória.

Tabela 6.1: Sumarização do *cluster* utilizado

Nó #	CPU	Memória	Rede de Controle	Rede Pública (VM)	Função na *Cloud*
1	2x Xeon Dual-Core	8GB	Sim	Sim	*Cloud Controller*
2	Xeon Dual-Core	1GB	Sim	Não	Armazenamento
3	Xeon Quad-Core	8GB	Sim	Sim	Virtualização
4	Core2 Quad	4GB	Sim	Sim	Virtualização
5	Core2 Quad	4GB	Sim	Sim	Virtualização
6	Core2 Duo	3GB	Sim	Sim	Virtualização

Fonte: autoria própria.

Os servidores foram interligados através de duas redes distintas, sendo uma rede de 1Gbps destinada para gerenciamento e controle, enquanto a outra, de 100Mbps destinada para tráfego de dados das máquinas virtuais (chamada de rede pública). Ambas as redes de controle e pública foram configuradas com endereçamento IPv4 com máscara /24.

Com o ambiente de experimentação definido e implantado, definiu-se que o protótipo seria definido em linguagem Java, na forma de *webservices* (REST), dado que trata-se de uma linguagem portátil, e o conceito de *webservices* tende a facilitar a integração com ambientes de CC. Optou-se por hospedar o protótipo em um servidor GlassFish e o banco de dados utilizado foi o MySQL.

Um ponto forte em se utilizar webservices/REST é que este protocolo permite a interação entre recursos de forma simplificada e escalável entre cliente e servidor FIELDING (2000). O protocolo utiliza URL para endereçar cada recurso disponibilizado pelos componentes sistema e fornece interfaces genéricas para a manipulação

dos valores a eles deles associados. Esta tarefa tarefa de manipulação de valores e estados é conhecida como *representação*.

Uma representação, neste contexto, contém um conjunto de dados (*bytes*), seguidos de metadados e, em alguns casos, metadados de metadados. A resposta de uma requisição REST é a representação do objeto solicitado, podendo incluir ambos dados e metadados. Ainda segundo FIELDING (2000), os benefícios associados a este protocolo são: independência entre componentes, escalabilidade de interação entre componentes, redução de latência do sistema, devido a componentes intermediários e ainda, aprimoramento de segurança.

Para fins de validação, foi escolhido o uso do *dataset* Google Cluster Trace, que consiste de um registro de 29 dias de atividade em um *datacenter* do Google. Nele, é possível encontrar informações sobre eventos de máquinas, *jobs*, tarefas, utilização de recursos, dentre outros (REISS; WILKES; HELLERSTEIN, 2014). O *dataset* escolhido foi utilizado para a geração de um modelo de detecção de comportamento através algoritmos de classificação e predição de dados.

As informações contidas no *dataset* foram filtradas de forma a se obter apenas as *features* necessárias para o sistema. Com o propósito de se realizar esta etapa, foram elaborados códigos em linguagem Java. Para classificação, foram testatos os algoritmos PAM e o K-Means, sendo o último, o utilizado neste trabalho. Para geração do modelo preditivo, foram verificados os algoritmos SVM e Regressão Logística Binomial, sendo SVM a utilizada neste trabalho. Os códigos relativos às etapas de classificação e preição supracitados foram implementadas na linguagem R[3].

Após o treinamento do modelo, utilizou-se a técnica de validação cruzada para a estimação de um novo conjunto de dados para fins de validação. O método proposto foi avaliado em termos de taxa de acerto e acurácia, apresentados e discutidos na seção 8. Por fim, os resultados obtidos foram discutidos com aqueles apresentados nos trabalhos relacionados.

[3] R Project: https://cran.r-project.org

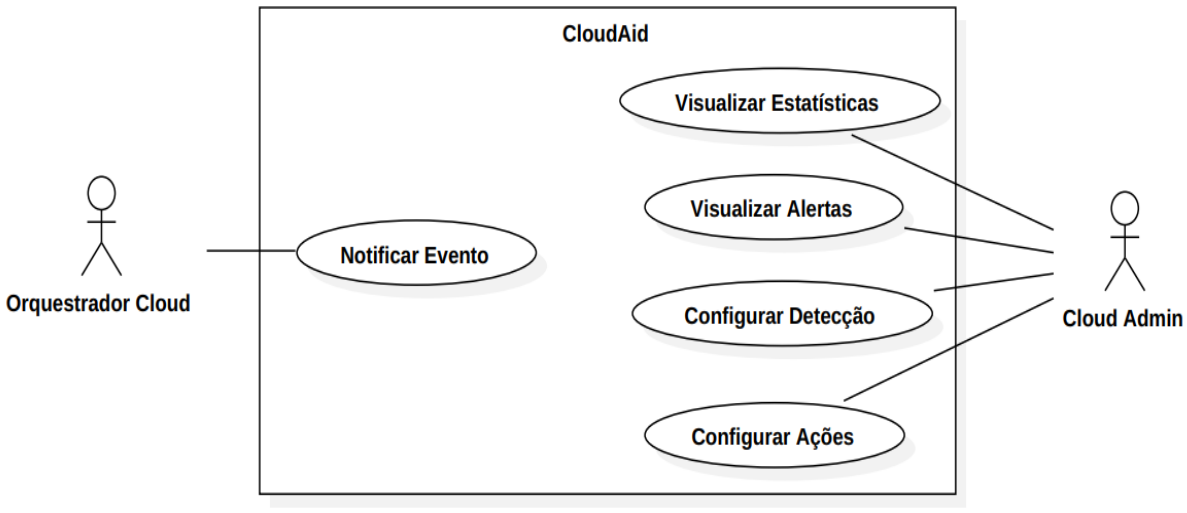
7 PROPOSTA - CLOUD AID

A possibilidade de se efetuar um ataque de *side channel* com relativa facilidade na nuvem, bem como sua dificuldade de tetecção corroboram para a necessidade do desenvolvimento de novas formas de prevenção das ameaças nos ambientes de *cloud computing*. Assim sendo, foi desenvolvido o protótipo de aplicação de auxílio à segurança denominado *Cloud Aid*. O foco de atuação desta ferramenta consiste na prevenção dos ataques de *side channel* na etapa de descoberta de co-alocação. As seções seguintes detalham o funcionamento do sistema desenvolvido.

7.1 MODELAGEM DO SISTEMA

Para o desenvolvimento do sistema, primeiramente é necessário identificar os atores envolvidos e os requisitos necessários para o funcionamento. Desta forma, é apresentada a modelagem em UML utilizada.

Figura 7.1: Casos de uso da ferramenta



Fonte: Próprio autor.

Conforme exibido pelo diagrama de casos de uso (Figura 7.1), existem dois atores envolvidos na ferramenta: o administrador do sistema (Cloud Admin) e o orquestrador da nuvem (ator não-humano). Deve ser observado que usuários dos serviços

de nuvem não interagem diretamente com a ferramenta, embora seus dados de uso sejam analisados por ela.

Os casos de uso disponíveis a um administrador da nuvem são: Vizualizar Estatísticas , Visualizar Alertas, Configurar Detecção e por fim, Configurar Ações. Eles são explicados como segue:

1. Vizualizar Estatísticas: o administrador pode visualizar informações detalhadas sobre os usuários do ambiente de nuvem;

2. Visualizar Alertas: ao serem detectados comportamentos anormais, segundo configurações do administrador, eles podem ser visualizados por meio desta função.

3. Configurar Detecção: o administrador deve ter a capacidade de ajustar parâmetros dos algoritmos de detecção, para que os resultados sejam fiéis ao máximo para os mais diversos ambientes de utilização do sistema.

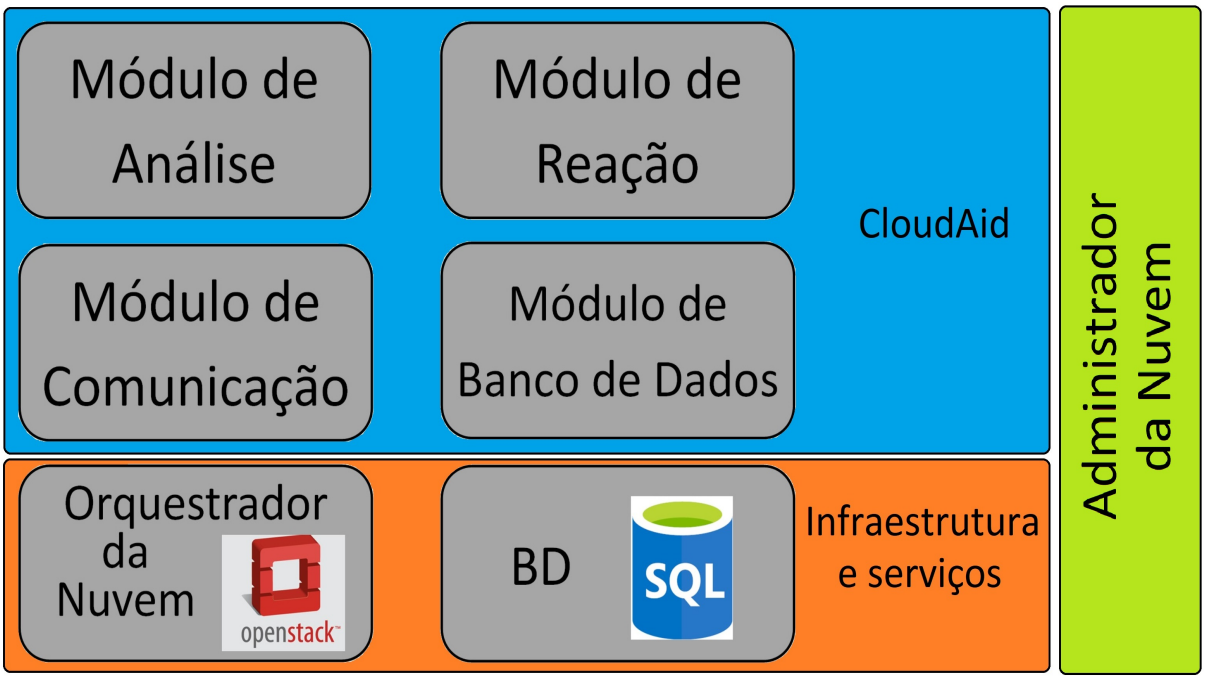
4. Configurar Ações: deve ser possível para o administrador configurar as ações a serem tomadas no caso do sistema detectar um comportamento suspeito. Este caso de uso é independente da visualização de alertas.

O orquestrador *cloud*, por sua vez, somente envia notificações para o sistema (Notificar Evento), lembrando-se que os usuários encontram-se involvidos indiretamente nesta ação.

7.2 AQUITETURA

A arquitetura do método desenvolvido é composta de quatro módulos principais, sendo eles: Módulo de Comunicação (MC), Módulo de Análise (MA), Módulo de Banco de Dados (MBD) e Módulo de Reação (MR). Estes módulos podem ser representados em camadas, como exibido na Figura 7.2.

Figura 7.2: Módulos componentes do sistema



Fonte: Próprio autor.

Ainda sobre a Figura 7.2, identifica-se que os módulos de comunicação e o módulo de banco de dados podem compõem as inferiores do sistema, enquanto os módulos de análise e o de reação consistem de camadas superiores. As seções seguintes deste capítulo se destinam à apresentação destes componentes do sistema.

7.2.1 Módulo de Comunicação

O módulo de comunicação (MC) é responsável por prover uma forma de comunicação entre o orquestrador da nuvem e a ferramenta. Isto é obtido através da definição de uma API REST, a qual é acessada pelo orquestrador. Deve ser observado que para isto, o código do orquestrador em questão deve ser modificado para fazer uso do recurso disponibilizado.

Ao receber uma notificação através da API REST, o MC repassa as informações necessárias para o módulo de análise, para que o comportamento do usuário seja inspecionado. O MC também é responsável por enviar solicitações da ferramenta para o orquestrador da nuvem, como, por exemplo, bloquear uma conta de usuário.

7.2.2 Módulo de Banco de Dados

Para armazenar as informações recebidas pelo módulo de comunicação, o sistema faz uso de um banco de dados. Nele são armazenadas informações sobre os usuários, suas instâncias e as respectivas métricas analisadas. Para isto, faz-se uso de duas tabelas: tabela de usuários e tabela de instâncias.

Na tabela de usuários, são armazenados o identificador da conta de usuário (*userID*), a quantidade de instâncias a ele associadas, a média de lançamentos e o último número de instâncias lançadas.

Na tabela de instâncias, armazenam-se o ID da instância (*instanceID*), o ID do usuário que a possui, o *uptime* médio e o *host* que está hospedando no momento atual. Para a identificação de uma instância específica, faz-se uso da tupla [*userID*, *instanceID*]. Desta forma, é possível saber qual instância pertence especificamente a qual usuário da *cloud*.

O conjunto destas informações, também conhecidas como *features*, são úteis no processo de deteção de anomalias de comportamento e foram definidas com base nos trabalhos estudados no decorrer desta pesquisa. Para garantir a consistência do banco de dados, estas informações são mantidas atualizadas pelo módulo de análise, conforme é apresentado a seguir.

7.2.3 Módulo de Análise

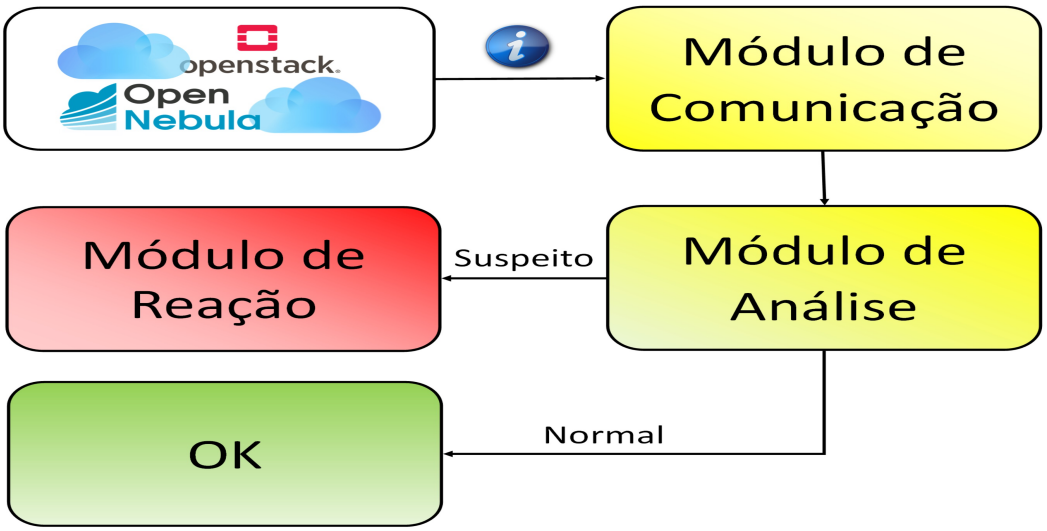
O módulo de análise possui importante papel dentro da metodologia proposta, como expresso pelas Figuras 7.2 e 7.3. Seu objetivo é a identificação da possibilidade de ataque, durante a fase de co-alocação. Para que isto seja possível, este módulo é alimentado com informações oriundas do MC e informações contidas no banco de dados e a classificação de dados ocorre de forma similar a um IDS baseado em anomalias.

O *Cloud Aid* supervisiona os dados relacionados a lançamentos de máquinas virtuais, de forma similar à ferramenta ElasticDetector (SecludIT, 2018). Porém, existem duas principais diferenças entre ambas. Primeiramente, enquanto ElasticDetector assume valores fixos de tempo e lançamentos de VM para classificação de comportamento, a ferramenta proposta faz uso de métodos estatísticos para esta função, onde

parâmetros podem ser ajustados. Segundo, a ferramenta proposta recebe os dados diretamente através de chamadas REST assim que eles são gerados. Esta abordagem tende a ser mais rápida do que a leitura de *logs* do sistema, uma vez que alguns *logs* podem ser gerados e atualizados de uma forma conhecida como *deferred update* (ELMASRI; NAVATHE, 2010), (MOHAN et al., 1992) - atualização tardia, a fim de gerar menor *overhead*. Outros aspectos positivos o uso de REST constituiem-se da maior escalabilidade da interação dos componentes do sistema, generalização de interfaces, independência de componentes e segurança

A relação entre este componente do sistema e os demais, exceto MBD, é apresentada na Figura 7.3. Como pode ser observado, notificações do sistema de *cloud* são recebidas pelo MC, o qual extrai dados relevantes e os repassa ao MA. Este, por sua vez, realiza a classificação do comportamento recente do usuário como normal ou suspeito. Em se tratando de um comportamento suspeito, é notificado o módulo de reação, para que a medida adequada seja tomada.

Figura 7.3: Diagrama de Sequência da Ferramenta



Fonte: Próprio autor.

Para o funcionamento deste módulo, conta-se com dois pincipais métodos, sendo eles: 1) cálculo da probabilidade de co-alocação na nuvem; e 2) agrupamento, classificação e predição de dados. Salienta-se que os dados de interesse utilizados nestes algoritmos são: a) número de instâncias lançadas em um determinado momento; b) total de servidores no *cluster*; c) tempo total de atividade do usuário; d)

média de lançamentos por unidade de tempo. Estas *features* são importantes para a acurácia da deteção de ataques, e encontram-se embasadas nos trabalhos de INCI et al. (2015), ZHANG et al. (2015), SECLUDIT (2013), ALJAHDALI et al. (2014).

7.2.4 Cálculo da Probabilidade de Corresidência

Neste algoritmo, deseja-se saber qual a probabilidade do usuário atacante obter corresidência em um mesmo servidor onde está localizada sua possível vítima. Nesta etapa, são utilizados os dados de lançamentos em um determinado instante de tempo e o total de servidores do *cluster* da nuvem.

Definção 7.2.1. Seja E uma experiência aleatória, representando a distribuição de I instâncias em um conjunto de S servidores. Sejam n_i o número de instâncias criadas pelo usuário em um instante de tempo e n_s o número total de servidores. Considerando que uma máquina virtual vítima está alocada em um servidor alvo $s_a \in S$, que todas as instâncias $i \in I$ são iguais e que todos os nodos do cluster possuem a mesma quantidade de recursos disponíveis. Sendo A um evento do experimento E, representando uma destas instâncias serem co-alocadas dentro do mesmo servidor s_a. Então, a probabilidade de ocorrência do evento A pode ser expressa pela equação 7.1:

$$P(A) = \frac{n_i \times (n_s - 1)!}{n_s!} \tag{7.1}$$

Ou seja, $P(A)$ é dada pela razão entre todas as formas possíveis de distribuir n_i instâncias em n_s servidores e as possibilidades de preencher cada servidor. Contudo, a equação pode ser simplificada, conforme a equação 7.2:

$$P(A) = \frac{n_i \times (n_s)!}{n_s \times (n_s - 1)!} = \frac{n_i \times n_s!}{n_s! \times n_s} \tag{7.2}$$

De onde obtem-se que (7.3):

$$P(A) = \frac{n_i}{n_h} \tag{7.3}$$

Portanto, é possível aproximar a probabilidade de um usuário atacante co-alocar uma de suas máquinas virtuais maliciosas em um mesmo servidor-alvo s_a onde está a vítima, através da razão entre a quantidade de instâncias lançadas por esse usuário e a quantidade de servidores existentes no *cluster*. Logicamente, outros fatores podem interferir na probabilidade real em cada cenário. Por exemplo a carga utilizada de cada servidor é um fator que pode interferir na corresidência, dependendo das heurísticas utilizadas pelo escalonador de instâncias do gerenciador de máquinas da nuvem.

Além disso, torna-se necessário a análise de outros parâmetros para que seja possível obter uma classificação mais precisa. Desta forma, também são analisadas métricas relativas ao tempo do período de atividade das VM e a quantidade de lançamentos. É sabido através da literatura (INCI et al., 2015), (ZHANG et al., 2015), (SECLUDIT, 2013), (ALJAHDALI et al., 2014), que estas variáveis possuem relação entre si. Desta forma, para a sua análise, faz-se necessário primeiro uma identificação do padrão de lançamentos dentre os usuários da nuvem para posterior construção de modelo preditivo.

7.2.5 Construção do Modelo Preditivo

Embora a característica da fase inicial de um SCA, corresidência, seja o lançamento frequente de diversas VM, não existe ainda um padrão bem definido desta etapa do ataque. Com base nisso, é necessário a construção de um modelo dinâmico de detecção, ou seja, com funcionamento similar a um IDS baseado em anomalias.

Conforme citado anteriormente, a primeira etapa para a construção do modelo consta na identificação de um padrão comportamental dos usuários. Para isso, o sistema faz uso do algoritmo de agrupamento K-Means. A entrada do algoritmo é a matriz de variáveis de lançamentos, contendo o tempo total de atividade, total de instâncias pertencentes ao usuário e o total de lançamentos (TL). Cada célula na matriz portanto, representa o registro das métricas de cada usuário em específico, as quais são atualizadas no banco da dados no decorrer da operação do sistema.

Em posse das classes geradas pelo algoritmo de classificação, o próximo passo adotado pelo sistema é a sua aplicação em um método de aprendizado estatístico. Mais especificamente, faz-se uso de uma máquina de vetores de suporte (vide a seção

4.1). Um ponto a ser enaltecido é que o modelo deve ser "treinado" para cada novo ambiente onde o sistema for executar, a partir de dados existentes naquele ambiente de nuvem específico. Uma vez treinado, o modelo pode enfim ser utilizado na predição de novas ocorrências entre normais ou suspeitas. A partir daí, novos eventos são preditos e utilizados para manter as variáveis TL, número de instâncias n_i e tempo t coerentes, atualizando estes registros no banco de dados.

De forma a sumarizar o funcionamento do módulo de análise, apresenta-se o algoritmo 1, o qual retrata o funcionamento do módulo em questão:

Algoritmo 1: Módulo de análise do Cloud Aid

```
F = realiza_treinamento();
Input: nh: número de hosts no cluster
Input: tr: treshold da probabilidade de coalocação
Input: A: novo evento de lançamento de VM
evento ← A;
userID ← A.userID;
n_launches ← A.nLaunches;
timestamp ← A.timestamp;
atualiza_banco();
P1 = n_launches  n_s ;
se P1 > tr então
    modulo_reacao(userID);
senão
    tl ← taxa_lancamento;
    se F(n_launches, TL, timestamp) == 1 então
        modulo_reacao(userID)
    fim
fim
```

No pseudocódigo exibido, primeiramente é realizado o treinamento da SVM responsável por predizer novos eventos. As *features* ID do usuário, número de lançamentos $n_{launches}$ informados, e *timestamp* são extraídas do evento em questão. Em seguida, atualiza-se o banco de dados, de forma a manter a consistência dos registros. Após esta etapa, calcula-se a probabilidade de corresidência $P1$, que é dada em função de $n_{launches}$ e o número de servidores n_s. Caso a probabilidade $P1$ seja superior ao gatilho configurado, invoca-se imediatamente o módulo de reação. Caso o gatilho não seja ativado, faz-se uso da função F correspondente ao modelo preditivo treinado, onde $n_{launches}$, total de lançamentos TL e o *timestamp* são parâmetros analisados.

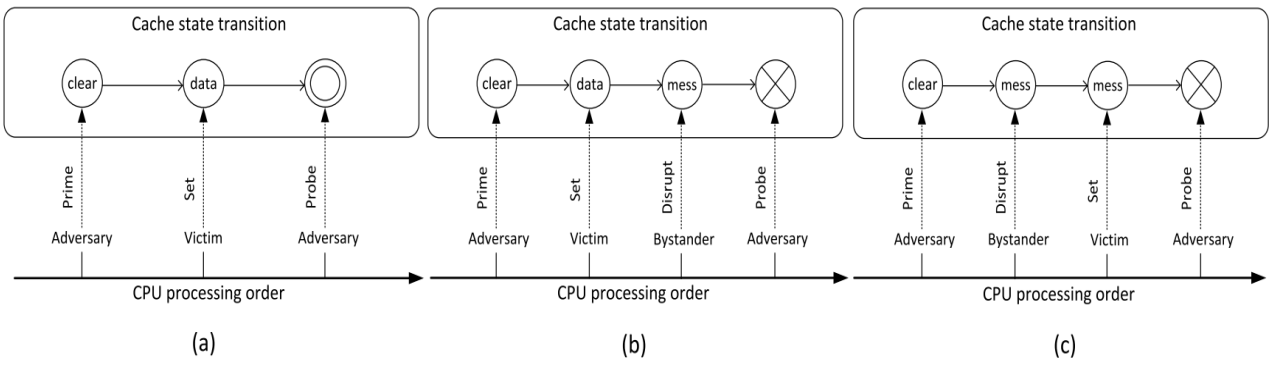
7.2.6 Módulo de Reação

Conforme citado anteriormente, uma vez que o MA classifica um comportamento como anômalo, o sistema solicita ao MR para que contramedidas sejam tomadas. Tais medidas podem incluir um alerta ao administrador ou mesmo uma ação dentro do ambiente de nuvem. As reações a serem tomadas devem ser configuradas pelo administrador do sistema. Estas podem ser meramente uma mensagem de alerta ou a intervenção no ambiente de nuvem. Através da API do controlador da nuvem, é possível que o MR acione o bloqueio de uma conta de usuário ou encerre as instâncias suspeitas, por exemplo.

Naturalmente, existem riscos associados a cada medida tomada, principalmente naquelas que envolvem a finalização das VM do usuário. Desta forma, o administrador deve levá-los em conta no momento de configurar tais ações, a fim de minimizar efeitos indesejados no ecositema dos serviços.

Uma reação em particular e ainda em desenvolvimento consiste na geração de ruído preventivamente em *hosts* com máquinas virtuais de usuários diferentes. Para isso, atualmente o sistema faz uso de uma máquina virtual alocada junto ao servidor compartilhado entre mais de um usuário. Dentro desta máquina virtual, é executado um algoritmo que cria arquivos aleatórios e faz uso de biblioteca criptográfica para cifrar os conteúdos gerados. Desta forma, produz-se interferência no processo de ataque de *side channel*.

Figura 7.4: Sequência da metodologia PTP.



Fonte: Zhang et. al(2015)

Como pode ser observado através da figura 7.4, a geração de ruído se dá no momento em que um terceiro usuário executa seu código no processador (casos "b" e "c"). Quando isso ocorre, os endereços de memória do processador são parcial ou completamente preenchidos pelos dados do terceiro, o que altera a sequência original do método PTP (caso "a"). Por fim, quando o atacante realizar a etapa *probe*, receberá uma leitura incorreta sobre quais posições de *cache* foram alteradas pelo usuário vítima, evitando ou retardando o sucesso do ataque.

7.3 CONSIDERAÇÕES DO CAPÍTULO

No decorrer deste capítulo foi abordada a modelagem da ferramenta e a arquitetura utilizada para a identificação de comportamentos suspeitos na etapa de corresidência de máquinas virtuais. Esta etapa, segundo autores como LIU et al. (2015), SECLUDIT (2013) constitui de fator necessário para o sucesso de um ataque de *side channel* (vide seção 2.1). Foi retratado que as principais características da tentativa de se obter corresidência são um grande número de lançamento de máquinas virtuais, repetidamente, em um curto intervalo de tempo.

Com base nestes fatores, foi definida uma metodologia de prevenção de ASC, através da detecção do estágio de corresidência por meios de algoritmos de classificação e predição de dados, cujos dados são alimentados pela infraestrutura de *cloud* e mantidos em uma base de dados. O uso destes algoritmos permite a geração de um modelo de classificação/predição de comportamento adaptativo ao cenário em que é empregado, diferentemente de metodologias rígidas como encontrado em SECLUDIT (2013).

Embora apresentados exemplos de possíveis medidas de reação a serem configuradas pelo administrador no caso de comportamento anômalo, sua determinação foge do escopo deste trabalho. A responsabilidade dos riscos inerentes a uma reação é, acima de tudo, do administrador do sistema e provedor de *cloud*. Contudo, ações como vigilância do usuário suspeito e a evacuação dos *hosts* onde se encontram o usuário malicioso são menos arriscadas que uma finalização de instância, podendo ser encontrados em SECLUDIT (2013), CHIAPPETTA; SAVAS; YILMAZ (2016).

Por fim, salienta-se que para fins de prova de conceito, a reação implementada pelo MR consiste de uma simples nofificação. Outra observação importante é que

para fins de validação, o MC não implementa criptografia, por levar em conta que a rede de gerenciamento da *cloud* seja privada e, portanto, segura. Sem dúvida, em se tratando de um ambiente de produção, deve-se adequar a conexão de forma a utilizar criptografia.

Enfim nesta etapa, concluem-se a apresentações dos aspectos teóricos referentes a este trabalho. Por conseguinte, o capítulo 8, a seguir, visa retratar a avaliação da metodologia proposta, com as respectivas apresentações e discussões dos resultados.

8 TESTES E AVALIAÇÃO

Este capítulo apresenta a condução dos experimentos realizados para a avaliação da ferramenta. Para tal foi desenvolvido um protótipo, o qual foi escrito em java. Optou-se por fazer uso da API Jersey para fornecer suporte a REST. O servidor de aplicação utilizado foi o Apache Tomcat e o banco de dados, MySQL. Para a validação do método, fez-se uso de um conjunto de dados disponibilizado pelo Google, denominado Google Cluster Trace. Para a construção dos modelos de classificação utilizados pela ferramenta, contou-se com o auxílio do ambiente de programação R. É mister salientar que outras bases de dados comumente utilizadas em validação de IDS, como a CAIDA 2007[4] e CTU-13[5] não são adequadas para a avaliação da metodologia exposta. Isso se deve ao fato de que essas bases são constituídas de traços de tráfego IP coletados em grandes redes, sendo que o objeto de análise da metodologia exposta são os padrões de lançamentos de máquinas virtuais.

8.1 GOOGLE CLUSTER TRACE

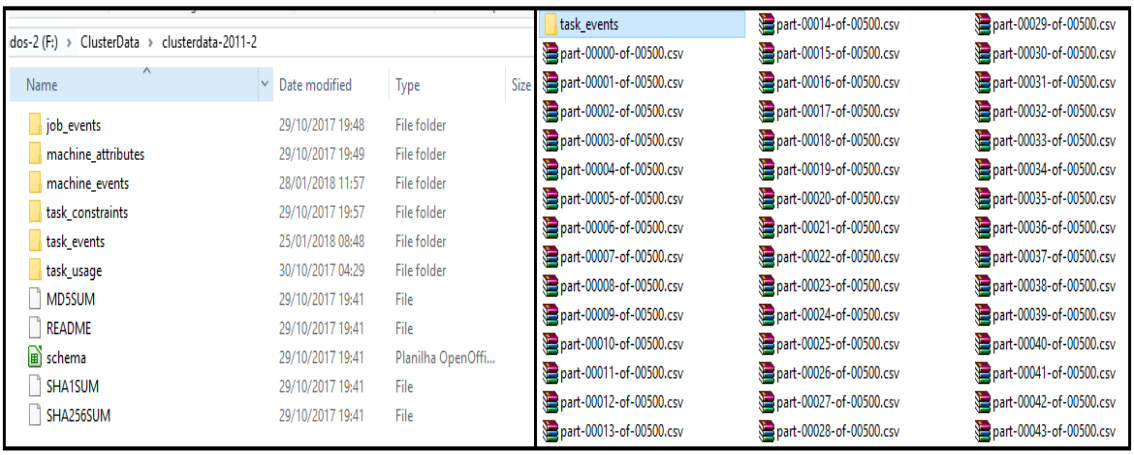
O Google é conhecido por ser uma das maiores empresas no ramo de TI mundial. Dentre seus produtos, encontram-se diversos serviços em *cloud computing*. Em 2011, foi lançado pela empresa, um *dataset* contendo informações de 29 dias de atividade de um de seus *datacenters*, o qual foi atualizado em 2014. O acesso ao *dataset* é livre e gratuito, podendo ser realizado através da aplicação Google Cloud Platform.

O pacote contém centenas de arquivos em formato *.csv*, organizado em diferentes diretórios. Dentre as diversas tabelas que o compõem, existem, por exemplo, aquelas relativas a máquinas do cluster de CC, as relativas a *jobs* e tarefas. Dentro de cada tabela, os registros são armazenados na forma de linhas. O formato de cada tipo de registro depende da categoria em que ele está associado. Por exemplo, eventos de máquina possuem atributos *timestamp*, identificador (ID) de máquina, tipo de evento, ID de plataforma, capacidade de CPU e de memória. A estrutura do conjunto de dados acima citado é demonstrado através da figura 8.1.

[4] http://www.caida.org/data/passive/ddos-20070804_ dataset.xml

[5] https://mcfp.weebly.com/the-ctu-13-dataset-a-labeled-dataset-with-botnet-normal-and-background-traffic.html

Figura 8.1: Estrutura do conjunto de dados.



Fonte: Próprio autor.

Dois conjuntos de dados em específico demonstram ser úteis para o desenvolvimento desta etapa de avaliação, sendo elas as tabelas *taskEvents* e *machineEvents*. Estas tabelas contém dados sobre tarefas (*tasks*) e máquinas componentes do *cluster*, respectivamente. Para que seja melhor compreendido os formatos desses dados, são expostos na Figura 8.2.a e 8.2.b, trechos de uma das tabelas *taskEvents* e *machineEvents*, respectivamente.

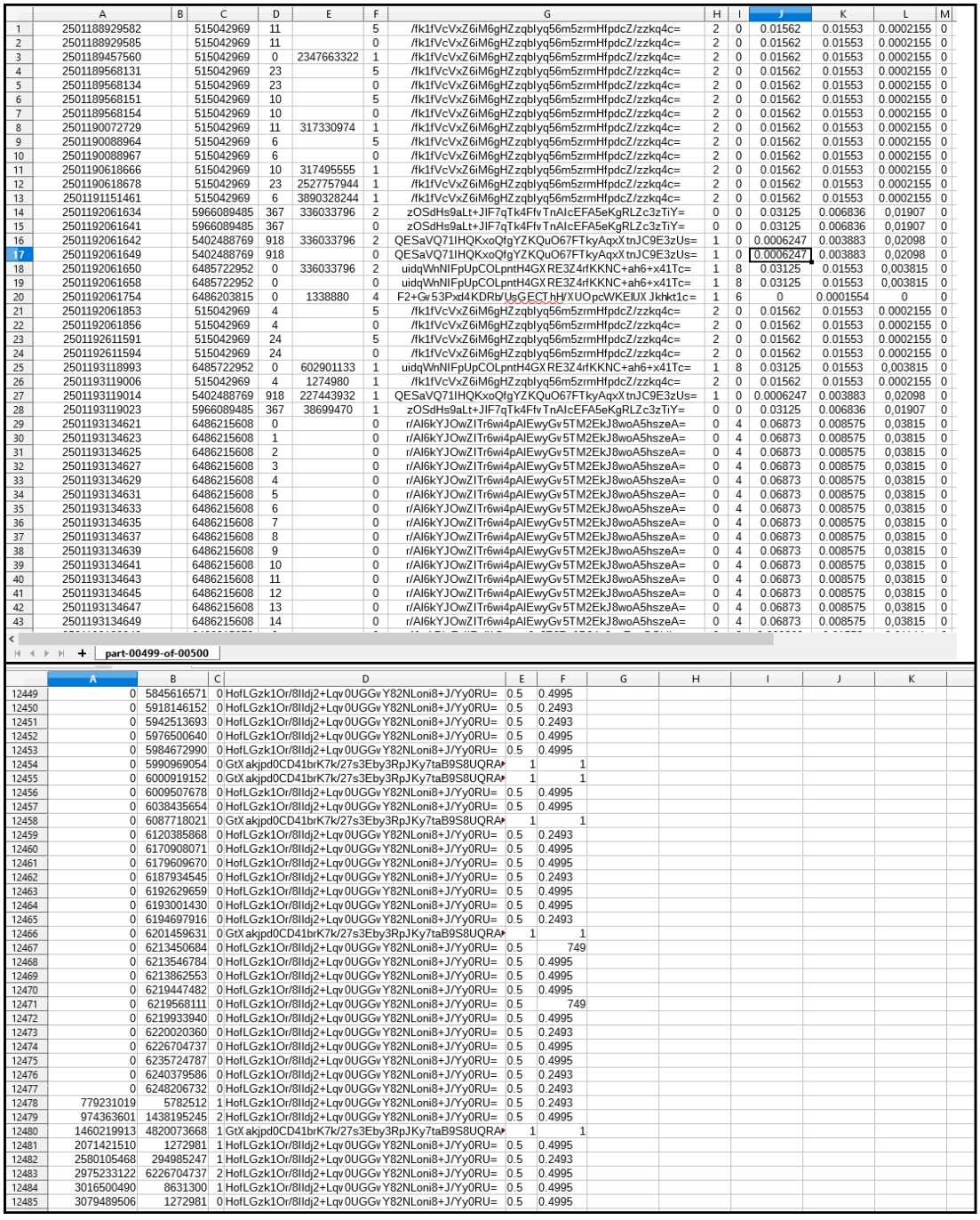
Cada coluna da tabela 8.2 representa uma variável aleatória qualitativa ou quantitativa a respeito das tarefas (instâncias). A estrutura da tabela de tarefas é descrita a seguir:

- Coluna "A": representa um instante de tempo denominado *timestamp*, expresso em microssegundos.

- Coluna "B": motivo de inclusão. Consiste de um critério de inclusão de um registro incompleto na tabela, representado por um número inteiro. Somente ocorre em eventos que não foram inicialmente capturados devido a falta de um ou mais metadados, mas após uma verificação de consistência dos registros, verificou-se sua ocorrência.

- Coluna "C": *Job* ID. Trata-se do número identificador dos *jobs* pertencente aos usuários. Jobs são unidades de nível hierárquico superior a tarefas, isto é, um usuário pode ter um ou mais *jobs* e cada *job* pode ter uma ou mais tarefas.

Figura 8.2: Estrutura do conjunto de dados.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	2501188929582		515042969	11		5	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
2	2501188929585		515042969	11		0	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
3	2501189457560		515042969	0	2347663322	1	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
4	2501189568131		515042969	23		5	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
5	2501189568134		515042969	23		0	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
6	2501189568151		515042969	10		5	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
7	2501189568154		515042969	10		0	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
8	2501190072729		515042969	11	317330974	1	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
9	2501190088964		515042969	6		5	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
10	2501190088967		515042969	6		0	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
11	2501190618666		515042969	10	317495555	1	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
12	2501190618678		515042969	23	2527757944	1	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
13	2501191151461		515042969	6	3890328244	1	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
14	2501192061634		5966089485	367	336033796	2	zOSdHs9aLt+JIF7qTk4FfvTnAIcEFA5eKgRLZc3zTiY=	0	0	0.03125	0.006836	0,01907	0
15	2501192061641		5966089485	367		0	zOSdHs9aLt+JIF7qTk4FfvTnAIcEFA5eKgRLZc3zTiY=	0	0	0.03125	0.006836	0,01907	0
16	2501192061642		5402488769	918	336033796	2	QESaVQ71IHQKxoQfgYZKQuO67FTkyAqxXtnJC9E3zUs=	1	0	0.0006247	0.003883	0,02098	0
17	2501192061649		5402488769	918		0	QESaVQ71IHQKxoQfgYZKQuO67FTkyAqxXtnJC9E3zUs=	1	0	0.0006247	0.003883	0,02098	0
18	2501192061650		6485722952	0	336033796	2	uidqWnNIFpUpCOLpntH4GXRE3Z4rfKKNC+ah6+x41Tc=	1	8	0.03125	0.01553	0,003815	0
19	2501192061658		6485722952	0		0	uidqWnNIFpUpCOLpntH4GXRE3Z4rfKKNC+ah6+x41Tc=	1	8	0.03125	0.01553	0,003815	0
20	2501192061754		6486203815	0	1338880	4	F2+Gv53Pxd4KDRb/UsGECThHYXUOpcWKEIUXJkhkt1c=	1	6	0	0.0001554	0	0
21	2501192061853		515042969	4		5	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
22	2501192061856		515042969	4		0	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
23	2501192611591		515042969	24		5	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
24	2501192611594		515042969	24		0	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
25	2501193118993		6485722952	0	602901133	1	uidqWnNIFpUpCOLpntH4GXRE3Z4rfKKNC+ah6+x41Tc=	1	8	0.03125	0.01553	0,003815	0
26	2501193119006		515042969	4	1274980	1	/fk1fVcVxZ6iM6gHZzqblyq56m5zrmHfpdcZ/zzkq4c=	2	0	0.01562	0.01553	0.0002155	0
27	2501193119014		5402488769	918	227443932	1	QESaVQ71IHQKxoQfgYZKQuO67FTkyAqxXtnJC9E3zUs=	1	0	0.0006247	0.003883	0,02098	0
28	2501193119023		5966089485	367	38699470	1	zOSdHs9aLt+JIF7qTk4FfvTnAIcEFA5eKgRLZc3zTiY=	0	0	0.03125	0.006836	0,01907	0
29	2501193134621		6486215608	0		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
30	2501193134623		6486215608	1		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
31	2501193134625		6486215608	2		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
32	2501193134627		6486215608	3		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
33	2501193134629		6486215608	4		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
34	2501193134631		6486215608	5		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
35	2501193134633		6486215608	6		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
36	2501193134635		6486215608	7		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
37	2501193134637		6486215608	8		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
38	2501193134639		6486215608	9		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
39	2501193134641		6486215608	10		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
40	2501193134643		6486215608	11		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
41	2501193134645		6486215608	12		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
42	2501193134647		6486215608	13		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0
43	2501193134649		6486215608	14		0	r/Al6kYJOwZITr6wi4pAlEwyGv5TM2EkJ8woA5hszeA=	0	4	0.06873	0.008575	0,03815	0

part-00499-of-00500

a)

	A	B	C	D	E	F	G	H	I	J	K
12449	0	5845616571	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12450	0	5918146152	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.2493					
12451	0	5942513693	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.2493					
12452	0	5976500640	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12453	0	5984672990	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12454	0	5990969054	0	GtXakjpd0CD41brK7k/27s3Eby3RpJKy7taB9S8UQRA=	1	1					
12455	0	6000919152	0	GtXakjpd0CD41brK7k/27s3Eby3RpJKy7taB9S8UQRA=	1	1					
12456	0	6009507678	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12457	0	6038435654	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12458	0	6087718021	0	GtXakjpd0CD41brK7k/27s3Eby3RpJKy7taB9S8UQRA=	1	1					
12459	0	6120385868	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.2493					
12460	0	6170908071	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12461	0	6179609670	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12462	0	6187934545	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.2493					
12463	0	6192629659	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12464	0	6193001430	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12465	0	6194697916	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.2493					
12466	0	6201459631	0	GtXakjpd0CD41brK7k/27s3Eby3RpJKy7taB9S8UQRA=	1	1					
12467	0	6213450684	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	749					
12468	0	6213546784	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12469	0	6213862553	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12470	0	6219447482	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12471	0	6219568111	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	749					
12472	0	6219933940	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12473	0	6220020360	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.2493					
12474	0	6226704737	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12475	0	6235724787	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12476	0	6240379586	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.2493					
12477	0	6248206732	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.2493					
12478	779231019	5782512	1	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.2493					
12479	974363601	1438195245	2	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12480	1460219913	4820073668	1	GtXakjpd0CD41brK7k/27s3Eby3RpJKy7taB9S8UQRA=	1	1					
12481	2071421510	1272981	1	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12482	2580105468	294985247	1	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.2493					
12483	2975233122	6226704737	2	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12484	3016500490	8631300	1	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					
12485	3079489506	1272981	0	HofLGzk1Or/8IIdj2+Lqv0UGGvY82NLoni8+J/Yy0RU=	0.5	0.4995					

b)

Fonte: Próprio autor.

- Coluna "D": *Task* Index. É o índice de cada tarefa compreendida dentro de um *job*. Pode-se ser visto também como um ID das tarefas contidas em um determinado *job*. Uma observação importante é que tarefas com o mesmo *taskID*, mas associadas a *jobID* diferentes são tarefas diferentes.

- Coluna "E": *Machine ID*. Representa o identificador único da máquina do *cluster* em que a tarefa foi alocada para processamento.

- Coluna "F": tipo de Evento. É constituído de um número inteiro entre zero e nove, sendo que zero representa a submissão da tarefa pelo usuário, ou seja, seu pedido de escalonamento e execução.

- Coluna "G": contém um *hash* codificado em 64 bits que representa o nome do usuário proprietário dos *jobs* e das tarefas. A apresentação se dá desta forma pela razão de garantir o anonimato e a privacidade dos usuários do *datacenter* em que foram capturados os registros. Vale também salientar que cada *hash* neste cenário é único, ou seja, registros com *hashes* iguais significam que pretencem ao mesmo usuário.

- Colunas "H"e "I": classe de escalonamento e prioridade, respectivamente.

- Colunas "J"e "K": quantidade de RAM e CPU requisitado pela tarefa, respectivamente, sendo normalizados para um valor decimal entre 0 e 1.

- Coluna "L": contém o requisito de espaço em disco no servidor que executará a instância, normalizado de forma similar aos valores de CPU e memória.

- Coluna "M": exigência de execução em máquinas diferentes: caso seja definido como "1", indica ao escalonador da nuvem que a instância deve executar em uma máquina física distinta das demais pertencentes ao mesmo *job*.

Cada linha da tabela 8.1 contém informações sobre um evento relativo a uma das diversas instâncias que executam no *cluster*, sendo que uma instância é identificada neste caso pela tupla [*jobID*, *taskID*] associada a um identificador de usuário (*userID*). Desta forma, é possível identificar cada instância de forma única através da tripla [*userID*, *jobID*, *taskID*].

Já no que se refere especificamente à tabela machineEvents, tem-se que as colunas "B"até "F"são: ID de máquina, tipo de evento, ID da plataforma (microarquitetura da máquina), capacidade de CPU e capacidade de memória. Os tipos de eventos são representados por valores inteiros de zero a 2, indicando a adição, remoção ou atualização da máquina no *cluster* da nuvem. Para fins de privacidade, a arquitetura

das máquinas são representadas sob a forma de *hash* e os valores de CPU e memória são normalizados.

8.2 UTILIZANDO O CONJUNTO DE DADOS

Segundo o guia de referência publicado por REISS; WILKES; HELLERSTEIN (2014), cada tarefa constitui um *container linux*. Vale ressaltar que, conforme exposto na seção 2.1, ASC em IaaS podem ser executados tanto entre VM quanto entre *containers*. ALJAHDALI et al. (2014) publicou um trabalho em que é feita uma análise do *dataset* em questão, no intuito de identificar possíveis ocorrências de ASC. Para tal, o autor fez uso dos dados contidos dentro das tabelas *taskEvents*, contabilizando o número de tarefas mortas pelo usuário (*killedTasks*). Ressalta-se que um maior número de tarefas mortas acaba por elevar também o número de lançamento de tarefas. Ao final do estudo, o autor conclui que, através da análise destes parâmetros, que existem três usuários suspeitos de praticarem ASC, o que reforça a viabilidade do uso de tal *dataset* para avaliação do sistema.

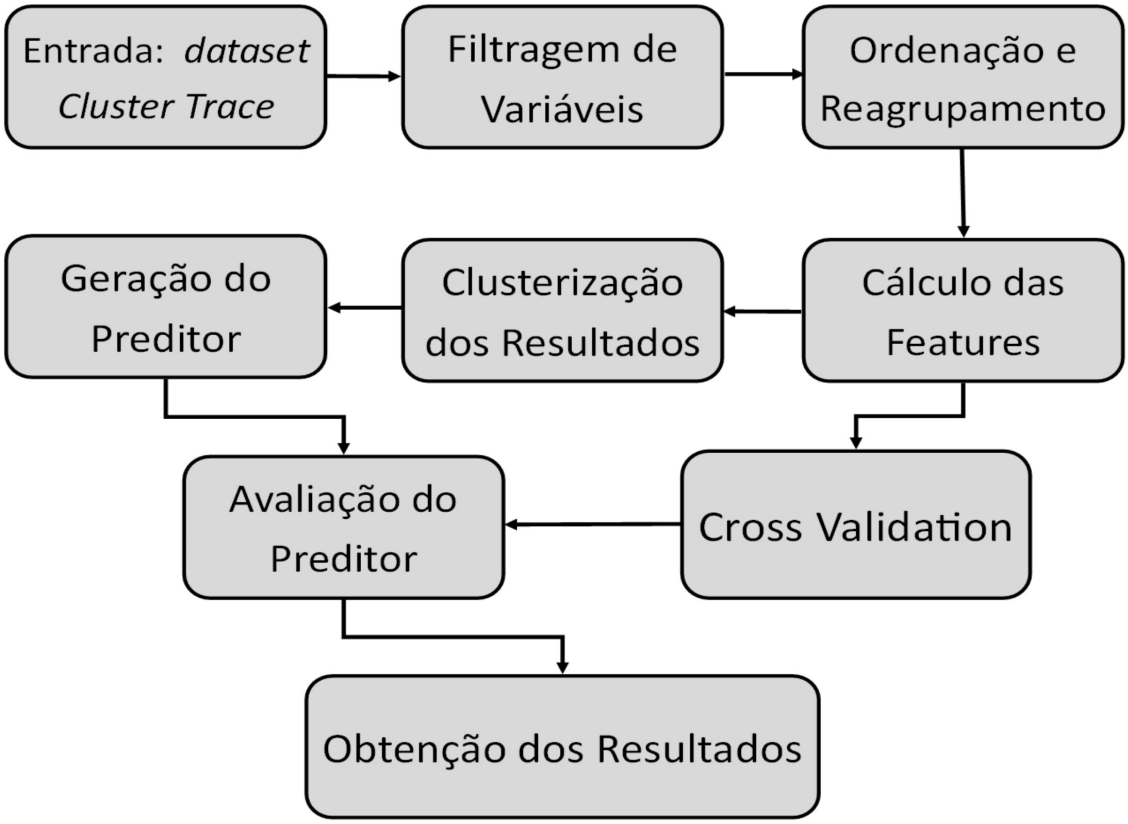
A utilização do *dataset* em questão foi dotada de um conjunto de passos (Figura 8.3) necessários para adequá-lo ao cenário utilizado, tanto no que diz sentido à extração das informações de interesse, quanto a utilização de recursos computacionais.

A entrada de dados representa o primeiro passo para o uso do *dataset* na avaliação do sistema e constituiu em identificar quais tabelas seriam necessárias e os campos adequados para que fosse possível realizar uma filtragem em seu conteúdo. As tabelas selecionadas foram *machineEvents* e *taskEvents*.

Na sequência, foi realizada uma filtragem de dados, de forma a manter apenas informações relevantes para o cálculo das *features* analisadas pelo sistema. Neste contexto, a primeira tabela foi útil para a determinação do número de servidores locados no *cluster* através de uma contagem de ID de máquinas, onde valores repetidos foram desconsiderados. Já os campos mantidos no estágio de filtragem das tabelas de eventos de tarefas foram: *timestamp*, *job*ID, *task*ID, tipo de operação e *hash* (ID) do usuário.

Em seguida, as tabelas foram carregadas sequencialmente e reordenadas, de acordo com ID de usuário passo 3 do diagrama da Figura 8.3. Ressalta-se que Através

Figura 8.3: Passos adotados para a utilização do *dataset*.



Fonte: Próprio autor.

destas variáveis, é possível determinar as *features* (passo 4), sendo elas: total de lançamentos, número de instâncias e tempo de atividade (vide capítulo 7) Ainda sobre as *features*, salienta-se que um número maior de *killedTasks*, métrica utilizada por ALJAHDALI et al. (2014) acaba implicando maiores registros de lançamentos. Por conseguinte, maior número de lançamentos vem de encontro ao modelo de ataque em que se busca identificar, de acordo com CHIAPPETTA; SAVAS; YILMAZ (2016) e SECLUDIT (2013).

As *features calculadas* de cada usuário são utilizadas na de *clusterização* dos dados, onfr o conjunto gerado é utilizado para a geração do modelo preditor. Concomitantemente, o conjunto de *features* alimenta o algoritmo de *cross validation*, que tem por objetivo estimar um *dataset* de testes.

Por fim, o *dataset* estimado pela etapa de *cross validation* é utilizado para a avaliação do modelo preditor construído, de forma a se obter os resultados do modelo. Estas etapas também são apresentadas pelo algoritmo 2.

Algoritmo 2: Filtragem e agrupamento dos usuários

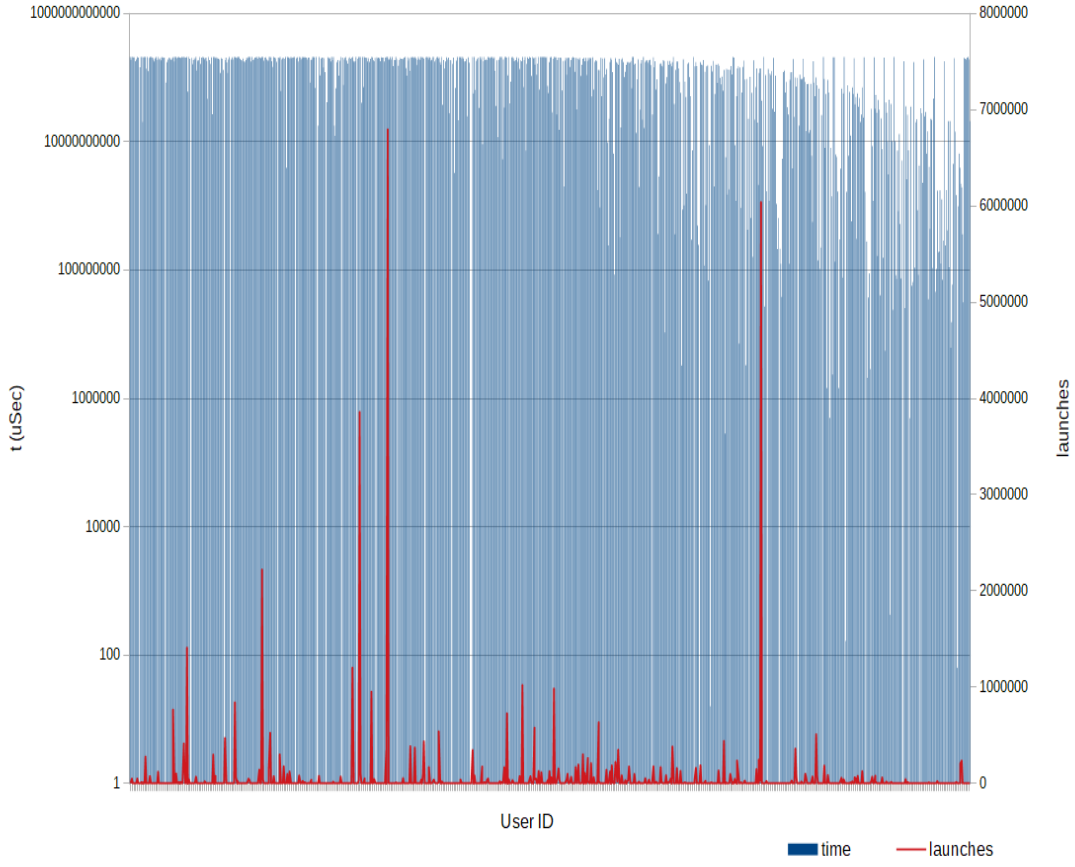
```
inicialização;
String linha, novaLinha, usuariosConhecidos[];
long timestamp, jobID, taskID;
Arquivo arquivoUsuario;
para todas as tabelas parciais faça
    linha <- leia uma linha;
    timestamp <- linha[0];
    jobID <- linha[2];
    taskID <- linha[3];
    userID <- linha[4];
    if usuariosConhecidos contem userID then
        novaLinha <- timestamp, jobID, taskID, userID;
        adicione novaLinha ao arquivoUsuario;
    else
        adicione userID a usuariosConhecidos[];
        crie um arquivo para o usuario userID;
        novaLinha <- timestamp, jobID, taskID, userID;
        adicione novaLinha ao arquivoUsuario;
    end
fim
```

Como pode ser visualizado, o algoritmo 2 consiste em agrupar todas as ocorrências de eventos de acordo com cada usuário da nuvem, mantendo apenas as variáveis de interesse (*features*) *timestamp, jobID, taskID, userID*. Os eventos registrados em cada uma das tabelas parciais foram separados em arquivos específicos, referentes a cada usuário. Ou seja, um determinado arquivo da etapa pós-filtragem contém eventos de um único usuário específico. Através desta metodologia, foi possível obter uma significativa redução no tamanho dos arquivos, reduzindo-se de aproximadamente 16GB para 3,5GB.

Este algoritmo de filtragem de variáveis de interesse foi implementado na linguagem Java, e representa um alto custo de processamento dentro da validacão do método. Versões iniciais do código consuiram mais de uma semana para a execução, mas após um etapa de otimização de complexidade de algoritmo, foi possível reduzir este custo para aproximadamente um dia de execução. Embora havendo este ônus, a redução de tamanho do *dataset*, aliada à nova disposição dos dados, constituiu de uma etapa essencial para a construção do modelo preditivo, por reduzir tempo de processamento e custo de memória das etapas seguintes.

Conforme afirmado anteriormente, a etapa seguinte à filtragem, ordenação e reagrupamento consistiu em calcular as *features*, sendo elas: número de instâncias lançadas n_i, tempo de uso t e o total de lançamentos n_l. O posicionamento das instâncias dentro do *cluster* está contido nos registros da tabela, porém, esta informação em não interfere nos cálculos de probabilidade efetuados na metodologia. Por fim, a posse das informações acima descritas já torna possível a visualização de um gráfico geral de lançamento de instâncias (Figura 8.4).

Figura 8.4: Gráfico de lançamentos dos usuários



Fonte: Próprio autor.

O gráfico exposto é constituído do tempo total de uso registado (barras, eixo Y principal) em conjunto com a média de lançamentos (linha, eixo Y secundário), para cada usuário. Nota-se a presença de três grandes picos na curva de lançamentos, o que sugere um comportamento fora do normal associado a estas três observações, o que confere com aos achados de ALJAHDALI et al. (2014).

Até o presente momento, esta seção retratou os processos utilizados na adequação do *dataset* para que a disposição das informações fosse compatível com a metodologia em desenvolvimento. O passo seguinte após esta análise inicial constitui no emprego do algoritmo de classificação para que seja possível "ensinar" ao modelo preditor como reconhecer comportamentos suspeitos. Para isso, foram realizados testes com dois algoritmos de classificação distintos: o K-means e o algoritmo PAM, sendo o primeiro o algoritmo definitivo usado.

8.2.1 Classificação e Predição

Conforme afirmado na seção 4, é necessário que o modelo preditor possua uma base de conhecimento para que seja possível predizer novos acontecimentos. Nesse sentido, foram realizados testes com os algoritmos classificadores K-means e PAM (*Partitioning around Medoids*). Primeiramente, aplicando-se o K-Means com base nas variáveis n_i, $tempo$ e n_l, é possível gerar os gráficos dispostos em 8.5.

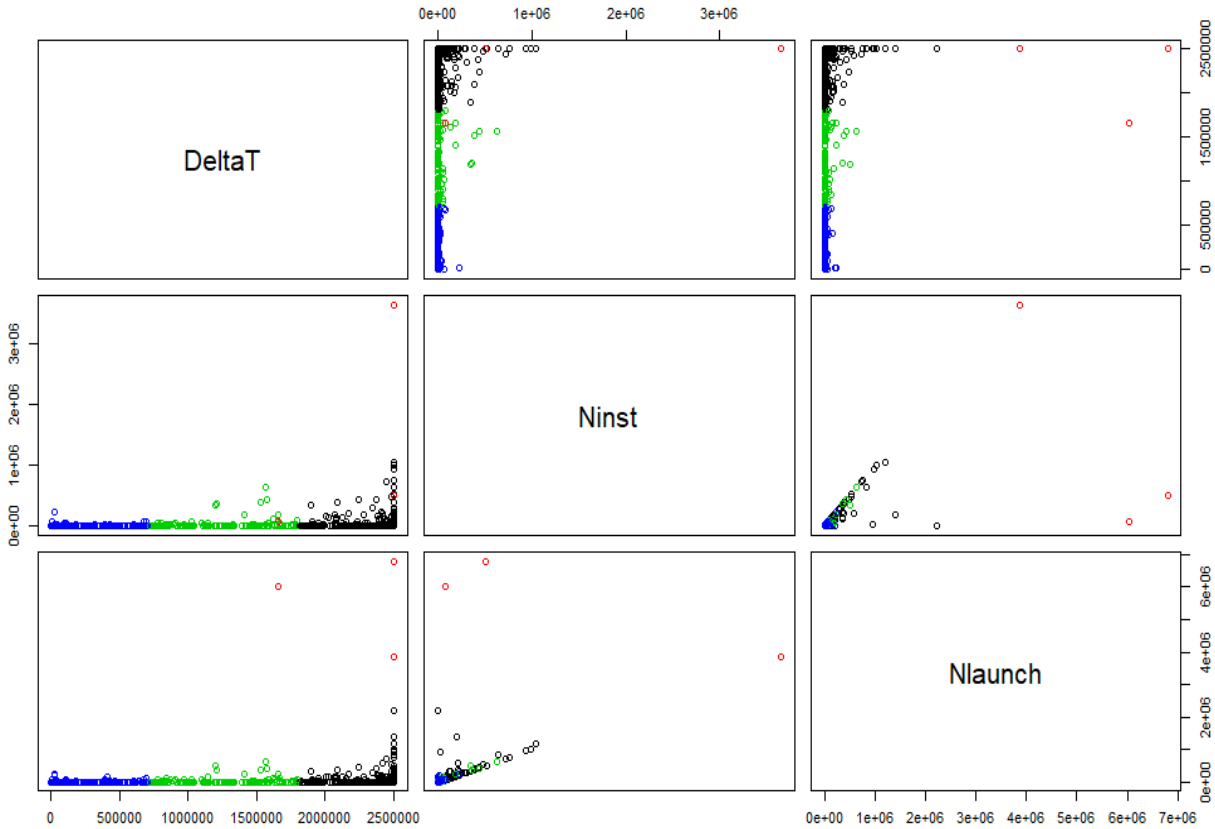
Na primeira linha, tem-se os gráficos gerados pelo cruzamento do tempo (eixo Y) dos usuários com o número de instâncias e número de lançamentos, respectivamente. Na segunda linha, observam-se os gráficos gerados pelo cruzamento do número de instâncias (eixo Y) com o tempo e número de lançamentos. Por fim, na última linha, estão dispostos os gráficos gerados pelo número de lançamentos (eixo Y) e tempo.

Nota-se que o primeiro gráfico inferior, à esquerda, remete ao comportamento do histograma de lançamento dos usuários (gráfico 8.4). Também é possível visualizar que o algoritmo k-means conseguiu classificar satisfatoriamente os três principais pontos suspeitos, de acordo com aquele mesmo gráfico. Visualmente, estas três ocorrências encontram-se em pontos vermelhos.

O primeiro gráfico à esquerda da linha do meio traz outra informação interessante: nem sempre um usuário suspeito é aquele que possui o maior número de instâncias em sua conta (pontos em vermelho). Já o segundo gráfico da linha inferior, confirma o padrão citado na literatura, em que são lançadas muitas instâncias para a realização dos testes de coresidência. Também, identifica-se que o maior número de lançamentos não necessariamente está atrelado ao usuário com mais instâncias.

Ainda com relação à classificação com o K-means, é exibido o gráfico 8.6, o

Figura 8.5: Gráficos de classificação do conjunto de treinamento com algoritmo K-Means

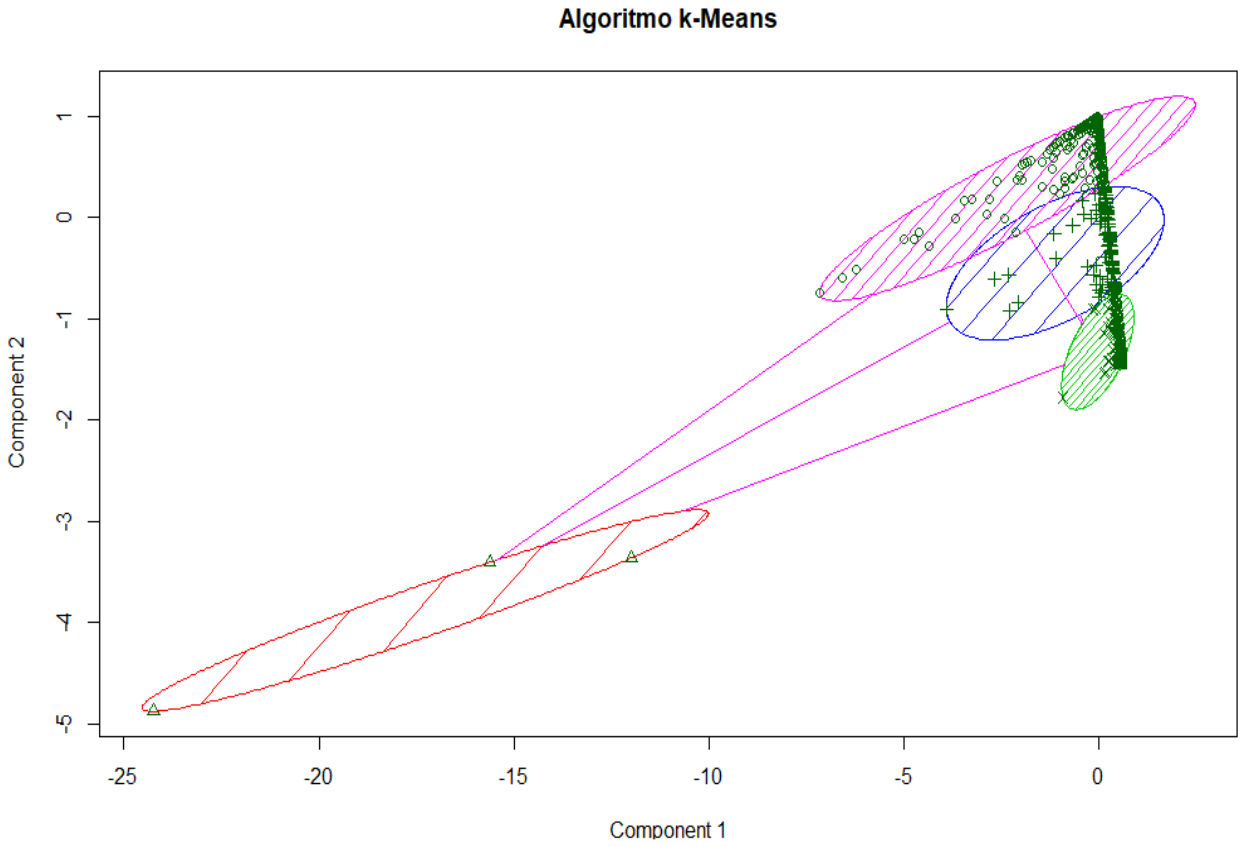


Cada cor da figura representa uma classe de eventos. Em vermelho: classe suspeita de ataque. Demais cores: ocorrências normais. Fonte: Próprio autor.

qual permite que os *clusters* gerados sejam visualisados de forma mais clara. O algoritmo conseguiu uma proporção da variância total explicada de 0,9269, ou seja, conseguiu agrupar os valores com uma taxa de acerto de 92,69%. Ainda sobre a mesma figura, observa-se que os três triângulos envolvidos pela elipe hachurada em vermelho consistem das três ocorrências suspeitas, o que demonstra uma taxa de 100% de acerto na classificação dos verdadeiros-positivos.

A classificação dos dados com o uso do PAM proporciona a construção dos mesmos tipos de gráfico, conforme estão representados pelas figuras 8.7 e 8.8. A disposição dos gráficos da primeira figura segue a mesma ordem dos gráficos contidos em 8.5

Figura 8.6: Classificação do conjunto de treinamento com algoritmo K-Means.

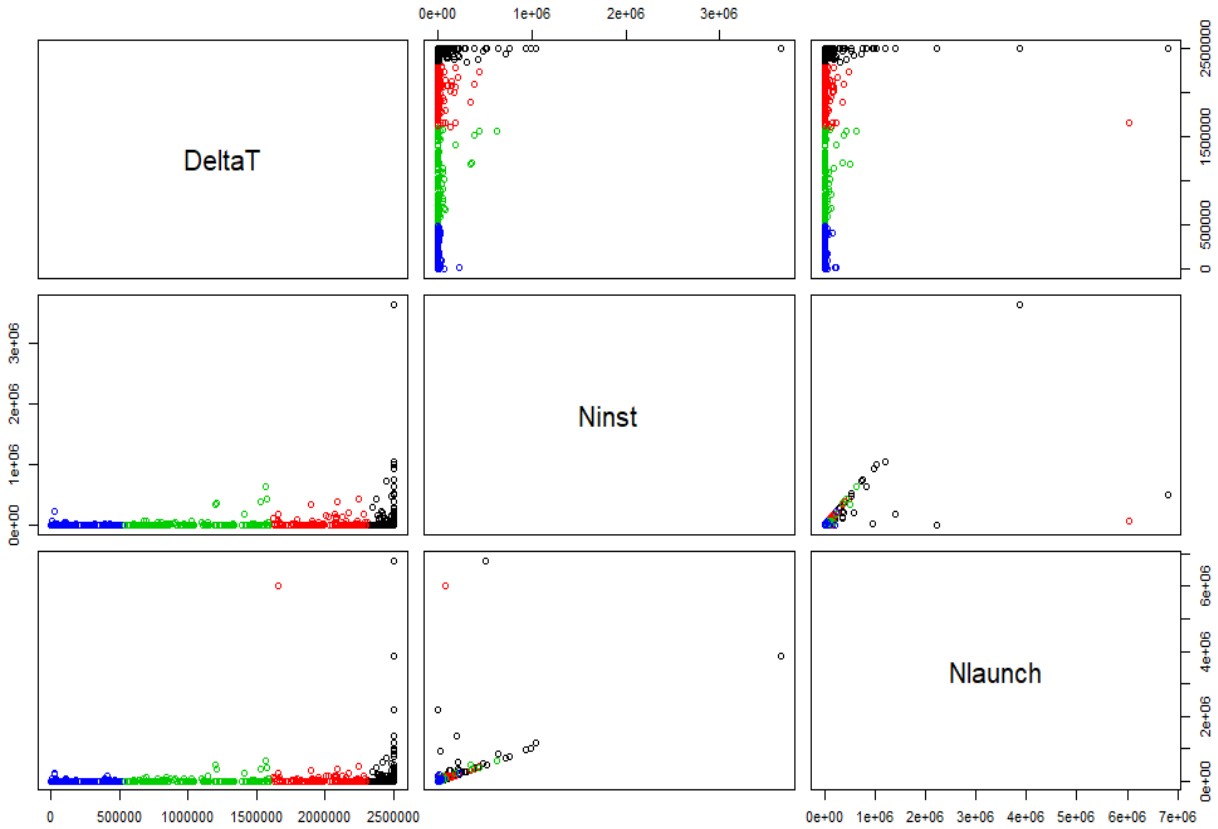


Cada elipse representa uma classe. Elipse hachurada em vermelho: instâncias suspeitas. Fonte: Próprio autor.

Após analisar os gráficos relativos ao PAM, fica visível que este algoritmo não se comportou adequadamente para a classificação do conjunto de dados. A qualidade da classificação por este método ficou em 55%. Desta forma, ficou definido no uso do K-Means como método classificador do sistema.

Como tratam-se de três o número de variáveis utilizadas durante o processo de classificação dos dados utilizados na metodologia deste trabalho, podem ser gerados gráficos tridimensionais para sua melhor visualização. A figura 8.9 representa tridimensionalmente as saídas dos algoritmos de classificação.

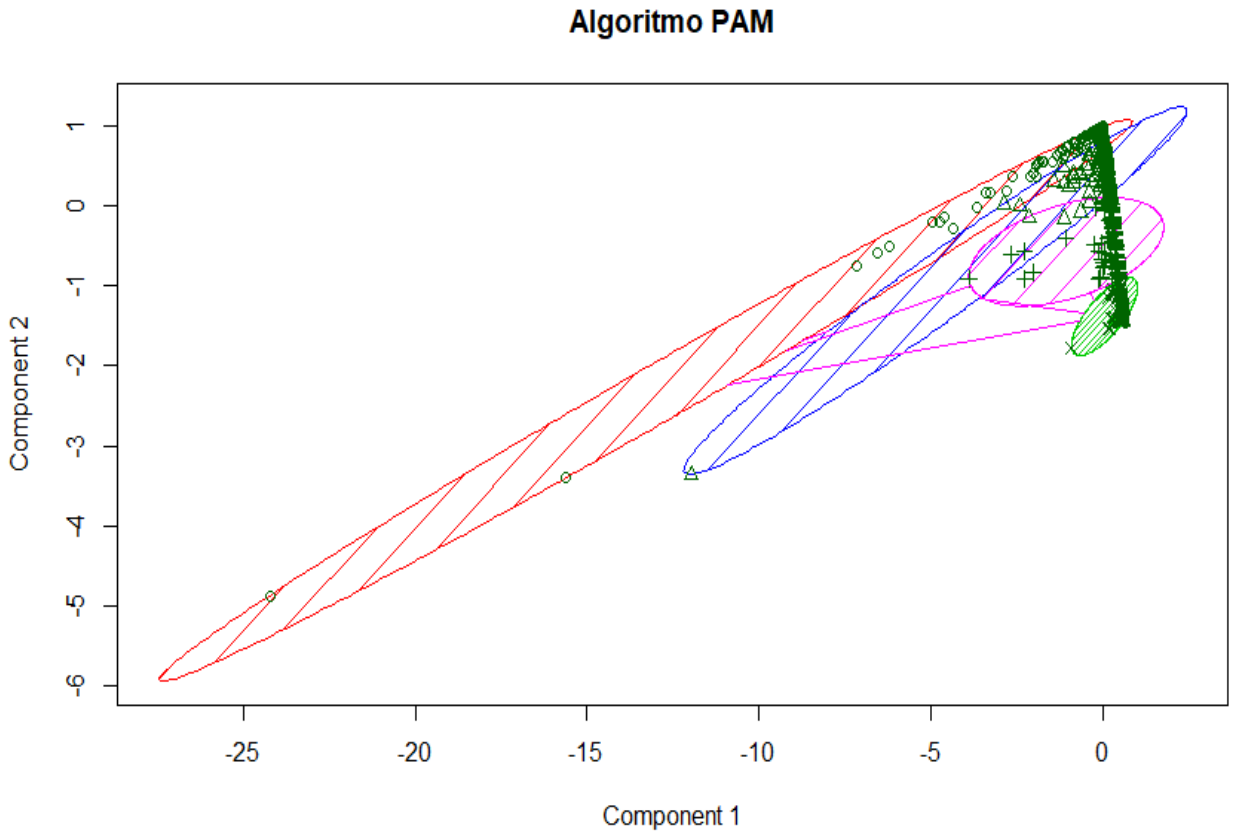
Figura 8.7: Gráficos de classificação do conjunto de treinamento com algoritmo PAM



Cada cor da figura representa uma classe de eventos. Em vermelho: classe suspeita de ataque. Demais cores: ocorrências normais. Fonte: Próprio autor.

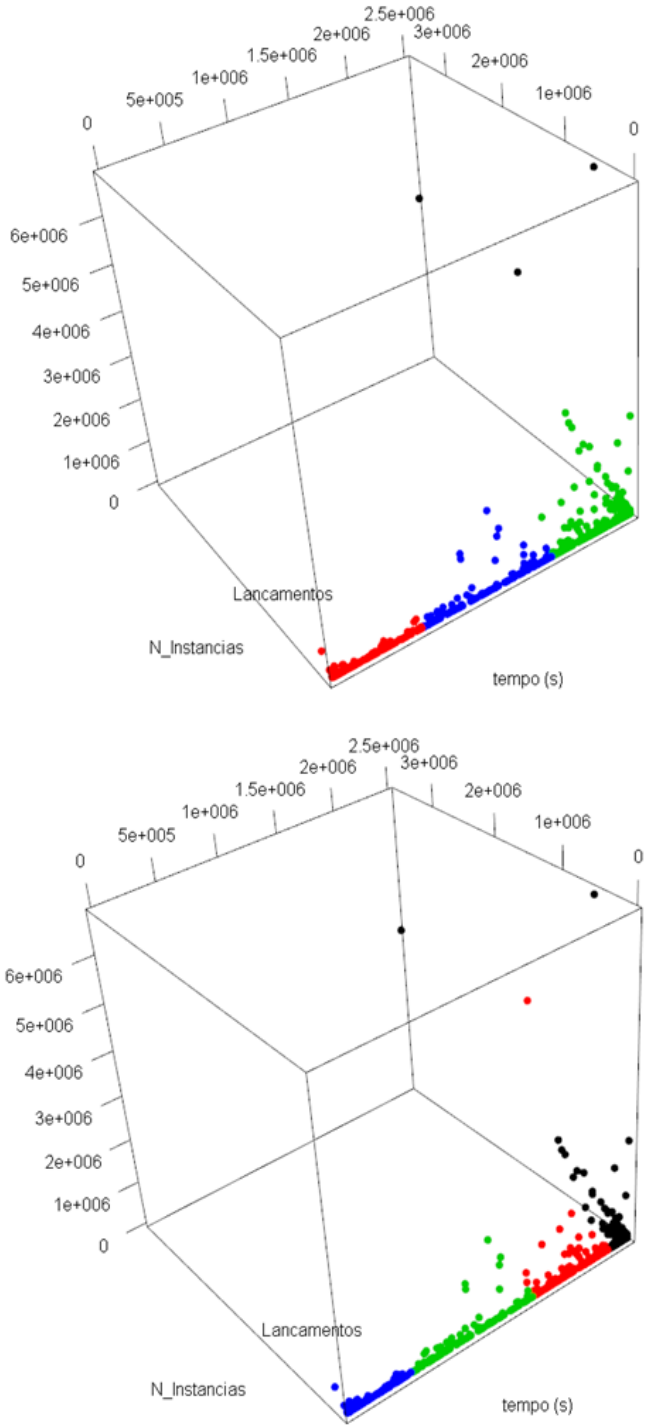
Em ambos os casos, os pontos representados em preto constituem a classe suspeita de ataque, enquanto as demais cores representam comportamentos normais. Visualiza-se em especial, que o classificador PAM comete um erro ao classificar uma das instâncias suspeitas como normal (figura 8.9 inferior), enquanto o K-means as isola corretamente.

Figura 8.8: Classificação do conjunto de treinamento com algoritmo PAM.



Fonte: Próprio autor.

Figura 8.9: Gráficos tridimensionais de classificação dos dados.



Cubo superior: classificação com K-Means. Cubo inferior: Classificação com PAM. Em preto: classe suspeita. Fonte: Próprio autor.

Com a conclusão desta etapa, utilizou-se SVM multiclasse como método preditor, onde a entrada para treinamento da SVM consistiu dos agrupamentos gerados pela aplicação do K-means. Foram testados três diferentes tipos de *kernel*, sendo eles: radial, sigmoidal e linear. O modelo treinado foi posteriormente testado através do emprego do método conhecido como *cross validation*, ou validação cruzada. Foram analisados os valores de prevalência, taxa de positivos, taxa de negativos e acurácia balanceada de cada classe em cada método, bem como a acurácia geral de cada método. Estes valores são representados na tabela 8.1.

Observando os dados contidos na tabela, primeiramente descobre-se que a prevalência de comportamentos que não-suspeito (classes 0, 1 e 2) correspondem a 30,11%, 13,33% e 56,24%, respectivamente. A prevalência de comportamento não suspeito somada, representa um total de 99,68%, já a prevalência de suspeita de ataque é de 0,32%.

Analisando os parâmetros de avaliação de cada tipo de SVM, percebe-se que em geral, todas foram bem sucedidas para a classificação de comportamentos ditos normais. Este achado é razoável, uma vez que a prevalência de comportamento não-anômalo mostrou ser extremamente superior a de comportamento suspeito (99,68% vs. 0,32%). Contudo, inspecionando os números relativos aos verdadeiros positivos (VP), nota-se que as SVM baseadas em *kernel* radial e sigmoidal não conseguiram detectar nenhuma dos comportamentos suspeitos. Já a SVM linear obteve um *score* VP de 1.0, indicando que todas as suas classificações supostamente verdadeiras são, de fato, verdadeiras.

Ainda com relação à tabela, percebe-se a medida de Negativo Predito (NP) e Falso Positivo (FP). O valor de NP representa a taxa em que os valores classificados como negativos são, de fato, negativos. Ou seja, este valor é inversamente proporcional aos falsos-positivos (FP), sendo que este último é dado por: $FP = 1 - NP$. Assim, quanto maior o valor de NP, menor a taxa de falsos positivos.

Novamente, é visível que os três tipos de *kernel* possuiram baixos falsos positivos para comportamento não-anômalo. Já com relação à detecção de comportamento suspeito, o *kernel* radial teve o maior número de falsos positivos: $1 - 0,7767 \simeq 0,324$ = 32,4%, enquanto que as SVM com *kernels* sigmoidal e linear obtiveram valores bastante inferiores de falso-positivos.

Tabela 8.1: Dados de avaliação dos modelos de SVM

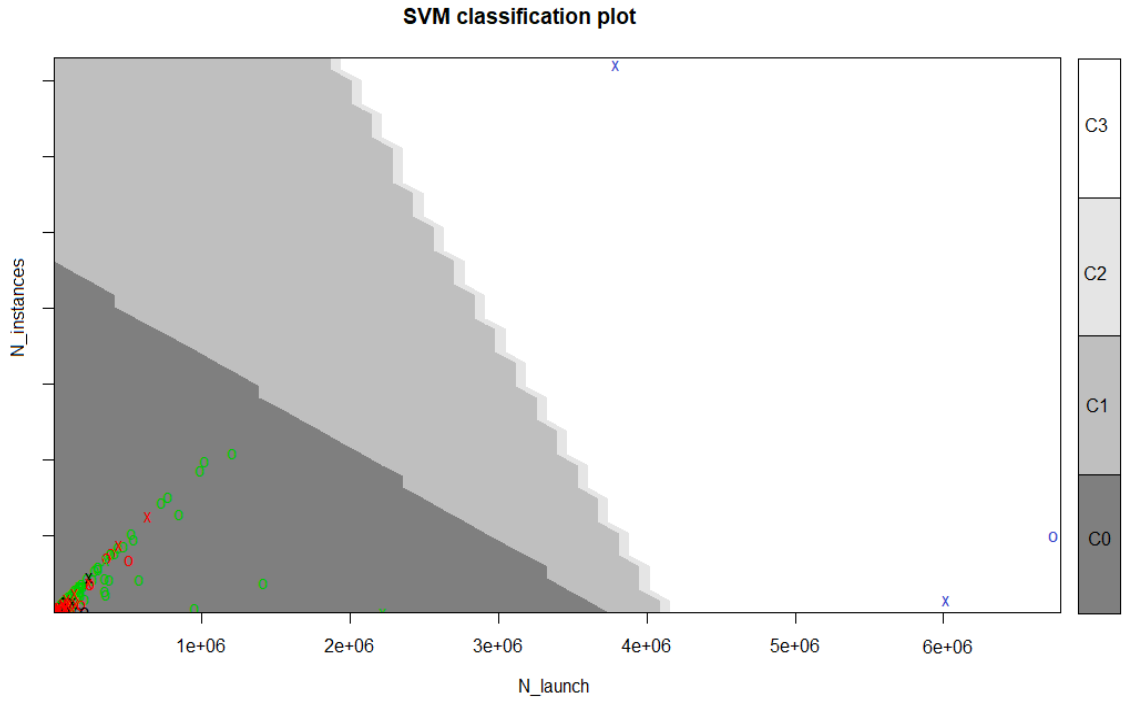
SVM Kernel	Valor	Classe 0	Classe 1	Classe 2	Classe 3 (suspeito)
Todos:	Prevalência Real	0,3011	0,1333	0,5624	0,003226
Radial	Verdadeiro Positivo	0,9964	1,0	0,9886	0
	Negativo Predito	0,9985	0,9938	0,9975	0,776767
	Falso Positivo	0,0015	0,0062	0,0025	0,223233
	Prevalência detect.	0,3022	0,1280	0,5677	0,002151
	Acurácia Balanceada	0,9967	0,9798	0,9917	0,498921
	Acurácia Geral			0,9892	
Sigmoidal	Verdadeiro Positivo	0,9894	0,9024	0,9810	0
	Negativo Predito	0,9985	0,9839	0,9802	0,996774
	Falso Positivo	0,0015	0,0061	0,00198	0,003226
	Prevalência detect.	0,3032	0,1323	0,5645	0
	Acurácia Balanceada	0,9959	0,9401	0,9801	0,5
	Acurácia Geral			0,9731	
Linear	Verdadeiro Positivo	0,9893	0,9832	0,9905	1.0
	Negativo Predito	0,9969	0,9832	0,9905	0,998922
	Falso Positivo	0,0031	0,0168	0,0095	0,001078
	Prevalência detect.	0,3022	0,1280	0,5677	0,002151
	Acurácia Balanceada	0,9941	0,9705	0,9939	0,833333
	Acurácia Geral			0,9892	

De forma geral, a acurácia dos três tipos de preditores foram de 98,92% (RBF), 97,31% (sigmoidal) e 98,92% (linear). Contudo apenas a SVM com *kernel* linear se demonstrou adequada para realizar a detecção de comportamento anômalo (acurácia balanceada de 0.833). Em termos de números de ataques detectados, este método conseguiu detectar 2 duas das 3 ocorrências registradas, o que é visível também através dos gráficos contidos em 8.10 e 8.11.

O primeiro gráfico mostra os hiperplanos de classificação no ponto de vista instâncias (eixo y) e lançamentos (eixo x). Já o segundo, mostra os planos sob o ponto de observação número de lançamentos (y) e tempo (x). A barra com tons de cinza, à esquerda, representa as classes utilizadas, sendo que C3 é a classe correspondente aos comportamentos anormais (vide tabela 8.1). As classes reais dos pontos estão representadas através de cores: preto, vermelho, verde e azul, sendo a última, a classe equivalente a ataques.

Enquanto o primeiro gráfico passa a impressão que as três ocorrências suspeitas (em azul) foram classificadas corretamente, por estarem no espaço correspondente a C3, o gráfico seguinte revela que na verdade, uma dessas observações ficou situada dentro do espaço correspondente a C2. Portanto, uma das observações suspeitas foi classificada erroneamente.

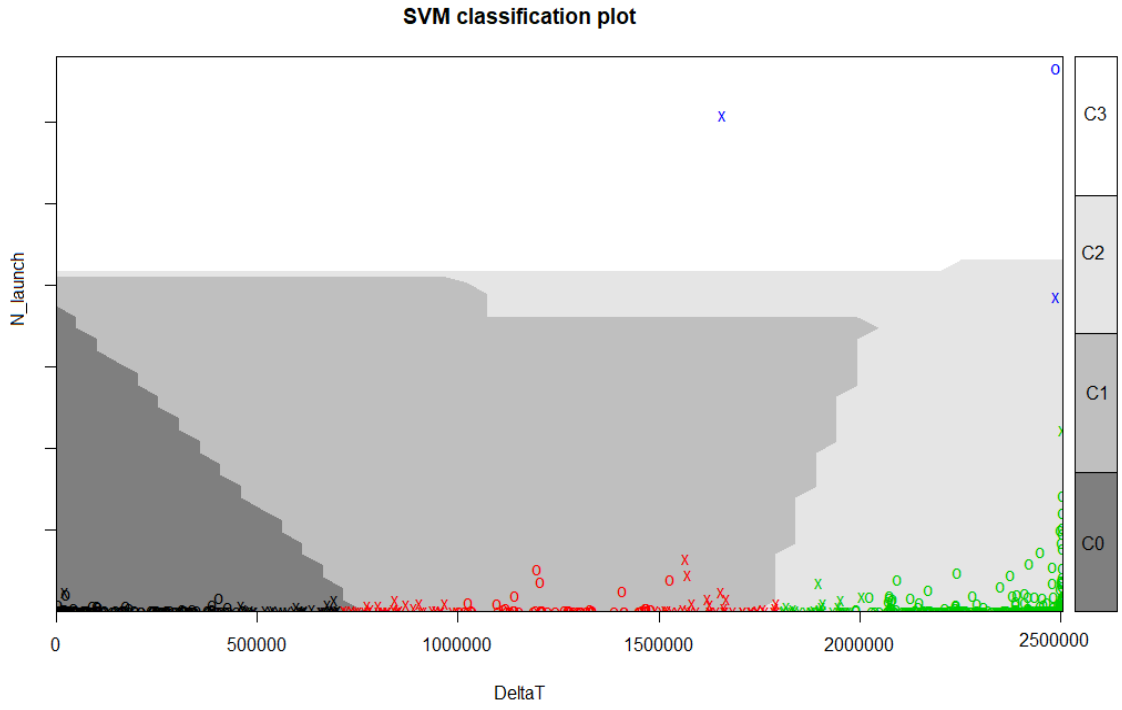
Figura 8.10: Hiperplanos classificadores em instâncias x lançamentos.



Fonte: Próprio autor.

Com base nos dados obtidos até o momento, pode-se afirmar que em geral, a associação do classificador K-means à uma SVM linear possui boa capacidade de predição de comportamento, com alta taxa de PV, baixa taxa de FP e, consequentemente, boa acurácia geral (98,92%). A taxa de acurácia balanceada na classe específica de ataques é relativamente alta, de 83%. Por fim, dando sequência à avaliação da proposta deste trabalho, apresentam-se os testes do módulo de geração de ruído, na seção seguinte.

Figura 8.11: Hiperplanos classificadores em tempo x lançamentos.



Fonte: Próprio autor.

8.3 TESTE DO GERADOR DE RUÍDO

Conforme apresentado anteriormente, a metodologia traz consigo a possibilidade de gerar ruído em *hosts* que hospedam usuários distintos, no intuito de dificultar a captura de informações do *cache* do processador. Isto é particularmente útil, no caso de um usuário malicioso resolver atacar uma máquina virtual aleatoriamente. Este cenário também deve ser considerado, pois mesmo sem realizar diversos testes e procedimentos de corresidência, existem chances do atacante não possuir um alvo pré-definido, bem como de que a co-alocação seja obtida já no primeiro lançamento.

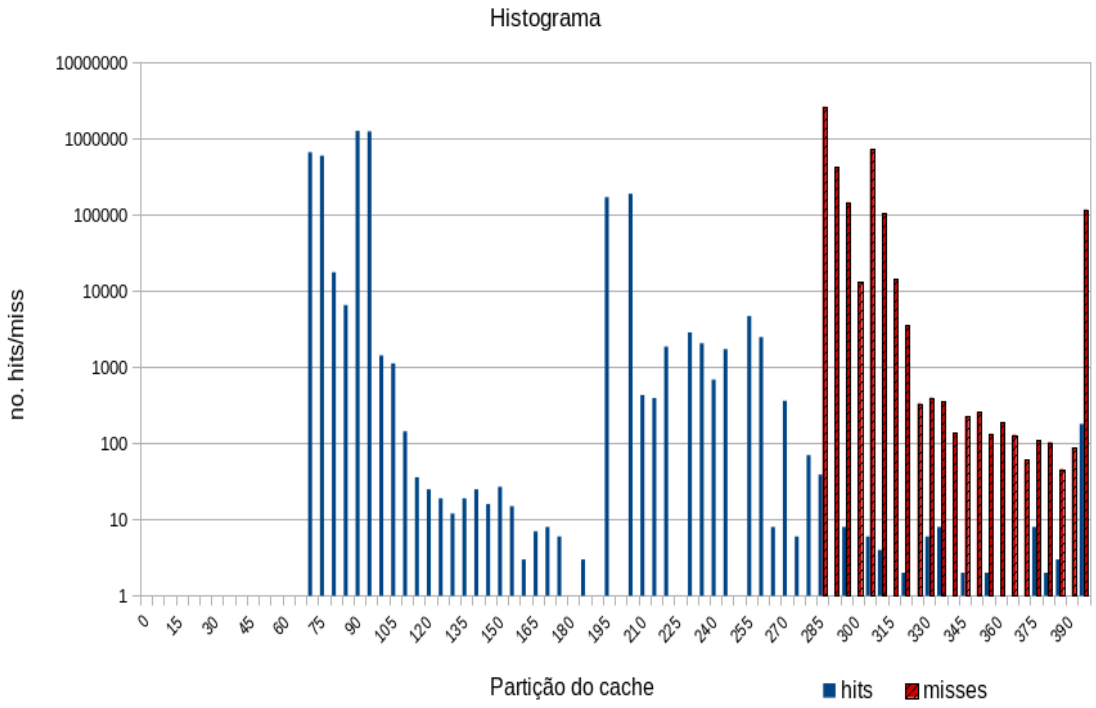
Nos testes deste recurso, foram criadas três máquinas virtuais alocadas dentro de um mesmo servidor do ambiente de testes criado. A primeira máquina foi configurada como sendo a vítima, onde fez-se uso da biblioteca OpenSSL para criptografar um conjunto de dados. A segunda VM, representa o gerador de ruído, executando o mesmo código da vítima, com a diferença da chave de segurança utilizada no processo criptográfico. Por fim, a última VM representa a VM do atacante, cujo objetivo é

extrair dados do *cache* da CPU.

Deve-se notar que por se tratar de uma prova de conceito, a instanciação estas três máquinas virtuais foi realizada manualmente. Todavia, a alocação automática do gerador de ruído e sua integração com o módulo de reação são recursos previstos pela metodologia apresentada (capítulo 7).

A bateria de testes constituiu em aplicar a suíte de ataques *side channel* denominada *Cache Template Attacks* (YAROM; FALKNER, 2014b) em três cenários distintos, sendo eles: 1) somente VM/código do atacante; 2) VM/código do atacante e VM/código da vítima; e 3) VM/código do atacante, VM/código da vítima e VM do gerador de ruído em execução. A metodologia de ataque adotada na ferramenta foi a Flush+Reload, descrita previamente neste trabalho. Antes da aplicação do método, realizou-se o teste de calibração e *profiling* do ambiente, na máquina do atacante. Os gráficos 8.12, 8.13 e 8.14 representam os histogramas de cada uma das situações acima descritas.

Figura 8.12: Histograma do Cenário 1.



Fonte: Próprio autor.

No primeiro teste (gráfico 8.12), observa-se o resultado do *profiler* executado

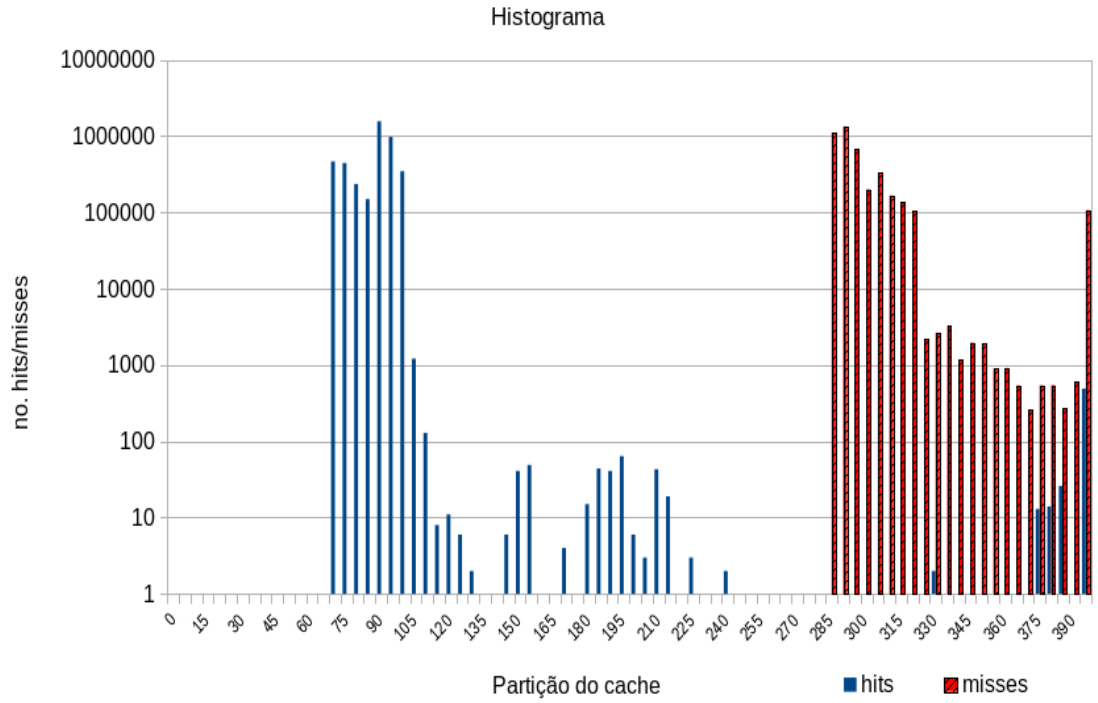
no cenário 1. As barras vermelhas hachuradas à esquerda representam a quantidade de *cache misses* ocorridos com relação à partição de cache testada, enquanto que as barras azuis, sem textura, representam a quantidade de *cache hits*. Um *hit* ocorre quando, durante a retomada da execução do código espião, o valor do endereço de *cache* lido for igual ao da leitura anterior, indicando seu uso recente. Já um cache *miss* ocorre quando o valor lido e o último valor conhecido são diferentes, indicando que os dados foram transferidos para a memória do sistema.

Por definição de YAROM; FALKNER (2014b), se ocorrer ao menos um registro de partição cujo valor mínimo de *cache hits* seja inferior ao máximo da quinta partição seguinte, fica constatado a viabilidade de aplicação do ataque de *side channel*. Observando-se o gráfico, fica evidente que é possível a realização do ataque, visto que existem diversas ocorrências em que a posição mais avançada de *cache* possui mais *hits* que as anteriores.

Já o gráfico 8.13 exibe o histograma gerado através do teste do cenário 2. Percebe-se que neste cenário, ocorre uma certa redução de *cache hits* com correspondente aumento de *cache misses*. Estas duas leituras demonstram ser coerentes, indicando que agora existe um processo (VM da vítima) concorrendo pelo uso de CPU do *host* físico. O aumento da ocorrência de *misses* indica a substituição de valores do *cache* pela vítima, fazendo com que aqueles inseridos pelo espião fossem transferido para a memória. Embora seja observado certa redução na quantidade de *hits* nas partições inferiores, o formato do histograma demonstra a aplicabilidade de um ASC.

Por fim, o gráfico 8.14 exibe o histograma obtido no cenário número 3. Observa-se um incremento substancial na ocorrência tanto de *hits* e *misses* tanto com relação ao cenário 1 e ao cenário 2. O aumento do número de *misses* é razoável, uma vez que neste cenário, existe um processo a mais concorrendo pela CPU em relação ao cenário 2; ou 2 processos a mais que o cenário 1. Por conta disto, a sobreescrita de certos endereços de memória ocorre com maior frequência. A disposição dos *hits* apontam que o ataque provavelmente ainda é possível no cenário em questão. Contudo, o aumento de *hits* concomitantemente ao aumento dos *misses* sugere a existência de interferência entre a definição do cache por parte do atacante e sua posterior verificação. Isto demonstra que o processo da VM atacante é preemptado pelo sistema operacional do *host* durante a definição do *cache*.

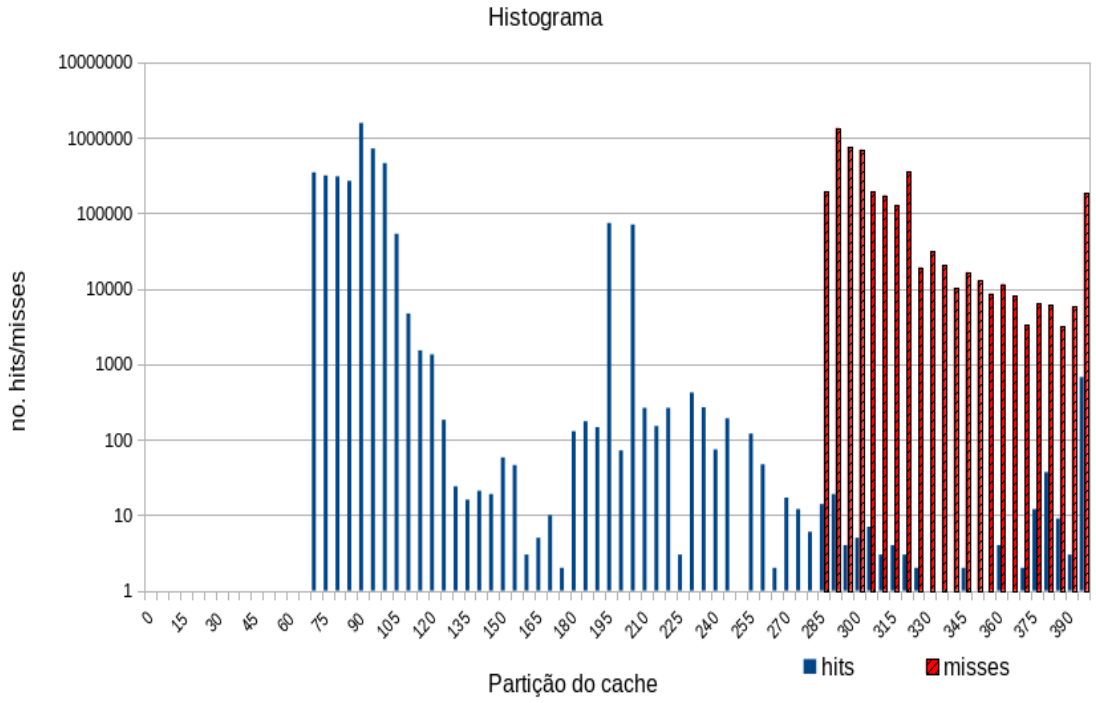
Figura 8.13: Histograma do Cenário 2



Fonte: Próprio autor.

Por conta desta preempção, pode-se inferir que os endereços iniciais obtiveram uma taxa falsamente elevadas de *hits* (similares ao primeiro gráfico), o que levanta a hipótese do ruído estar impactando no sucesso do ataque. Para melhor averiguar esta possibilidade, foi necessário avaliar os dados obtidos após a conclusão do processo de espionagem. Observou-se que o valor lido foi "0" para todas as partições do *cache*, ou seja, a chave secreta da VM vítima não vazou. Este achado aponta para a real aplicabilidade da geração de ruído como prevenção de ASCs aleatórios. Porém, ainda não é possível concluir a eficácia desta abordagem, o que torna necessário futuras pesquisas a respeito desta abordagem.

Figura 8.14: Histograma do Cenário 3



Fonte: Próprio autor.

8.4 DISCUSSÃO

Uma vez dispostos dos resultados obtidos nos testes de validação da metodologia, é possível analisar seus aspectos positivos e negativos, traçando um comparativo com as metodologias encontradas. Os trabalhos que a este se relacionam visam evitar ataques de SCA de quatro principais formas: evitando corresidência com migrações periódicas (AZAB; ELTOWEISSY, 2016), protegendo *cache* através de modificações no hypervisor (GODFREY; ZULKERNINE, 2013), protegendo *cache* através de alterações no *hardware* e *software* (KONG et al., 2013) e, por fim, detectando lançamentos anômalos de VM (SECLUDIT, 2013).

Embora as propostas baseadas em migrações periódicas dificultem o estabelecimento de corresidência, os recursos de rede tendem a ser um fator limitante. ZHANG et al. (2012) mitigou esta questão através do cálculo do intervalo de tempo ótimo entre as migrações. Contudo, deve-se lembrar do fato de que uma migração também oferece riscos (KHAN, 2016), e esta prática não se mostra adequada para evitar SCA

aleatórios.

As propostas baseadas em modificação do *hypervisor* possuem como ponto positivo a efetividade em evitar SCA, tendo a capacidade de evitar ataques aleatórios. Como fatores negativos, citam-se a necessidade de modificação do código-fonte do *hypervisor* e sua re-implantação, bem como o *overhead* de CPU, sendo de 15% no trabalho de GODFREY; ZULKERNINE (2013).

Metodologias com base em modificação de *hardware* podem no futuro, vir a ser uma boa solução para a vulnerabilidade em questão. A limitação desta abordagem consiste na existência da dependência dos fabricantes de *chips* desenvolverem e comercializarem uma nova arquitetura de seus produtos, imune a ataques.

Por fim, existe a abordagem de se detectar o lançamento anormal de máquinas virtuais, sendo este um fator indicativo do início de ASC. A solução desenvolvida por SecludIT atua exatamente neste aspecto, de uma forma relativamente simples. O principal ponto positivo daquela proposta é o fato de envolver pouca alteração no sistema de nuvem, uma vez que a ferramenta utiliza os *logs* do sistema como entrada de dados, os quais são armazenados na base OSSIM. As fragilidades, porém, consistem de não ser possível o ajuste de parâmetros de detecção, e a possibilidade de a atualização de *logs* do sistema ser tardia (*deferred update*).

Esta dissertação apresentou o desenvolvimento de uma nova ferramenta de segurança em *cloud*, cujo objetivo é mitigar a ameaça de SCA. Os principais aspectos positivos são a dinamicidade do módulo detector de lançamentos anômalos, boa acurácia da metodologia (98,92%) e a possibilidade do ruído gerado perturbar as leituras realizadas pelo processo espião. Exemplo de impacto na taxa de sucesso proporcionado pelo ruído é apresentado em IRAZOQUI et al. (2015), o que corrobora que ele pode vir a ser uma alternativa a modificações no código do *hypervisor*.

Um ponto ainda a ser enaltecido diz respeito à importância *features* utilizadas para a construção do modelo preditivo. O fato da necessidade do atacante estabelecer corresidência no mesmo servidor onde se encontra a vítima é consenso nas obras citadas (ALJAHDALI et al., 2014), (IRAZOQUI et al., 2015) (ZHANG et al., 2012), (INCI et al., 2015), GODFREY; ZULKERNINE (2013), (AZAB; ELTOWEISSY, 2016), (KHAN, 2016), (KONG et al., 2013). Os parâmetros número de lançamentos, tempo de atividade e total de instâncias do usuário são consenso entre autores como INCI et al.

(2015) e SECLUDIT (2013). Já o trabalho de ALJAHDALI et al. (2014) faz uma análise em termos de instâncias encerradas e intervalo de tempo, mas deve-se observar que um número maior de instâncias encerradas implicará um número maior de instâncias lançadas. Desta forma, pode-se concluir que as *features* tomadas são boas candidatas para a modelagem do sistema apresentado.

Deve ser ressaltado também que, devido ao fato do método apresentado receber entradas através de REST, torna-se necessária uma alteração no código-fonte do orquestrador da *cloud* a fim de estabelecer a comunicação entre os dois sistemas. Contudo, os benefícios desta abordagem como maximização escalabilidade de interação entre componentes e execução de forma segura (FIELDING, 2000) sobrepõe o ônus de customização de código.

8.4.1 Desafios e Limitações

Ao longo do desenvolvimento desta pesquisa foram encontrados diversos desafios a serem superados. Ao término deste trabalho existem naturalmente, limitações, sendo algumas delas referentes à forma com que a ferramenta é atualmente implementada. A primeira limitação diz respeito ao que tange a escalabilidade da versão atual. Embora apresentando desempenho satisfatório na avaliação, ocorre que o sistema não foi concebido de maneira a atender requisições de forma distribuída. Uma implementação distribuída poderia ajudar, por exemplo, a atender maiores números de solicitações de análise.

Outra limitação identificada diz respeito à geração do modelo preditio. A construção deste modelo é importante para que a ferramenta possa gerar classificações mais precisas. Para que esta etapa seja realizada é necessário obter uma base de treinamento. Porém, atualmente conta-se apenas com o Google Cluster Trace para este fim, conforme aponta a pesquisa na literatura. A indisponibilidade de mais conjuntos de treinamentos pode impactar negativamente na precisão da ferramenta.

Por fim, existe também a limitação no que tange os testes de precisão do método. Novamente, ressalta-se que durante as etapas de testes, dispôs-se apenas do conjunto de dados do Google. Embora utilizados métodos estatísticos para aproximar a qualidade e precisão do modelo gerado, reconhece-se que o ideal é a utilização de um conjunto de dados separados para teste. Porém, a indisponibilidade destes, visto que tratam-se de dados privados, terminaram por afastar esta abordagem de análise.

9 CONCLUSÃO

A crescente oferta de serviços baseados em nuvem até o período atual, ano de 2018, tem atraído cada vez mais usuários para este paradigma. Os principais fatores que auxiliam na escolha por adotar um serviço de nuvem tradicionalmente são a elasticidade, redução de custos e alta disponibilidade proporcionado por este modelo. Por outro lado, é sabido que existem riscos associados ao se adotar uma solução deste tipo, fator que dificulta a adoção plena desta tecnologia.

No decorrer da pesquisa realizada, identificaram-se os ataques de *side channel* como sendo uma ameaça de grande potencial à segurança e privacidade dos usuários legítimos de serviços *cloud*, possuindo já diversas demonstrações práticas na literatura. Os ASC baseiam-se em uma brecha na arquitetura de processadores da atualidade, onde o *cache* é compartilhado. Desta forma, nenhum mecanismo de segurança dos sistemas atuais é diretamente burlado, fato que torna a identificação desta prática uma tarefa não-trivial. Esta característica, associada à relativa facilidade em se realizar um ASC reforça a importância em se desenvolver novas medidas de contenção.

Como abordagem para o problema, este trabalho visou explorar a detecção de lançamentos anormais de máquinas virtuais em ambientes de *cloud* IaaS no intuito de minorar as chances destes ataques. A decisão desta metodologia foi norteada pelo fato de que o estabelecimento de coalocação é uma etapa frequentemente essencial para o sucesso desta prática maliciosa e possui características passíveis de serem identificadas, conforme demonstrado anteriormente por SecludIT (2018).

As principais contribuições deste trabalho foram:

- Identificação dos métodos atuais para prevenção de ataques de *side channel*, juntamente aos seus potenciais de aplicação;
- Elaboração de um método de prevenção de ataques de *side channel* adaptável ao ambiente IaaS onde é empregado, seguida de sua validação.

Embora havendo a existência de limitações, testes realizados para a validação da metodologia desenvolvida demonstram uma qualidade satisfatória a um custo não

elevado de recursos computacionais. É possível concluir que a análise de comportamento de lançamento das máquinas virtuais pode de fato auxilar na deteção de ASCs, evitando tentativas de ataques de co-alocação. Estas características fazem com que a metodologia seja aplicável em ambientes IaaS no intuito de minimizar a vulnerabilidade apresentada.

Por fim, devem ser ressaltados os trabalhos futuros. Primeiramente, sugere-se a aplicação da metodologia apresentada a fim de detectar ataques de *side channel* em novos conjuntos de dados públicos, conforme estes se tornarem disponíveis. Conforme afirmado ao final do capítulo 7, como prova de conceito, a ação tomada pelo módulo de reação consiste de uma notificação ao administrador. Desta forma, é possível implementar novas respostas automatizadas como, por exemplo, bloqueios de contas ou encerramento de máquinas virtuais.

A geração de ruído também possui potencial de investigação e expansão em novos trabalhos, buscando-se a diminuição do *overhead* identificado. Por fim, existe a possibilidade de estudo de novos parâmetros e padrões comportamentais e a respectiva verificação da forma com que estas novas *features* podem vir a interagir com o modelo preditor.

REFERÊNCIAS

AINAPURE, B. S.; SHAH, D.; RAO, A. A. Understanding Perception of Cache-Based Side-Channel Attack on Cloud Environment. In: PROGRESS IN INTELLIGENT COMPUTING TECHNIQUES: THEORY, PRACTICE, AND APPLICATIONS, Singapore. **Anais...** Springer Singapore, 2018. p.9–21.

ALJAHDALI, H. et al. Multi-tenancy in Cloud Computing. In: IEEE 8TH INTERNATIONAL SYMPOSIUM ON SERVICE ORIENTED SYSTEM ENGINEERING, 2014. **Anais...** [S.l.: s.n.], 2014. p.344–351.

ANWAR, S. et al. Cross-VM cache-based side channel attacks and proposed prevention mechanisms: a survey. **Journal of Network and Computer Applications**, [S.l.], v.93, p.259 – 279, 2017.

APECECHEA, G. I. et al. Fine grain Cross-VM Attacks on Xen and VMware are possible! **IACR Cryptology ePrint Archive**, [S.l.], v.2014, p.248, 2014.

AZAB, M.; ELTOWEISSY, M. MIGRATE: towards a lightweight moving-target defense against cloud side-channels. In: IEEE SECURITY AND PRIVACY WORKSHOPS (SPW), 2016. **Anais...** [S.l.: s.n.], 2016. p.96–103.

BAIG, M. B. et al. CloudFlow: cloud-wide policy enforcement using fast vm introspection. In: IEEE INTERNATIONAL CONFERENCE ON CLOUD ENGINEERING, 2014. **Anais...** [S.l.: s.n.], 2014. p.159–164.

BAMAKAN, S. M. H. et al. An effective intrusion detection framework based on MCLP/SVM optimized by time-varying chaos particle swarm optimization. **Neurocomputing**, [S.l.], v.199, p.90 – 102, 2016.

BAZM, M.-M. et al. Side-Channels Beyond the Cloud Edge : new isolation threats and solutions. In: IEEE International Conference on Cyber Security in Networking (CSNet) 2017, Rio de Janeiro, Brazil. **Anais...** [S.l.: s.n.], 2017. (1st Cyber Security in Networking Conference (CSNet'17)).

CARON, E.; CORNABAS, J. R. Improving users' isolation in iaas: virtual machine

placement with security constraints. In: CLOUD COMPUTING (CLOUD), 2014 IEEE 7TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.64–71.

CHANG, V.; RAMACHANDRAN, M. Towards Achieving Data Security with the Cloud Computing Adoption Framework. **IEEE Transactions on Services Computing**, [S.l.], v.9, n.1, p.138–151, Jan 2016.

CHIAPPETTA, M.; SAVAS, E.; YILMAZ, C. Real time detection of cache-based side-channel attacks using hardware performance counters. **Applied Soft Computing**, [S.l.], v.49, p.1162 – 1174, 2016.

Cloud Security Alliance. **The Treacherous 12 - Cloud Computing Top Threats in 2016**. Disponível em https://downloads.cloudsecurityalliance.org/assets/research/top-threats/Treacherous-12-Cloud-Computing-Top-Threats.pdf, acesso em janeiro de 2018.

Cloud Security Alliance. **Security Guidance for Critical Areas of Focus in Cloud Computing V4.0**. Disponível em https://downloads.cloudsecurityalliance.org/assets/research/security-guidance/security-guidance-v4-FINAL-feb27-18.pdf, acesso em janeiro de 2018.

COPPOLINO, L. et al. Cloud security: emerging threats and current solutions. **Computers and Electrical Engineering**, [S.l.], v.59, p.126 – 140, 2017.

DUAN, Y. et al. Everything as a Service (XaaS) on the Cloud: origins, current and future trends. In: IEEE 8TH INTERNATIONAL CONFERENCE ON CLOUD COMPUTING, 2015. **Anais...** [S.l.: s.n.], 2015. p.621–628.

Eberly College of Science - The Pennsylvanya State University. **STAT504 - Analysis of Discrete Data**. Disponível em https://onlinecourses.science.psu.edu/stat504/node/149, acesso em janeiro/2018.

EESA, A. S.; ORMAN, Z.; BRIFCANI, A. M. A. A novel feature-selection approach based on the cuttlefish optimization algorithm for intrusion detection systems. **Expert Systems with Applications**, [S.l.], v.42, n.5, p.2670 – 2679, 2015.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. 6th.ed. [S.l.]: Pearson, 2010.

ENDER, P. B. **Applied Categorical and Nonnormal Data Analysis**. University of California, disponível em http://www.philender.com/courses/categorical/notes3/logit1.html, acesso em janeiro de 2018.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. Tese (Doutorado em Ciência da Computação) — University of California, Irvine. Capítulo 5 - Representational State Transfer (REST). Disponível em https://www.ics.uci.edu/ fielding/pubs/dissertation/rest-arch-style.htm, acesso em fevereiro de 2018.

GARRISON, G.; KIM, S.; WAKEFIELD, R. L. Success Factors for Deploying Cloud Computing. **Commun. ACM**, New York, NY, USA, v.55, n.9, p.62–68, Sept. 2012.

GODFREY, M. M.; ZULKERNINE, M. Preventing cache-based side-channel attacks in a cloud environment. **IEEE transactions on cloud computing**, [S.l.], v.2, n.4, p.395–408, 2014.

GODFREY, M.; ZULKERNINE, M. A Server-Side Solution to Cache-Based Side-Channel Attacks in the Cloud. In: IEEE SIXTH INTERNATIONAL CONFERENCE ON CLOUD COMPUTING, 2013. **Anais...** [S.l.: s.n.], 2013. p.163–170.

GONZALEZ, N. et al. A quantitative analysis of current security concerns and solutions for cloud computing. **Journal of Cloud Computing: Advances, Systems and Applications**, [S.l.], v.1, n.1, p.11, Jul 2012.

GULMEZOGLU, B. et al. Cross-VM Cache Attacks on AES. **IEEE Transactions on Multi-Scale Computing Systems**, [S.l.], v.2, n.3, p.211–222, July 2016.

INCI, M. S. et al. Seriously, get off my cloud! Cross-VM RSA Key Recovery in a Public Cloud. **IACR Cryptology ePrint Archive**, [S.l.], v.2015, p.898, 2015.

IQBAL, S. et al. On cloud security attacks: a taxonomy and intrusion detection and prevention as a service. **Journal of Network and Computer Applications**, [S.l.], v.74, p.98 – 120, 2016.

IRAZOQUI, G. et al. Wait a minute! A fast, Cross-VM attack on AES. In: INTERNATIONAL WORKSHOP ON RECENT ADVANCES IN INTRUSION DETECTION. **Anais...** [S.l.: s.n.], 2014. p.299–319.

IRAZOQUI, G. et al. Know thy neighbor: crypto library detection in cloud. **Proceedings on Privacy Enhancing Technologies**, [S.l.], v.2015, n.1, p.25–40, 2015.

JAMES, G. et al. **An introduction to statistical learning**. [S.l.]: Springer, 2013. v.112.

JANSEN, W. A. Cloud hooks: security and privacy issues in cloud computing. In: SYSTEM SCIENCES (HICSS), 2011 44TH HAWAII INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.1–10.

KALLONIATIS, C. et al. Towards the design of secure and privacy-oriented information systems in the cloud: identifying the major concepts. **Computer Standards and Interfaces**, [S.l.], v.36, n.4, p.759 – 775, 2014. Security in Information Systems: Advances and new Challenges.

KHAN, M. A. A survey of security issues for cloud computing. **Journal of Network and Computer Applications**, [S.l.], v.71, p.11 – 29, 2016.

KHORSHED, M. T.; ALI, A. S.; WASIMI, S. A. A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing. **Future Generation Computer Systems**, [S.l.], v.28, n.6, p.833 – 851, 2012. Including Special sections SS: Volunteer Computing and Desktop Grids and SS: Mobile Ubiquitous Computing.

KONG, J. et al. Architecting against Software Cache-Based Side-Channel Attacks. **IEEE Transactions on Computers**, [S.l.], v.62, n.7, p.1276–1288, July 2013.

KUMAR, P.; WASAN, S. K. Comparative study of k-means, pam and rough k-means algorithms using cancer datasets. In: CSIT: 2009 INTERNATIONAL SYMPOSIUM ON COMPUTING, COMMUNICATION, AND CONTROL (ISCCC 2009). **Proceedings...** [S.l.: s.n.], 2011. v.1, p.136–140.

LIU, F. et al. Last-Level Cache Side-Channel Attacks are Practical. In: IEEE SYMPOSIUM ON SECURITY AND PRIVACY, 2015. **Anais...** [S.l.: s.n.], 2015. p.605–622.

MACQUEEN, J. et al. Some methods for classification and analysis of multivariate observations. In: BERKELEY SYMPOSIUM ON MATHEMATICAL STATISTICS AND PROBABILITY. **Proceedings...** [S.l.: s.n.], 1967. v.1, n.14, p.281–297.

MISHRA, P. et al. Intrusion detection techniques in cloud environment: a survey. **Journal of Network and Computer Applications**, [S.l.], v.77, p.18 – 47, 2017.

MOHAN, C. et al. ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. **ACM Transactions on Database Systems (TODS)**, [S.l.], v.17, n.1, p.94–162, 1992.

MOON, S.-J.; SEKAR, V.; REITER, M. K. Nomad: mitigating arbitrary cloud side channels via provider-assisted migration. In: ND ACM SIGSAC CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, 22., New York, NY, USA. **Proceedings...** ACM, 2015. p.1595–1606. (CCS '15).

MOURATIDIS, H. et al. A framework to support selection of cloud providers based on security and privacy requirements. **Journal of Systems and Software**, [S.l.], v.86, n.9, p.2276 – 2293, 2013.

NG, A. **CS229 Lecture Notes - Support Vector Machines**. Disponível em http://cs229.stanford.edu/notes/cs229-notes3.pdf, acesso em janeiro de 2018.

OPENSTACK. Acesso em junho de 2017.

PITROPAKIS, N. et al. It's All in the Cloud: reviewing cloud security. In: IEEE 10TH INTERNATIONAL CONFERENCE ON UBIQUITOUS INTELLIGENCE AND COMPUTING AND 2013 IEEE 10TH INTERNATIONAL CONFERENCE ON AUTONOMIC AND TRUSTED COMPUTING, 2013. **Anais...** [S.l.: s.n.], 2013. p.355–362.

RECALCATTI, E. **Uma plataforma virtual para experimentos de ataques de canal lateral em sistemas baseados em rede-em-chip**. 2015. Dissertação (Mestrado em Ciência da Computação) — UNIVERSIDADE DO VALE DO ITAJAÍ.

REISS, C.; WILKES, J.; HELLERSTEIN, J. **Google cluster-usage traces**: format and schema. Disponível em https://drive.google.com/file/d/0B5g07T-gRDg9Z0lsSTEtTWtpOW8/view, acesso em janeiro de 2018.

SECLUDIT. **How to detect side-channel attacks in cloud infrastructures**. Disponível em https://elastic-security.com/2013/09/11/how-to-detect-side-channel-attacks-in-cloud-infrastructures, acesso em janeiro de 2018.

SHAMIR, A.; TROMER, E. Acoustic cryptanalysis: on nosy people and noisy machines [eb/ol]. **Disponível em http://www.wisdom.weizmann.ac.il/˜tromer/acoustic, acesso em dezembro de 2017**, [S.l.], 2004.

SHIKIDA, M. et al. An Evaluation of Private Cloud System for Desktop Environments. In: ANNUAL ACM SIGUCCS CONFERENCE ON USER SERVICES, 40., New York, NY, USA. **Proceedings...** ACM, 2012. p.131–134. (SIGUCCS '12).

SIMOVICI, D. A. **The PAM Clustering Algorithm**. Department of Computer Science, University of Massachusetts Boston, Disponível em https://www.cs.umb.edu/cs738/pam1.pdf, acesso em março de 2018.

SINGH, S.; JEONG, Y.-S.; PARK, J. H. A survey on cloud computing security: issues, threats, and solutions. **Journal of Network and Computer Applications**, [S.l.], v.75, p.200 – 222, 2016.

SUBASHINI, S.; KAVITHA, V. A survey on security issues in service delivery models of cloud computing. **Journal of Network and Computer Applications**, [S.l.], v.34, n.1, p.1 – 11, 2011.

XAVIER, T. A.; REJIMOAN, R. Survey on various resource allocation strategies in cloud. In: INTERNATIONAL CONFERENCE ON CIRCUIT, POWER AND COMPUTING TECHNOLOGIES (ICCPCT), 2016. **Anais...** [S.l.: s.n.], 2016. p.1–4.

XIAO, Z.; XIAO, Y. Security and Privacy in Cloud Computing. **IEEE Communications Surveys Tutorials**, [S.l.], v.15, n.2, p.843–859, 2013.

YAROM, Y.; FALKNER, K. FLUSH+ RELOAD: a high resolution, low noise, l3 cache side-channel attack. In: USENIX SECURITY SYMPOSIUM. **Anais...** [S.l.: s.n.], 2014. p.719–732.

YAROM, Y.; FALKNER, K. FLUSH+RELOAD: a high resolution, low noise, l3 cache side-channel attack. In: USENIX SECURITY SYMPOSIUM (USENIX SECURITY 14), 23., San Diego, CA. **Anais...** USENIX Association, 2014. p.719–732.

ZHANG, R. et al. On Mitigating the Risk of Cross-VM Covert Channels in a Public Cloud. **IEEE Transactions on Parallel and Distributed Systems**, [S.l.], v.26, n.8, p.2327–2339, Aug 2015.

ZHANG, T.; ZHANG, Y.; LEE, R. B. Cloudradar: a real-time side-channel attack detection system in clouds. In: INTERNATIONAL SYMPOSIUM ON RESEARCH IN ATTACKS, INTRUSIONS, AND DEFENSES. **Anais...** [S.l.: s.n.], 2016. p.118–140.

ZHANG, Y. et al. Cross-VM Side Channels and Their Use to Extract Private Keys. In: ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, 2012., New York, NY, USA. **Proceedings...** ACM, 2012. p.305–316. (CCS '12).

ZHANG, Y. et al. Incentive Compatible Moving Target Defense against VM-Colocation Attacks in Clouds. In: SEC. **Anais...** [S.l.: s.n.], 2012. p.388–399.