UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

Dhileane de Andrade Rodrigues

Mecanismo de Tolerância a Falhas para o EICIDS: Sistema de detecção de intrusão elástico e interno baseado em nuvem

São Luís - MA
2014

Dhileane de Andrade Rodrigues

Mecanismo de Tolerância a Falhas para o EICIDS: Sistema de detecção de intrusão elástico e interno baseado em nuvem

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como requisito parcial para a obtenção do grau de MESTRE em Engenharia de Eletricidade.

Orientador: Zair Abdelouahab

Doutor em Ciências da Computação – UFMA

São Luís - MA

2014

Rodrigues, Dhileane de Andrade

Mecanismo de Tolerância a Falhas para o EICIDS: Sistema de detecção de intrusão elástico e interno baseado em nuvem / Dhileane de Andrade Rodrigues. – São Luís - MA, 2014.

110 f.

Orientador: Zair Abdelouahab.

Dissertação (Mestrado) – Universidade Federal do Maranhão, Programa de Pós-Graduação em Engenharia de Eletricidade. São Luís - MA, 2014.

1. Computação em nuvem - elasticidade. 2. Sistema de Detecção de Intrusos. I. Título. I. Abdelouahab, Zair, orient. II. Título.

CDU 621.31:004

MECANISMO DE TOLERÂNCIA A FALHAS PARA O EICIDS: SISTEMA DE DETECÇÃO DE INTRUSÃO ELÁSTICO E INTERNO BASEADO EM NUVEM

Dhileane de Andrade Rodrigues

Dissertação aprovada em 16 de junho de 2014.

Prof. Zair Abdelouahab, Ph.D.
(Orientador)

Prof. Ricardo de Andrade Lira Rabêlo, Dr.
(Membro da Banca Examinadora)

Prof. Samyr Beliche Vale, Dr.
(Membro da Banca Examinadora)

À minha mãe e a meu pai, que me ajudaram nessa caminhada.

Resumo

A computação em nuvem está cada vez mais em evidencia no mundo pela sua forma de disponibilizar recursos e sua elasticidade. Tais características tornam esse ambiente um atrativo para a intrusão e consequentemente vulnerável a ataques. Portanto, há uma necessidade de ferramentas adequadas para prover a segurança na nuvem. Entre as diversas ferramentas que atualmente foram propostas para manter a segurança na nuvem destaca-se o Sistema de Detecção de Intrusão (SDI), uma vez que seu objetivo é identificar indivíduos que tentam usar um sistema de forma não autorizado ou abusivo, ou seja, abusa dos privilégios concedidos aos mesmos. Embora inúmeras pesquisas direcionadas à área de detecção de intrusão no ambiente de CN, muitos são os problemas enfrentados por essa técnica, tais como: a elasticidade dos componentes, a autoproteção, a disponibilidade dos seus serviços, a auto resistente sem nenhum ponto único de falha e proporcionar ações de resposta automática com base no conjunto de políticas. Um SDI para realizar corretamente sua função na nuvem, deve possuir a capacidade de aumentar ou diminuir rapidamente a quantidade de sensores, de acordo com a elasticidade da nuvem. Além de prover a capacidade de expandir-se o SDI deve garantir a confiabilidade e disponibilidade de suas próprias aplicações. Assim sendo, um SDI aplicado na nuvem deve dar continuidade aos seus serviços mesmo diante de falhas, principalmente falhas oriundas de ações maliciosas. Esta dissertação propõe um mecanismo de tolerância a falhas para o *Elastic and Internal Cloud-based Intrusion Detection System* (EICIDS), um sistema de detecção de intrusão baseado em virtualização e dinâmico. O mecanismo utiliza algumas técnicas: o monitoramento do sistema, emissão de echo e a replicação. O mecanismo possui um grupo de componentes que monitoram o sistema para coletar informações relacionadas aos comportamento do próprio SDI e das *Virtual machines* (VMs) para prover a recuperação adequada. Usando a informação que é coletada, o sistema pode: descobrir as VMs inativas; descobrir VMs com ações maliciosas e descobrir aplicações que estão sendo usadas de forma indevidas. A replicação ocorre no momento em que não existe comunicação entre os componentes do SDI que se localizam nos nodes e o elemento central. Além disso, esse monitoramento também permite realizar outras importantes tarefas tais como: emissão de sinal para todas as VMs se uma ação maliciosa for detectada, bloquei de usuário mal intencionado, e monitoramento do elemento central, para garantir a segurança do próprio SDI (*Self protection*). A implementação da arquitetura proposta e os testes realizados demostram a viabilidade da solução.

Palavras-chaves: Computação em Nuvem, Detecção de Intrusão, Tolerância a Falhas, Virtualização.

Abstract

Cloud computing is increasingly in evidence in the world for its elasticity, in providing resources. The characteristics of cloud computing make the environment attractive for intrusion and therefore vulnerable to attack. Therefore, there is a need for adequate tools to provide security in the cloud. A tool that is currently proposed to maintain security in the cloud is the Intrusion Detection Systems (IDS), since its objective is to identify individuals who attempt unauthorized use an or abusive the system with its privileges . Although numerous studies aimed at detection area intrusion into the cloud computing environment, there are many problems faced in area, such as: the elasticity of the components, self-protection, the availability of the services, self-resilient study with no single point of failure and automatic response actions based on the set of policies. To properly perform the function in the cloud, the IDS should have the ability to quickly increase or decrease the number of sensors, according to the elasticity of the cloud. Besides providing the ability to expand, the SDI should ensure reliability and availability of their own applications. An SDI in the cloud should continue its services even before failures, especially failures arising from malicious actions. This dissertation proposes a mechanism for fault tolerance and Internal Elastic Cloud-based Intrusion Detection System (EICIDS), an intrusion detection system based on dynamic virtualization. The mechanism uses some technique: monitoring system, echo and replication. The mechanism has a group of components that monitor the system to collect the IDS behavior and information of the Virtual machines (VMs) to provide adequate recovery. Using the information that is collected, the system can: find the inactive VMs; discover VMs with malicious actions and discover applications being used in an improper form. Replication occurs at the moment no communication exists between the components of SDI nodes that are located in the element and Central. In addition, this monitoring allows also perform other important tasks such as signal output for all VMs if a malicious action is detected, block malicious user, and monitoring of the central element, to ensure the safety of SDI itself (self-protection). The implementation of the proposed architecture and testing demonstrate the feasibility of the solution.

Keywords: Cloud Computing, Intrusion Detection, Fault Tolerance, Virtualization.

Agradecimentos

Ter a quem agradecer é uma graça divina, pois isto significa que ao longo dessa caminhada não estive sozinho e que comigo estiveram todos aqueles que de alguma forma desejaram esta vitória.

Deus se faz presente em todos os momentos da minha vida, todas as grandes conquista que eu obtive foram graças a ele e, portanto, agradeço a Deus hoje e sempre pelas vitórias a mim concedidas.

Ao falar em Deus, lembro em especial da minha avó que sempre me quis o bem, e agora se encontra com o Pai. Saudades eterna.

A meu pai (José Raimundo Rodrigues) e a minha mãe (Deusileide Pereira de Andrade) por todo o carinho e apoio que sempre me deram ao longo de minha caminhada, obrigada porque sei que com você posso contar Sempre.

Ao meu orientador Prof. Zair Abdelouahab pela confiança e credibilidade depositadas em mim desde o início deste trabalho. Agradeço por seus ensinamentos, orientações sempre que eu precisava. Agradeço também por sua paciência, compreensão nos momentos mais difíceis, e não desistir de me orientar, por acreditar que eu seria capaz de chegar até o fim. A este sempre terei respeito e admiração.

A Josenilson que sempre me deu boas dicas assumindo o papel de meu co-orientador.

A Profª. Dra Christiane Lima, pelo fundamental auxílio na co-orientação desta pesquisa, por sua paciência, dedicação, experiência partilhada, e especialmente pela revisão e correção desta dissertação; A esta também terei um grande respeito e admiração.

Aos meus companheiros de longa caminhada. Em especial aos que compõem o LABSAC (Luiz, Marcos Sá, Willian, Higo, Jhonatan, Mário, Leonardo, Jean, Steve, Bruno, Cláudio, Renato, Carol e Wagner). Muito obrigado por toda ajuda e companheirismo durante esta etapa.

Ao Programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão e a todos os professores do programa pelo aprendizado propiciado. Ao Neto e Alcides, pelo bom atendimento às solicitações acadêmicas.

À CNPQ, pelo suporte financeiro durante o mestrado. A este órgão, exprimo meu reconhecimento.

Aos meus valiosos amigos que ao longo dessa caminha foram meus braços, pernas, meu tudo nas horas alegres e tristes. Em especial Antônio Filho, Alcilene Dalilia e Laise Nayra.

Enfim muito obrigada a todos que direta ou indiretamente contribuíram para a realização deste trabalho.

"Guardo no coração as tuas palavras, para não pecar contra ti."

Salmo 119.11

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AFTRC	*Adaptive Fault Tolerance in Real Time Cloud Computing*.
BDUL	Base de Dados de Usuários Legítimos da Nuvem.
CIDS	*A Framework for Intrusion Detection in Cloud Systems*.
CN	Computação em Nuvem.
CTF	Componentes de Tolerância a Falhas.
DoS	*Denial of Service*.
EICIDS	*Elastic and Internal Cloud-based Intrusion Detection System*.
FTWF	*Fault tolerant workflow scheduling based on replication and resubmission of tasks in Cloud Computing*.
GCCIDS	*Intrusion Detection for Grid and Cloud Computing*.
GOS	*Gest Operating System*.
HA-CIDS	*A Hierarchical and Autonomous IDS for Cloud Systems*.
HIDS	*Host-based Intrusion Detection*.
HOS	*Host Operating System*.
IaaS	*Infrastructure as a Service*.
IDMEF	*Intrusion Detection Message Exchange Format*.
IDSaaS	*Intrusion Detection System as a Service in Public*.
LLFT	*Fault tolerance middleware for cloud computing*.
MMV	*Monitor de Máquina Virtual*.

NIDIA	*Network Intrusion Detection System based on Inteligent Agents*.
NIDS	*Network-based Intrusion Detection*.
NIST	*National Institute of Standards and Technology*.
PaaS	*Platform as a Service*.
SaaS	*Software as a Servic*.
SDI	Sistema de Detecção de Intrusão.
SO	Sistema Operacional.
TI	Tecnologia da Informação.
UCP	Unidade Central de Processamento.
UFMA	Universidade Federal do Maranhão.
VFT	*VFT: A virtualization and fault tolerance approach for cloud computing*.
VMs	*Virtual machines*.

Sumário

1 Introdução

A Computação em Nuvem (CN) é a denominação genérica para um conjunto de recursos disponíveis e escaláveis que ficam à disposição dos usuários para o acesso a hora que desejar e de qualquer lugar independentemente da tecnologia utilizada. Os recursos disponíveis vão desde de uma simples aplicação ou toda infraestrutura (redes, servidores e armazenamento), onde a alocação deverá ser de forma rápida e de fácil gestão [37]. A escalabilidade é uma característica essencial da computação em nuvem e é obtido através de virtualização de servidores [50].

A CN veio para revolucionar a forma de prover Tecnologia da Informação (TI), liberando os usuários de qualquer complicação com infraestrutura e fornecendo os recursos computacionais a medida que se faz necessários, ou seja, abstrai toda infraestrutura e fornece serviços sob demanda. Mas devido a essa característica de abstração da infraestrutura da CN, surge também uma enorme preocupação quanto à segurança. A falta de segurança é um dos fatores que dificulta a adoção da computação em nuvem [50]. Além dessas características, a CN possibilita a redução de custos e o aumento da flexibilidade na gestão do ambiente. No entanto, as organizações ainda caminham a passos lentos na adesão dessa nova abordagem, que também pode ser chamada de *Cloud Computing* [56].

Essa preocupação com a segurança é fruto do crescimento desordenado dos incidentes na vulnerabilidade[1] da nuvem, que promove cada vez mais a eficácia dos ataques. A pesquisa realizada em [12] mostra um crescimento dos incidentes na nuvem ao longo dos últimos 5 anos, onde pode-se observar um aumento alarmante. Na Figura 1.1 pode ser observado o crescimento dos incidentes na vulnerabilidade da nuvem ao longo dos últimos 4 anos.

Os resultados de pesquisas, ao longo dos anos, tornam evidente o enorme crescimento de incidentes na nuvem. O ano de 2009 encontra-se um nível de incidentes um pouco abaixo de 40%, e em 2011 esses incidentes duplicaram [12]. Esse crescimento alarmante dos incidentes na nuvem segundo o autor em [3], deve-se ao crescimento da implantação dos modelos dessa nova abordagem.

[1]Vulnerabilidade é definida como uma condição que, quando explorada por um atacante, pode resultar em uma violação de segurança.

Figura 1.1: Frequência de Incidentes de Vulnerabilidade da Nuvem. Adaptado de [12]

Com base nas informações obtidas por [12] sobre o aumento dos incidentes na nuvem com o passar dos anos, constatou-se que há uma necessidade de ferramentas que auxiliem as organizações a manter a segurança da CN e de seus componentes. Entre as ferramentas destaca-se o SDI, ferramenta que vem sendo utilizada por inúmeras pesquisas e que apresentam resultados promissores para a segurança da computação em nuvem.

1.1 Motivação e Justificativa

A Computação em nuvem é uma abordagem bastante utilizada nos últimos anos, ela provê a liberação de recursos sob demanda e fornece seus serviços pela internet. A forma de provimento de recursos da nuvem é bastante promissora, já que os serviços são liberados aos usuários a medida que se faz necessário, com isso o usuário não terá preocupação com a instalação de programas e nem terá problemas com o armazenamento (falta ou desperdício da capacidade de armazenamento). Com a computação em nuvem, as aplicações, bem como arquivos e outros dados relacionados, não precisam estar instalados no computador do usuário. Estas aplicações passam a ficar disponíveis nas nuvens, isto é, serão liberados mediante a necessidade através da internet. Toda a infraestrutura fica sob responsabilidade do fornecedor da aplicação, caberá a ele todas as tarefas referentes a manutenção, atualização, armazenamento, etc. Com essa caracteristica oferecida pela computação em nuvem o usuário

fica livre de qualquer preocupação com os aspectos citados, terá apenas que solicitar o serviço e utilizá-lo.

A computação em nuvem, por sua forma de centralizar as aplicações e armazenar dados em provedores distantes dos usuários finais, traz uma grande preocupação no que diz respeito à segurança e à privacidade. Um usuário ao solicitar os serviços da nuvem, estará entregando todos seus dados e informações importantes ao provedores responsáveis, o que gera muitas vezes dúvidas quanto a segurança, dado uma sensação aos usuários de vulnerabilidade, já que seus dados e informações ficaram alojados na nuvem.

Para assegurar os usuários o provedor de nuvem deve garantir a segurança e a privacidade dos dados e informações. Os problemas de segurança abordados na nuvem podem ser previamente identificados com o uso adequado de ferramentas que possibilitam a prevenção ou até mesmo a recuperação de uma falha ou defeito gerado no sistema de forma maliciosa ou não.

A justificativa para a realização deste trabalho é propor um mecanismo de tolerância a falhas para sistemas de detecção de intrusão no ambiente de computação em nuvem e que possa, de forma eficiente, identificar e recuperar-se de ações maliciosas, sem provocar danos nos dados e informações dos usuários da nuvem.

1.2 Problemática

Na nuvem há uma abstração do ambiente tradicional, e as aplicações são entregues aos usuários por provedores de serviços que por sua vez utilizam tecnologias diferentes [63]. Essa nova forma de provisionar os recursos tecnológicos é o que proporciona tantas pesquisas voltadas para a segurança em computação em nuvem, ganhando destaque nesse meio inúmeras técnicas de tolerância a falhas podendo ser citados trabalhos como [64] [57] [25] [28].

Na Universidade Federal do Maranhão (UFMA), o projeto EICIDS proposto por [18] foi desenvolvido. O EICIDS é um sistema de detecção de intrusão para ambientes de computação em nuvem, que tem como base a técnica de virtualização, ele protege as máquinas virtuais que compõem a camada *Software as a Servic* (SaaS). O EICIDS é sistema de detecção de intrusão escalável e tem como importante característica a escalabilidade de seus componentes a medida que se faz necessários e a capacidade de gerar alerta ao administrador.

Segundo os autores em [9] e [52] um SDI precisa dar continuidade aos seus serviços mesmo em caso de falhas, principalmente aquelas causadas por usuários mal intencionados. Com isso, ele estará garantindo a confiabilidade e a disponibilidade à sua própria aplicação.

Em virtude disso, dado o contexto e a função ao qual o EICIDS foi aplicado, surgiu a necessidade de torná-lo tolerante a falhas, tirando a responsabilidade do administrador do sistema diante de uma possível falha. Este mecanismo deve ser capaz de detectar falhas acidentais e maliciosas.

A solução para o problema citado é aplicar a tolerância a falhas ao EICIDS de modo que ele possa atingir seu estado mais seguro e resistente a falhas. O mecanismo de tolerância a falhas proposto é um conjunto de técnicas e ações empregadas na tentativa de diminuir as falhas e evitar o máximo da intervenção do administrador da nuvem.

1.3 Objetivos Gerais e Específicos

Esta dissertação propõe um mecanismo de tolerância a falhas ao EICIDS de modo que ele possa atingir seu estado mais seguro e resistente a falhas; E evitar a propagação de ataques a nível de máquinas virtuais. Este mecanismo deve ser capaz de detectar falhas acidentais e maliciosas. Com as técnicas que serão utilizadas no decorrer da proposta, o EICIDS será capaz de realizar corretamente suas funções e garantir a continuidade de seus serviços mesmo em caso de falhas, principalmente falhas causadas por ações maliciosas dentro da nuvem. O mecanismo é composto por componentes que monitoram e recuperam o sistema quando uma falha é detectada. Por meio das informações que são coletadas é possível: Detectar quais os processos estão em execução; Verificar se há comunicação entre os elementos do SDI; Detectar se as VMs estão ativas; Identificar qual VM atacou e qual sofreu o ataque; Detectar qual VM deve ser replicada e Detectar se está acontecendo alguma interação não autorizada com o *hypervisor* (Essa interação só será considerada maliciosa se a VM que tentou acessar o hospedeiro não encontra-se cadastrada como uma VM que possui o acesso liberado).

Esta dissertação tem por objetivo específicos:

1. Apresentar um mecanismo de tolerância a falhas ao EICIDS;
2. Apresentar um protótipo do mecanismo de tolerância a falhas por meio de componentes sinal; componente de proteção do SDI e monitoramento do Admin (elemento principal do EICIDS, responsável pelo gerenciamento do processo de detecção de intrusão); e

Componente responsável pela replicação. Juntos irão monitorar e restaurar o sistema de forma automática.

3. Apresentar o resultado da avaliação, por meio de testes experimentais (eou simulações computacionais).

1.4 Estrutura da Dissertação

Esta dissertação encontra-se organizada em 6 capítulos, descritos a seguir:

No capítulo 1, os problemas atuais da adoção da computação em nuvem apresentados em uma breve introdução. Também os objetivos gerais e específicos e a organização desta dissertação são discutidos.

No capítulo 2, uma visão geral sobre computação na nuvem e sistemas de detecção de intrusão é apresentado, destacando-se classificação, característica e o projetos de sistemas de detecção para nuvem. Por fim, informações sobre mecanismo de tolerância a falhas são discutidas, junto com trabalhos relacionados, ou seja, tolerância a falhas para o ambiente de computação em nuvem.

No capítulo 3, o EICIDS é apresentado. A arquitetura, funcionamento e todos os componentes do EICIDS são mostrados.

No capítulo 4, o mecanismo proposto para prover tolerância a falhas ao EICIDS é apresentado. A arquitetura do mecanismo e seus componentes são detalhados.

No capítulo 5, detalhe da implementação da arquitetura do mecanismo e do EICIDS e os protótipos dos componentes do mecanismo são apresentados. Alguns resultados parciais obtidos nas simulações realizadas em laboratório são apresentados.

No capítulo 6, finalmente, as conclusões obtidas no desenvolvimento deste trabalho, possíveis trabalhos futuros e, por fim, as considerações finais desta dissertação serão apresentadas.

2 Fundamentação Teórica

Neste capítulo, os conceitos fundamentais das tecnologias que serviram de base para o desenvolvimento da proposta são apresentados. Os conceitos aqui abordados tiveram papel relevante no entendimento individual de cada tecnologia e na obtenção do conhecimento para melhor integrá-las. A integração destas tecnologias é o "pilar"principal do desenvolvimento dessa proposta.

2.1 Computação em Nuvem

A CN é definida pelo *National Institute of Standards and Technology* (NIST) [37], como um modelo que possibilita acesso, de modo conveniente e sob demanda, a um conjunto de recursos computacionais configuráveis (por exemplo: redes, servidores, armazenamento, aplicações e serviços). Esses recursos podem ser rapidamente adquiridos e liberados com mínimo de esforço gerencial ou interação com o provedor de serviços.

Segundo o autor em [55], a computação em nuvem surge com inúmeras facilidades, pois utiliza os recursos ociosos de computadores independentes, sem a preocupação com a localização física e sem investimentos em hardware. Tais facilidades ao longo dos anos estão seduzindo muitas organizações. As inúmeras definições encontradas na literatura científica levaram o autor em [55] á conclusão de que a computação em nuvem é um termo que descreve um ambiente de computação baseado em uma imensa rede de servidores, sejam estes virtuais ou físicos.

A busca por diminuição na perda de tempo com o gerenciamento de infraestrutura, a fim de dispor do tempo pedido com esse gerenciamento para expandir o negócio é o que tem contribuído para o avanço da CN. As empresas não abandonam seus primeiros investimentos em tecnologias, mas criam camadas de tecnologias mais recentes em cima das antigas segundo o autor em [55]. Embora a computação traga inúmeras vantagens como já visto até aqui, a mudança para essa nova abordagem não irá ocorrer tão rápido, isso porque as empresas ainda não colocaram total confiança em entregar todos seus dados a terceiros [55]. Essas preocupações com a segurança tornam-se um dos grande empecilhos para a total migração para a nuvem.

A CN utiliza a tecnologia da computação virtual [54]. Essa virtualização elimina os serviços e o tempo gasto com as infraestrutura dos recursos físicos (Hardware, rede). Essa abstração deixa toda localização real dos recursos na camada mais baixa da computação em nuvem *Infrastructure as a Service* (IaaS) (abordada na seção 2.1.1), sendo transparente aos usuários das demais camadas.

Esse novo paradigma propicia aos seus usuários a ilusão de recursos ilimitados para o uso. A CN disponibiliza serviços aos usuários a qualquer momento e lugar desde que se tenha acesso a uma infraestrutura de rede [7].

A CN revolucionou a forma de execução das tarefas de computação, pois veio distribuindo-as para *pools* de recursos virtuais. Os serviços fornecidos sob demanda pela computação em nuvem vieram para melhorar o desempenho, a capacidade de armazenamento e a liberação conforme o uso, sem desperdício e com rapidez, conforme [54].

É unânime o conhecimento de que na computação em nuvem existem duas questões muito importantes e que vem sendo motivo para inúmeras pesquisas nesse âmbito, são elas: confiabilidade e segurança. As grandes empresas preferem guardar seu dados sob sua cobertura, ou seja, cuidados pela própria empresa, mas com a computação em nuvem isso vem mudando, já que aderir a essa nova tecnologia reduz custos econômicos, e as empresas que aderem ganham mais tempo para seus negócios.

Com a análise da estrutura e do funcionamento de um *data center* local com um *data Center* em nuvem pode-se constatar que a utilização da nuvem diminui os custos. Em *data center* locais há a necessidade de investimentos em recursos, sendo esses fixos e não tem como desfazer a infraestrutura adquirida, pois a empresa terá seus próprios hardware e software. Isso sem levar em conta os gastos com licença de software e com pessoas qualificadas para operar o sistema. Os *data center* em nuvem possuem elasticidade do serviço que é a capacidade de disponibilizar e remover recursos em tempo de execução. Uma pesquisa feita pelo NIST, são listadas cinco características essenciais da CN: serviço sob demanda, amplo acesso à rede, pooling de recursos, rápida elasticidade e serviço medido, as quais serão apresentadas em detalhes na seção 2.1.2 [11].

2.1.1 Arquitetura da Computação em Nuvem

A arquitetura da CN é composta por camadas cada uma com seu nível de abstração e controle sobre as demais, por sua vez são compostas por servidores, equipamentos de rede e sistemas operacionais. Quanto mais alto nas camadas, maior o nível de abstração e menor o

controle. Por outro lado, quanto mais baixo, maior o controle e menor a abstração [8]. A Figura 2.1 ilustra como estão organizadas as camadas de uma infraestrutura da nuvem.

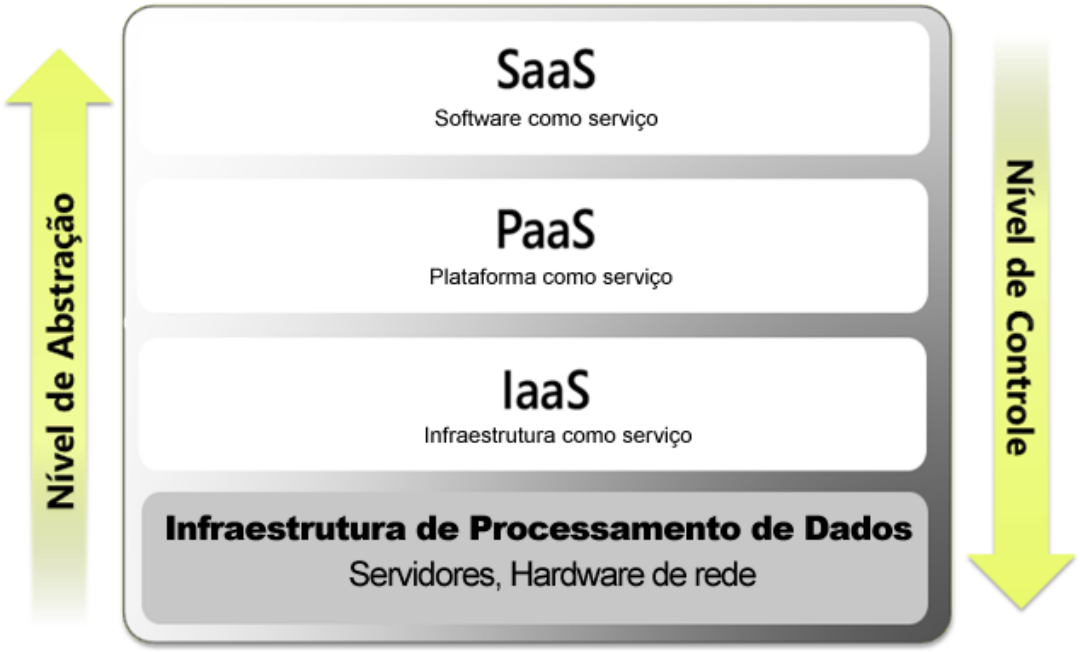

Figura 2.1: Camadas da infraestrutura física da computação em nuvem. Adaptado de [8]

A definição do NIST lista cinco características essenciais, três modelos de serviços (SaaS - Software como Serviço, *Platform as a Service* (PaaS) - Plataforma como Serviço e IaaS - Infraestrutura como Serviço) e quatro modelos de implantação (publica, privada, hibrida e comunitária), esses modelos definidos pelo NIST são vistos na Figura 2.2. Na seção seguinte serão descritos as características, os modelos de serviços e os modelos de implantações [19].

2.1.2 Característica Essências da Computação em Nuvem

As características essenciais da nuvem, definidas pelo NIST, tornam a computação uma opção de TI, conforme ilustrado na Figura 2.2.

Broad Network Access - Usuários terão acessos aos recursos que estão disponíveis na rede e para isso terão que simplesmente ter algum mecanismo que promovam o padrão utilizado por plataformas heterogêneas (laptops, telefones celulares e PDAs). Com isso, poderão acessar de qualquer lugar, a qualquer hora, de qualquer dispositivo, desde que tenham conexão (internet);

Elasticidade rápida - Usuários têm a ilusão de recursos ilimitados, podendo ser adquiridos em qualquer quantidade e a qualquer momento. Essa caracteristica da nuvem

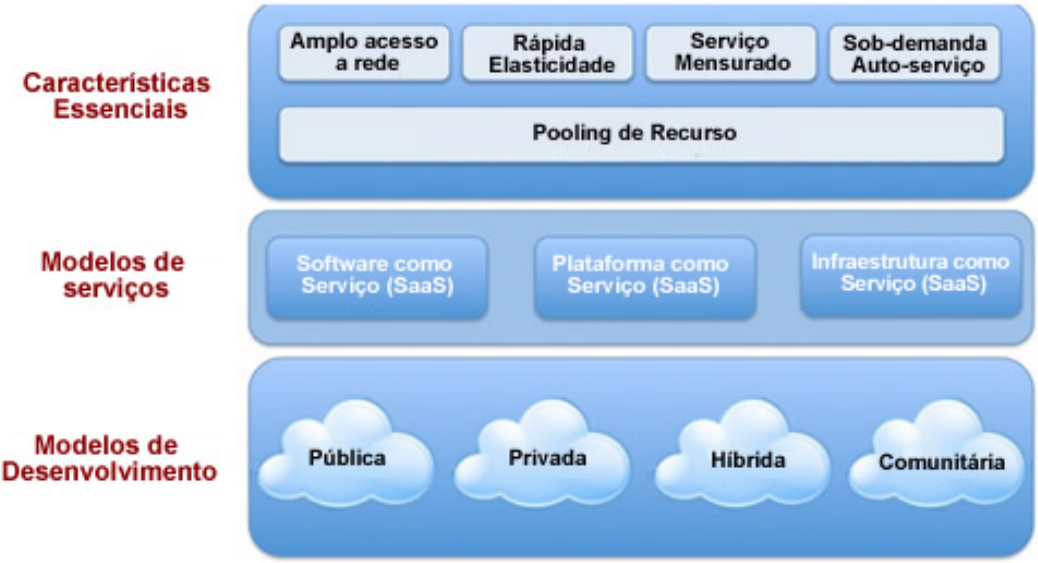

Figura 2.2: Modelo visual de definição de computação em nuvem do NIST. Adaptado de [19]

propociona a capacidade de prover e poupar os recursos da nuvem, de forma rápida, a medida que se faz necessário;

Serviço Medido - Uso de recursos pode ser monitorado, controlado e reportado, oferecendo transparência tanto para o provedor como também o consumidor do serviço utilizado.

On-Demand Self-Service - Recursos podem ser requisitados à medida que necessitados automaticamente e toda mudança no sentido de reconfigurações é feita de forma transparente para os usuários;

***Pooling* de recursos** - De acordo com a demanda do consumidor, recursos serão fornecidos (exemplos de recursos incluem o armazenamento, processamento, memória, largura de banda de rede e máquinas virtuais). Esse conjunto de recursos será liberado quando requisitado pelos usuários da nuvem, podendo ser em grande ou pequena escala, dependendo da necessidade.

A nuvem, de acordo com o seu uso, tem sua função, custo, e protocolos de rede específicos que serão utilizados segundo as necessidades dos usuários. Essas funções são de suma essência para a manutenção e controle da nuvem e é por meio dessas funções que o modelo de serviço é oferecido. Logo, o usuário paga pelo que usar, tem o que quer, na hora que quer e do jeito que seus negócios precisarem.

2.1.3 Modelos de Serviços da Computação em Nuvem

Em função do nível de abstração que oferece a Nuvem às organizações, na literatura, indentificam três categorias de plataformas de Computação em Nuvem, podem ser observados na Figura 2.3 [29]. O Saas , o PaaS e o IaaS descritos a seguir.

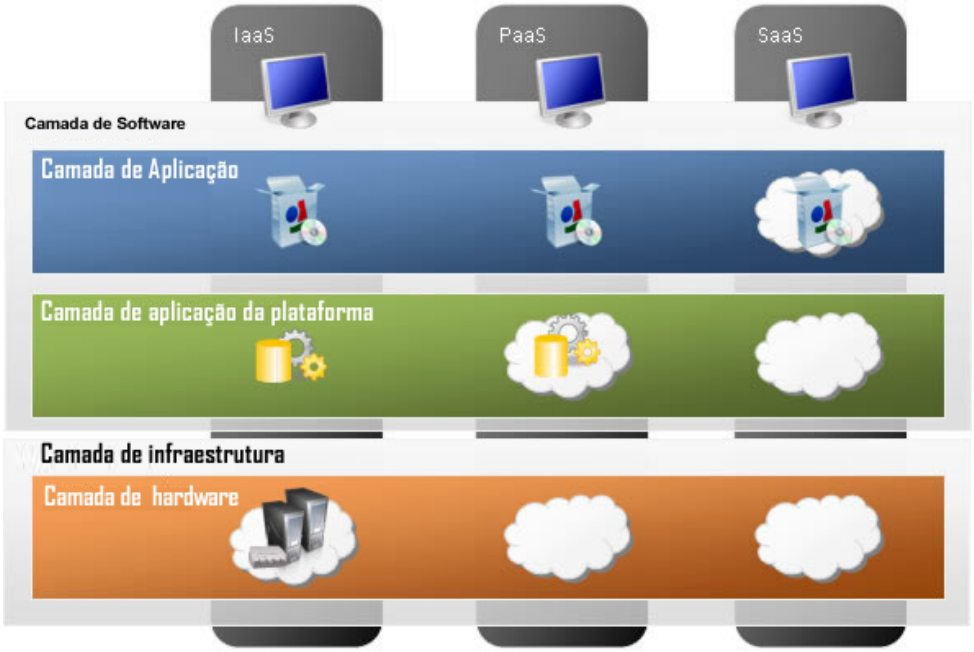

Figura 2.3: Modelos de serviços da computação em nuvem [29].

SaaS - nessa tipo de serviço da nuvem o usuário através da rede utiliza recursos fornecido pelo provedor a qualquer momento e de qualquer lugar sem se preocupar com a forma com esses serviços estão dispostos e organizados. Isso libera o usuário do gerenciamento total da infraestrutura, podendo utilizar somente o que lhe for necessário, pois é sob demanda e paga pelo que usar.

PaaS - plataformas de desenvolvimento de software e componentes são disponibilizados pela rede à medida que haja uma necessidade por conta do usuário desse serviço da nuvem. As licenças para cada linguagem proprietária utilizada ou ferramentas para o desenvolvimento são entregues pelo provedor. Com isso os usuários não precisam se preocupar com instalações de software e manutenção [24].

IaaS - O serviço fornecidos ao usuário é um conjunto de recursos computacionais, tais como: a infraestrutura de hardware (servidores, armazenamento, capacidade de processamento) e os software associados(Sistema operacionais, tecnologias de virtualização) [5] [24]. A camada de infraestrutura consiste nos ativos físicos (servidores, dispositivos de rede, discos de armazenamento, entre outros) e a base da nuvem.

É muito importante definir as responsabilidades de cada usuário, chamados de atores do modelo na nuvem. Esses atores podem ser visto na Figura 2.4. Por isso, existem as definições conforme os papeis desempenhados por cada usuário, a saber [59].

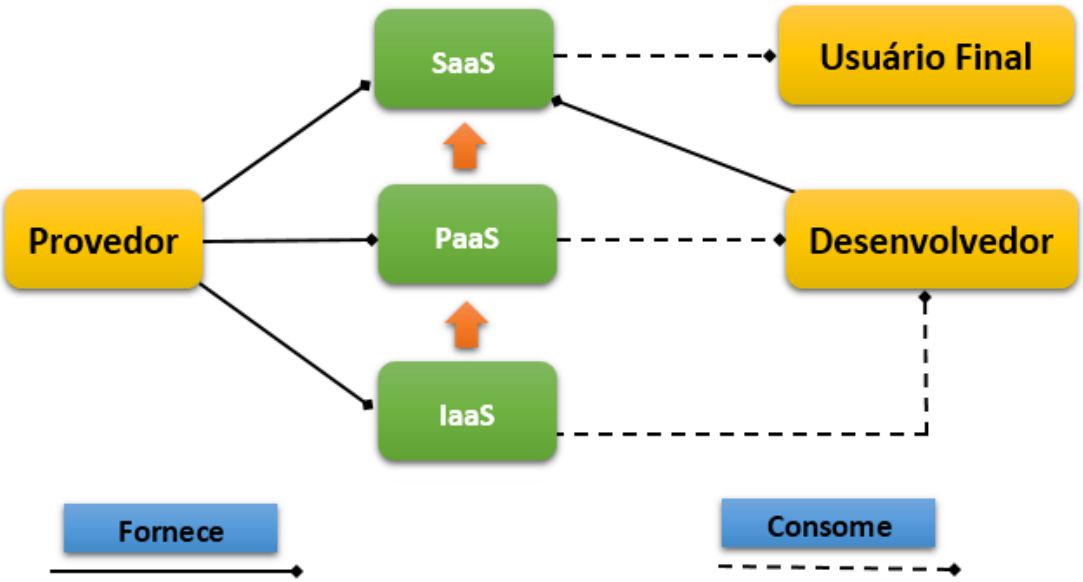

Figura 2.4: Papéis da computação em nuvem, adaptado de [64]

- Provedor: fornece toda a estrutura abstraindo-a do usuários final. Essa categoria gerencia e monitora toda a infraestrutura.

- Desenvolvedor: utiliza serviços fornecidos pelos provedores e provê a outros usuários serviços, assumindo assim o papel de um provedor.

- Usuários final: usa os serviços oferecidos pela nuvem.

2.1.4 Modelos de Implantação da Computação em Nuvem

A implantação da nuvem, segundo os autores em [37], irá depender da necessidade da aplicação e do tipo de contrato de prestação de serviços oferecidos. O resultado dessa dependência classifica os modelos de implantação como:

- A nuvem privada é criada para uso exclusivo de uma única organização. Esse tipo de nuvem é gerenciada pela própria organização. O autor em [40] afirma que se a nuvem for cuidada por terceiros, a infraestrutura utilizada pertencerá aos usuários, que terão total controle sobre toda a implantação das aplicações na nuvem.

- No modelo de nuvem pública os serviços são prestados para o público em geral. E os serviços oferecidos por esse modelo de nuvem podem ser gratuitos ou pagos pelo o que foi usado.

- Nuvem comunidade é provisionada para uso exclusivo por uma comunidade específica de consumidores de organizações que têm preocupações comuns. Pode ser gerenciada e operada por uma ou mais das organizações na comunidade, por um terceiro, ou por alguma combinação deles.

- Na nuvem híbrida temos que um modelo de implantação onde contém características tanto da nuvem pública quanto da nuvem privada, sendo que cada nuvem permanece como entidade única.

Em [23] o autor aponta um fator muito importante, ao se pensar em implantar solução de computação em nuvem deve-se decidir sobre que tipo de nuvem irá utilizar.

2.2 Virtualização

Por um certo período o foco da tecnologia de servidores era descentralizada, isso porque os *data center* centralizados eram vistos como caros e consequentemente difíceis de se manter. Diante disso, organizações com servidores centralizados passaram a utilizar várias máquinas físicas, o que se reverteria em um aumento satisfatório no desempenho e ainda garantia o isolamento das aplicações. Isso ajudaria na segurança caso uma máquina fosse comprometida não afetaria as demais. Mas essa não foi uma solução tão viável, porque haveria um desperdício de recursos e apresentava um altíssimo custo [36].

A virtualização ultimamente vem sendo bastante explorada no meio tecnológico, mas não é uma tecnologia nova, ganhou espaço nos anos 60 com a IBM que desenvolveu a primeira máquina virtual, ou seja, um único computador foi dividido em vários [36] e [10].

Segundo o autor [36], ao pensar em virtualização e considerando a arquitetura x86, alguns conceitos tem que ser pré-definidos, como: instruções privilegiadas/não-privilegiadas; o modo de operação do computador; sistema operacional hospedeiro/visitante e *Monitor de Máquina Virtual* (MMV) também conhecido como *hypervisor*.

Instruções privilegiadas e não privilegiadas

Instruções não privilegiadas são aquelas que não modificam o meio em que estão, nem o estado de recursos que estão sendo compartilhados, tais como processadores, memória

principal e registradores especiais. Já as privilegiadas podem alterar tanto o estado como a alocação de recursos.

O modo de operação do computador

O computador pode operar tanto no modo de usuário, que é onde as aplicações são executadas e nesse modo não se executa instruções privilegiadas, como no modo de supervisor que, possui controle total sobre a Unidade Central de Processamento (UCP) e dar o poder para executar tanto instruções privilegiadas como as não-privilegiadas.

Sistema operacional hospedeiro/visitante

O *Host Operating System* (HOS) é o sistema operacional que se localiza no hardware físico (hospedeiro), ou seja, é o sistema nativo da máquina no qual se terá a virtualização. Já o *Gest Operating System* (GOS) é os sistema que executa sobre o hardware virtualizado (visitante). Um ambiente virtualizado terá apenas um Sistema Operacional (SO) hospedeiro enquanto os sistemas visitantes poderão ser de vários SOs simultaneamente.

MMV ou Hypervisor

Segundo o autor [38] MMV é o hospedeiro das máquinas virtuais. Controla os recursos compartilhados pelas máquinas virtuais, como o processador, dispositivos de entrada e saída, memória e armazenamento. Também tem a função de escalonador, pois determina a hora de acionar uma maquina virtual para executar.

A figura 2.5 mostra o relacionamento das Vms e do VMM.

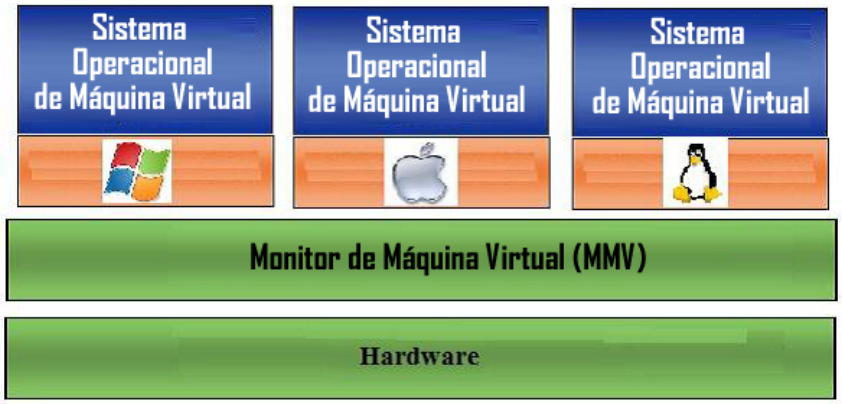

Figura 2.5: Relacionamento das VMs e do VMM. Adaptado de [36].

2.2.1 Vantagens e Desvantagens

Os autores [38] e [36] destacam algumas vantagem e desvantagens da utilização de virtualização.

As vantagens na utilização de virtualização segundo [38] [36] são:

- A segurança, já que as máquinas se encontram isoladas e cada máquina tem seus serviços. Logo a vulnerabilidade de um serviço não prejudica os demais;

- A confiabilidade e disponibilidade, pois a falha de um software não prejudica os outros serviços;

- Adaptação às diferentes cargas de trabalho. Essa vantagem pode ser obtida com ferramentas de realocar recursos de uma maquina virtual para outra;

- Balanceamento de carga, máquinas virtuais podem ser trocadas de plataformas. Com isso haverá um aumento no seu desempenho.

Por outro lado as desvantagens da virtualização são:

- A segurança: é evidente que todas as máquinas virtuais estão hospedadas em máquinas físicas. Este ponto é interessante, pois se o sistema operacional hospedeiro tiver alguma vulnerabilidade, todas as máquinas virtuais que estão hospedadas nessa máquina física estão vulneráveis. Logo são vulneráveis;

- A complexidade do gerenciamento: como em todo ambiente há a necessidade de monitoramento, configurações e ações;

- O desempenho: por não saber exatamente a quantidades de máquinas virtuais por processos, isso leva a duvidar da boa qualidade dos serviços diante de um grande número de máquinas virtuais.

2.2.2 Tipos de virtualização

Diante da dificuldade de implementar o modelo de virtualização proposto originalmente (virtualização completa de hardware), devido a não compatibilidade de drivers

de hardware, muitas técnicas de virtualização foram desenvolvidas. Tendo sido melhoradas no decorrer de anos, as técnicas de implementação de sistemas virtuais podem ser agrupadas em dois grupos principais Virtualização total e Para-virtualização, apresentadas nas Figuras 2.6 e 2.7.

Virtualização total: um ambiente com virtualização total provê copias fieis do hardware subjacente, de tal modo que o sistema operacional e as aplicações serão executados como se tivessem diretamente sobre o hardware original, não percebendo a virtualização. Para esse tipo de virtualização existe uma grande vantagem, o sistema operacional hospedeiro não precisa ser modificado [10].

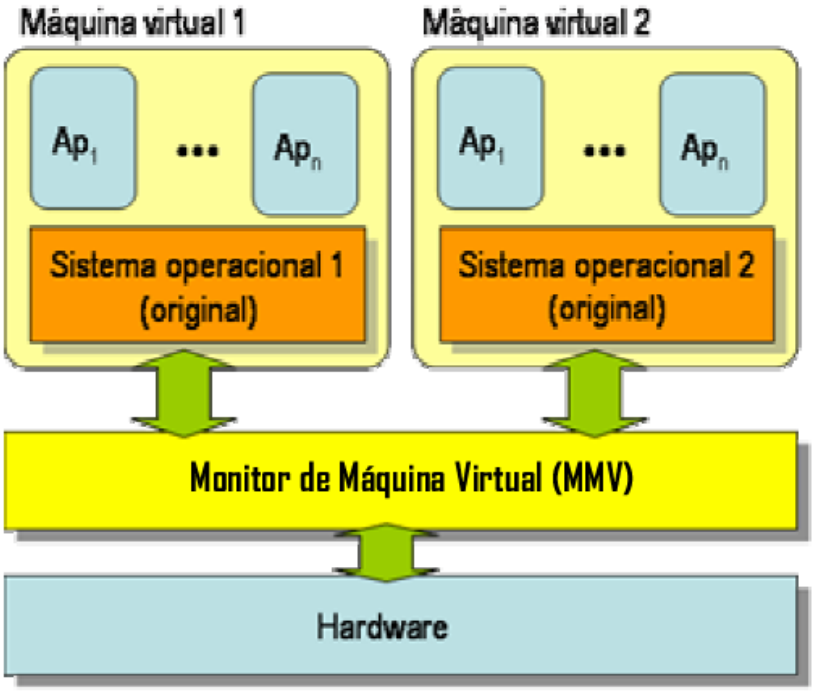

Figura 2.6: Virtualização total. Adaptado de [10]

Para-virtualização: é uma alternativa de virtualização que facilita a implementação do software [32]. Na para-virtualização as máquinas virtuais utilizam os seus próprios drivers, com isso poupa a capacidade total do dispositivo subjacente. Para esse tipo de virtualização o sistema operacional é modificado para chamar o *hypervisor* sempre quando for executar uma instrução que venha alterar o estado do sistema. Diante dessa característica da para-virtualização o *hypervisor* deixará de ser chamado para testar instruções por instruções, ganhando assim significativo desempenho de processamento.

Depois de abordar esta parte da virtualização, constata-se que ela responde perfeitamente ao desafio de isolamento com o qual a Nuvem se depara. Em particular, ela amplifica a prática da mutualização de recursos, que é o coração da Nuvem. A sua

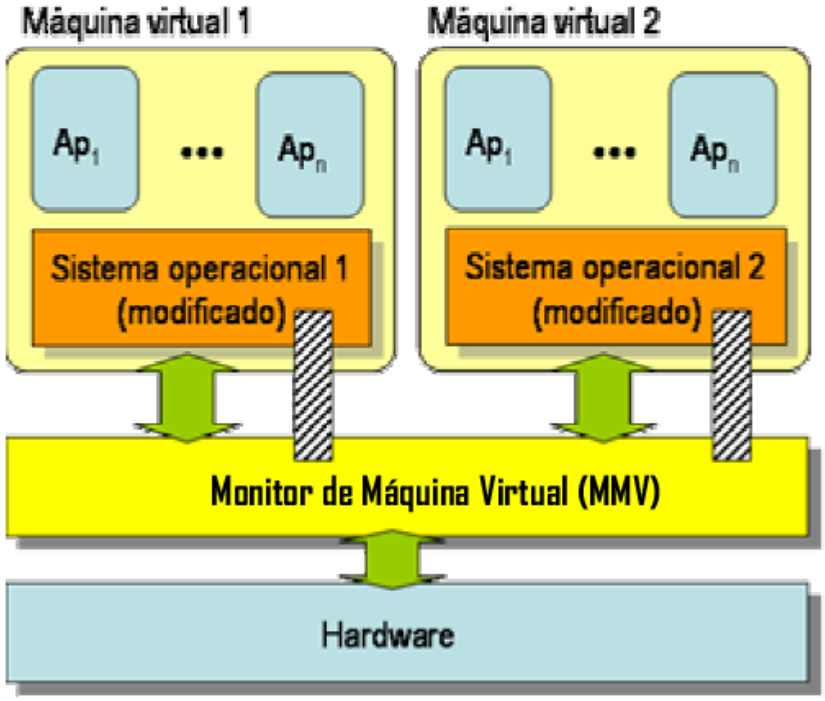

Figura 2.7: Para-virtualização. Adaptado de [10]

introdução na Nuvem envolve a modificação do modo de gerenciamento de recursos e tarefas administrativas em geral. No que se refere ao gerenciamento de recursos, a unidade de alocação na Nuvem vai da máquina real à máquina virtual.

2.3 Segurança

Ao longo da história da tecnologia é possível ver a crescente busca por segurança, isso porque os índices de incidentes que afetam a seguranças vêm aumentando exemplos: infecção por vírus, acessos não autorizados, ataques *denial of service* contra redes e sistemas, furto de informação proprietária, invasão de sistemas, fraudes internas e externas, uso não autorizado de redes sem fio, entre outras. Com esse novo paradigma da computação, essa é uma das questões mais importantes, pois o provedor de nuvem tem que assegurar:

Confiabilidade - dispor dos recursos somente a quem é autorizado e que esteja credenciado;

Integridade - certificar que a informação não foi modificada ou destruída de maneira não autorizada ou acidental;

Disponibilidade - permitir que a informação esteja acessível e utilizável sob demanda a qualquer momento.

Na literatura científica [17], [61], [47]) apresentam-se alguns princípios para garantir segurança de um sistema:

- *Autenticidade* – É a garantia de que as informações são verdadeiras ou que o usuário é quem diz ser;

- *Não repúdio* – é a garantia de que uma pessoa não consiga negar ou rejeitar um ato ou documento de sua autoria.

- *Legalidade* – Garante a legalidade (jurídica) da informação; Característica das informações que possuem valor legal dentro de um processo de comunicação, em que todos os ativos estão de acordo com as cláusulas contratuais pactuadas ou a legislação política institucional, nacional ou internacional vigentes.

- *Privacidade* – Foge do aspecto de confidencialidade, pois uma informação pode ser considerada confidencial, mas não privada. Uma informação privada deve ser vista, lida, alterada somente pelo seu dono.

- *Auditoria* – Rastreabilidade dos diversos passos que um negócio ou processo realizou ou que uma informação foi submetida, identificando os participantes, os locais e horários de cada etapa.

Quando uma organização migra para um ambiente de computação em nuvem com todas suas informações e identidades, passando a utilizar a infraestrutura do provedor da nuvem ela está abrindo mão de alguns níveis de controle. Para que essa migração ocorra nos dias de hoje, é preciso que os provedores da nuvem ofereçam confiança nos seus serviços.

Os principais elementos para proteger uma nuvem são: segurança da identidade (preserva a integridade e a confiabilidade), segurança da informação (a segurança estará centrada nas informações, já que as barreiras protetoras em nuvem são diluídas) e segurança da infraestrutura (a infraestrutura base de nuvem deve ser inerentemente segura) [6].

A computação em nuvem enfrenta o grande desafio de prover segurança aos seus usuários. O autor em [6] relata alguns dos principais riscos de segurança na utilização de computação nas nuvens: acesso privilegiado de usuários, cumprimento de regularização, localização dos dados, segregação dos dados, recuperação dos dados, apoio à investigação e viabilidade em longo prazo.

Para que a computação em nuvem possa atender as necessidades dos usuários de confiabilidade dos dados, o provedor da nuvem deve oferecer níveis elevados de seguranças.

Com a aplicação da segurança de forma correta as organizações irão começar a sentir confiança e consequentemente irão passar a depositar suas informações confidencias na nuvem. Diante disso o ambiente de computação em nuvem deve prover técnicas de segurança adequadas e métodos eficientes para seus usuários.

2.4 Sistema de Detecção de Intrusão

Com o avanço das tecnologias de comunicações de redes, em que as informações são compartilhadas, manter a segurança de um ambiente em rede é muito crítico. Por ter dados compartilhados e informações sigilosas, surgiram inúmeras ameaças que visam as informações. Diante do enorme avanço da tecnologia, houve também um enorme crescimento nas formas de fraudar a segurança de rede. Diante disso o uso de ferramentas de segurança, tornou-se indispensável para a proteção do ambiente em redes.

O sistema de detecção de intrusão é uma ferramenta utilizada para reconhecer comportamentos ou ações intrusivas na rede ou em um sistema. Um SDI pode acionar diversas contra medidas, inclusive alertar o administrador [58]. O administrador é o responsável pela configuração do SDI de acordo com as políticas de segurança da organização [41]. Um SDI é composto de diversos elementos: sensores, que geram eventos de segurança, ou seja, coletam informações; interface com o usuário para monitorar eventos e alertas e controlar os sensores; e uma base de dados que grava os eventos registrados pelo sensor e usa um sistema de regras para gerar alertas a partir dos eventos de segurança recebidos e analisados segundo as políticas da organização. Uma vez que é detectado alguma instrução, alertas são enviados. As respostas aos alertas podem ser de dois tipos: as ativas são respostas geradas aos incidentes pelo próprio sistema ou as passivas respostas geradas apenas para o administrador na forma de relatórios, para que diante de um incidente o administrador venha a tomar as medidas que julgar necessárias [45] [18].

Assim o SDI tem o papel de identificar todos os verdadeiros ataques e identificar todos os falsos ataques, e portanto, terá o conhecimento do comportamento ou ação intrusiva para poder alertar o administrador ou por si só disparar contra-medidas [21]

Segundo [51] um SDI realiza as seguintes tarefas:

- Monitoramento e análise de atividades de sistema e dos utilizadores;
- Realização de auditorias à infraestrutura, às falhas e às vulnerabilidades do sistema;

- Identificação do modelo de atividades do sistema, de forma a reconhecer-se sinais de ataques e de alertas;

- Realização de uma análise estatística a um modelo de comportamento anômalo;

- Avaliação da integridade dos ficheiros de dados e de sistema;

- Realização de auditorias de gestão ao sistema operativo e de reconhecimento do comportamento dos utilizadores que desobedecem à política de segurança da organização.

2.4.1 Classificação dos SDI

Conforme [4] [43] [15] [4] [20] [41], serão descritos algumas classificações dos sistemas de detecção de intrusão.

Quanto ao local de coleta de dados

- *Network-based Intrusion Detection* (NIDS): é um coletador das informações da rede, com o intuito de identificar ataques. Os NIDS são considerados *sniffers* de alto nível, capturam e analisam pacotes que trafegam na rede. Os pacotes oriundos da captura feita pelo SDI irão passar por uma analise, e na presenção de um desvio de padrão que já são pré-definidos, o sistema irá lançar um alerta informando uma intrusão, caso não seja encontrado um desvio os pacotes circulam normalmente. Uma desvantagem desta técnica de detecção é que ela pode detectar somente ataques conhecidos, ou seja, que estão incluídos no conjunto de assinaturas que o SDI possui, necessitando-se assim de constante atualização diante da rapidez que novos ataques surgem.

- *Host-based Intrusion Detection* (HIDS): Captura e avalia dados internos a um *host*, ou seja, o objetivo dele é monitorar e identificar ataques na máquina no qual foi instalado. Depois de coletados, os dados podem ser analisados localmente ou por máquina remota responsável pela avaliação desses dados. Um exemplo típico de um sistema de coleta de dados seria um sistema de registros (logs).

- Híbridos: Um SDI, para ser considerado híbrido, tem que utilizar sensores que capturem o tráfego de rede como sensores que capture os registros de eventos do *host*, com isso ele aumenta a capacidade de detecção. Boa parte das abordagens usam os dois tipos de SDI quanto ao local de coleta de dados (rede e *host*);

- SDI baseado em Máquina Virtual: em [41] especificamente pode ser vista essa definição de SDI. Segundo o autor, a coleta de informações de estados dos *host* são feitas de maneira diferente do convencional. As informações capturadas pelo *hypervisor* ou MMV podem ser utilizadas na detecção de intrusão. Esse VMM é uma camada entre o hardware e o sistema operacional.

Quanto à arquitetura

- Centralizada: contém elementos que capturam e analisam dados do sistema, em que as tarefas ficam aos cuidados de um único elemento, tornando-se assim um único ponto de falhas;

- Hierárquica: constituída por camadas, onde se tem analisadores, elemento que faz o tratamento dos dados e por fim um coordenador central responsável por todo o SDI, ou seja, toda ação tomada é feita pelo coordenador central. Como desvantagem, caso haja falhas em alguma camada todas as demais serão comprometidas.

- Distribuída: todos os elementos dessa arquitetura cooperam entre si, porém possuem autonomia e independência, características essas que dispensam a presença de um gerenciador central.

Quanto ao método de detecção

- Anomalia: Este tipo de SDI tem como definição de ataques todas as ações diferentes das atividades normais do sistema. É com as informações da base de dados que o sistema distingue o que é ou não permitido, em caso de divergência com o padrão o administrador da rede é alertado. Em uma rede existe diversos usuários, isso complica a configuração dos SDI por anomalia, pois é difícil estabelecer o que é padrão diante dessa situação, por outro lado é mais eficiente na detecção de novos ataques. No entanto falsos positivos podem ter uma alta taxa havendo a necessidade de treinamento do sistema.

- Assinatura: Um SDI por assinatura analisa os dados em busca de traços de intrusões por meio de padrões de taques pré-definidos pelo gerenciador do sistema, guardados em uma base de dados. Sistemas que analisam por assinatura possuem uma grande desvantagem, pois só conseguem detectar somente ataques conhecidos, ou seja, que estão incluídos no conjunto de assinaturas (base de dados) que o SDI possui. Nesse caso o administrador deve fazer constante atualização da base de dados de assinatura (ataques) a medida que novos ataques surgem.

- Híbrido: Este método detecta tanto por ataques conhecidos como comportamentos anormais, utilizando assim as duas abordagens anteriores.

Quanto ao comportamento na detecção

- Passivo: um SDI considerado passivo é aquele que simplesmente detecta o tráfego suspeito ou malicioso, gera um alerta ou notificação sobre a intrusão e envia para o administrador. Diante da detecção de um pacote suspeito na rede, esse tipo de SDI não toma nenhuma atitude em relação à ação suspeita.

- Ativo: além de detectar ações intrusivas e alertar o administrador, o SDI também possui ações pré-definidas para responder às ameaças.

Quanto à frequência de utilização

- Monitoramento Contínuo: todo o tráfego é coletado e analisado no mesmo instante, pois o sistema é em tempo real.

- Análise Periódica: o sistema funciona de modo estático, pois a captura das informações serão feitas periodicamente e uma análise é feita se necessário.

2.5 Sistemas de Detecção de Intrusão para Nuvem

Computação em nuvem é uma nova abordagem onde os recursos computacionais são oferecidos sob demanda e escalável. A forma que os recursos são disponibilizado na nuvem, torna esse ambiente mais propício a invasões e ataques. Mas a forma de tornar esse ambiente seguro não dirente. Há uma exigência de técnicas adequadas e robustas para tratar ataques e invasões no ambiente em nuvem.

Pesquisas feitas por [2] [60] [16] [46] [27], mostram a eficiência de sistema de detecção de intrusão na nuvem. Porém, é notória uma preocupação com o melhoramento dessas técnicas. Logo mais são descritos, brevemente, esses trabalhos.

***Intrusion Detection System as a Service in Public* (IDSaaS)**

Em [2] foi proposto um IDS escalável e ajustável aos usuários da nuvem, provendo a habilidade de monitorar e reagir a ataques em várias VMs existentes dentro da rede virtual privada dos usuários. O IDSaaS usa a rede para coletar os dados e é baseado em assinatura.

Esse modelo por ser na nuvem além de escalável, portável e sob demanda, o usuário pagará somente pelo que usar.

***Intrusion Detection for Grid and Cloud Computing* (GCCIDS)**

Vieira et. al. (2010), propôs um sistema de detecção de intrusão para ambientes de computação em grade e em nuvem. O GCCIDS é a nível de middleware da nuvem. Pelas características de isolamento da virtualização é protegido contra intrusos. A arquitetura do SDI proposta pelo autor possui arquitetura distribuída, e na ocorrência de uma intrusão todos os nós ficarão sabendo, pois um alerta é emitido para todo o ambiente [60].

Intrusion Detection System in Cloud Computing Environment

O trabalho proposto por [16] apresenta um SDI que detecta tanto por assinatura quanto por aprendizagem. o SDI é composto por instâncias classificadas como "mini SDI" criadas por um controlador central e único chamado de SDI *Controller*. Segundo o autor, a principal vantagem do IDSCCE é a redução da carga de trabalho, já que as instâncias são implementadas entre cada usuários e o provedor da nuvem.

IDS *Management architecture*

O trabalho elaborado por [46] apresenta uma arquitetura extensiva de SDI, onde existe um controlador componente central que analisa os resultados coletados pelos vários sensores inseridos nas máquinas virtuais. Os sensores espalhados fazem troca de mensagens com o padrão *Intrusion Detection Message Exchange Format* (IDMEF). Os componentes desta estrutura são: máquinas virtuais, chamadas de VM SDI e A unidade central de gerenciamento que controla as VM IDS e é formada por quatro componentes: *Event Gatherer, Event Database, Analysis* Componente e IDS *Remote Controler*.

***A Framework for Intrusion Detection in Cloud Systems* (CIDS)**

Kholidy e Baiardi (2012) apresentam um framework para SDI baseado em nuvem, no qual o objetivo é lidar com ataques como *masquerade attacks* (onde atacantes se fazem passar por usuários legítimos), *Host-based attacks* (que podem ser uma consequência de *masquerade attacks*) e *Network-based attacks*. O CIDS envia ao administrador relatórios resumidos. Os elementos do CIDS estão localizados fora das máquinas virtuais. Devido a isso, mantêm seus próprios componentes protegidos de ameaças que possam afetar as VMs. CIDS não possui um coordenador central, é escalável e usa uma solução P2P. Sua arquitetura distribui a carga de processamento por vários locais da nuvem. Cada nó na arquitetura possui uma base de dados com assinaturas de ataques conhecidos (conhecimento) e outra com padrões de comportamento anormal (comportamento).

A Tabela 2.1 apresenta uma análise comparativa relacionada às características relevantes das principais arquiteturas de SDI apresentadas nesta seção.

Tabela 2.1: Comparativo entre SDI baseados em VM para Computação em Nuvem

	Arquitetura	Forma de Proteção do SDI	Captura pacotes nas Vms	Localização	Uso da elasticidade	Tolerância a falhas
GCCIDS	Distribuído	Por nó	Não	Nó Físico	Não	Não
IDSCCE	Centralizado	Por nó	Não	Nó Físico	Aumenta a capacidade do SDI	Não
IDSaaS	Centralizado	Por serviço	Sim	Nas VMs	Aumenta a capacidade do SDI	Não
MA	Centralizado	Por nó	Sim	Nó Físico	Aumenta a capacidade do SDI	Não
CIDS	Distribuído	Por nó e VM	Sim	No Físico e nas VMs	Aumenta a capacidade do SDI	Não

2.6 Tolerância a Falhas

A busca por sistemas mais sofisticados e com um poder tecnológico maior tornou mais complexo os sistemas e com isso sujeitos a falhas. Com essa sofisticação, distribuição, dinamismo e complexidade dos sistemas há uma necessidade de torná-los mais seguros e tolerantes a falhas.

Conforme visto em [62] [52], o principal objetivo da Tolerância a Falhas (TF) em um sistema é alcançar a dependabilidade que é a qualidade do serviço fornecido por um dado sistema e a confiança depositada no serviço fornecido, onde o funcionamento não será interrompido mesmo na presença de falhas.

Um sistema para ser TF tem que continuar fornecendo seus serviços mesmo diante de falhas. A TF é o conjunto de técnicas utilizadas para detectar, mascarar e tolerar falhas no sistema [52].

Na literatura encontramos inúmeras descrições de como projetar um sistema de tolerância a falhas, e para se projetar um sistema é preciso primeiramente entender como as falhas ocorrem. Em um sistema os níveis de percepção podem ser: **falha, erro e defeito**.

Weber (2003) define um **defeito** como um desvio da especificação. E ainda diz que os defeitos não podem ser tolerados, mas devem ser evitados. Para o autor já citado, um sistema está em estado errôneo ou em **erro** se o processamento posterior a partir desse estado pode levar a defeito. Por fim enfatiza o conceito de **falha** como a causa física ou algorítmica do erro [62].

Segundo [52], os autores afirmam que o principal obstáculo na implementação de tolerância a falhas é a imprevisibilidade de ocorrência de uma falha e os efeitos que elas trarão ao sistema. A ação do mecanismo de tolerância a falhas só será iniciada mediante a detecção de erros encontrados no sistema, ou seja, resultados de falhas. Com a detecção de um erro, medidas como remoção, e caso necessário, o sistema deve ser reconfigurado para isolar o percursor da falha. Na literatura esse processo é chamado de recuperação de erro. Com a recuperação de erro conjetura-se que o sistema continue com seu funcionamento normal, sem apresentar defeitos.

2.7 Trabalhos Relacionados

O ambiente de computação em nuvem por ser virtualizado e escalável ainda é um grande desafio fornecer tolerância a falhas, embora muitos trabalhos tenham sido feitos nessa área, há uma necessidade de técnicas de tolerância a falhas eficazes para a Computação em nuvem. Na literatura existem muitas pesquisas que fornecem várias técnicas de tolerância a falhas como [35] [1] [64] [57] [25] [13] [28]. Enumeram-se a seguir trabalhos relacionados.

Adaptive Fault Tolerance in Real Time Cloud Computing (AFTRC)

O trabalho proposto por [35] aplica a tolerância a falhas para a computação em nuvem em tempo real. Este trabalho teve um aproveitamento tanto da capacidade computacional como do ambiente virtualizado escalável de computação em nuvem. Neste modelo, o sistema proposto tolera a falha de forma proativa, ou seja, evita a recuperação de falhas, erros e defeitos, prevendo-os e os componentes suspeitos são trocados por outros componentes de trabalho e faz a dicção com base na confiabilidade dos nos de processamento.

Fault tolerance middleware for cloud computing (LLFT)

O modelo proposto por [64] é um middleware de tolerância a falha de baixa latência para fornecer a tolerância a falhas para aplicações distribuídas implantadas no ambiente de Computação em Nuvem como um serviço oferecido pelos proprietários da Nuvem. Este modelo baseia-se em um dos principais desafios da Computação em Nuvem, o de assegurar

aplicações que estejam executando na Nuvem sem interrupção no serviço fornecido para o usuário. Este middleware replica aplicação pelo uso de replicação semiativa ou processo de replicação semi-passiva para proteger a aplicação contra vários tipos de falhas, podendo ser citadas: *Crash fault* – o processo ou o processador não produz quaisquer resultados, *Timing fault* - um processo ou processador não produz um resultado de restrição dentro de um determinado tempo. O middleware LLFT não lida com falhas bizantinas e que foi implementado na linguagem C++ para o LINUX.

O ambiente proposto consiste em grupos de serviço que são conjuntos de grupos de processos que interagem através de conexão virtuais dentro de uma nuvem, fornecendo coletivamente um serviço para o utilizador e grupos de processos que é um conjunto de réplicas de um processo. Cada membro de um grupo de processos deve saber o básico de seu grupo. Essa interação é feita por uma conexão virtual ponta-a-ponta de TCP. O middleware LLFT é transparente para o aplicativo, e não requer recopilação ou religamento do programa de aplicação.

Fault tolerant workflow scheduling based on replication and resubmission of tasks in Cloud Computing (FTWF)

Foi proposto por [25] um algoritmo de tolerante a falhas de agendamento de fluxo de trabalho para fornecer tolerância a falhas usando a replicação e a resubmissão de tarefas com base na prioridade das tarefas em uma heurística matricial.

O FTWS permite aos usuários executar seus fluxos de trabalho por prazo satisfatório apesar de diferentes falhas que ocorrem no meio ambiente. A replicação de tarefas depende de uma métrica heurística que é calculado encontrando o equilíbrio entre o fator de replicação e fator de reenvio. A métrica heurística é considerada porque a replicação por si só pode levar ao desperdício de recursos e reapresentação sozinha pode aumentar o *makespan* (diferença de tempo entre o início e o fim de uma sequência de trabalhos ou tarefas). As tarefas são priorizadas com base na importância da tarefa que é calculado usando parâmetros como o grau, mais cedo prazo e de alto impacto reapresentação. A prioridade ajuda a cumprir o prazo de uma tarefa e reduzindo, assim, o desperdício de recursos. O ambiente de nuvem é simulada usando máquinas virtuais VMware. A comunicação entre eles ocorre por meio de programação de SSH. Diferentes tipos de falhas podem ocorrer em um ambiente de nuvem também são simulados. Os servidores de nuvem podem ser de dois tipos: servidor de armazenamento e servidor computacional. O FTWS foi projetado com quatro módulos que são: *Preprocessor Module* (PM), *Replication based Scheduler Module* (RSM), *Executor Module* (ResEM) e *Rescheduling if required and Data Scheduler* (DS).

VFT: A virtualization and fault tolerance approach for cloud computing (VFT)

É apresentado por [13] uma técnica de virtualização e Tolerância a Falhas o VFT: A virtualization and fault tolerance approach for cloud computing, para reduzir o tempo de serviço e aumentar a disponibilidade do sistema. Para gerenciar a virtualização, balanceamento de carga e para lidar com as falhas é utilizado os seguintes componentes nessa proposta: CM - Cloud Manager e um DM – Decision Maker.

O processo desse trabalho se dá:

- A virtualização e o balanceamento de cargas;
- A tolerância a falhas que é alcançada por redundância, checkpointing e manipulador de falhas.

O VFT foi projetado principalmente para fornecer tolerância a falhas reativa. E possui um manipulador de falhas, que se encontra tanto no hypervisor [26] como na parte virtualizada do sistema. A função do manipulador é bloquear nodes com falhas irrecuperáveis junto com seus nós virtuais para futuras solicitações e remover as falhas de software temporários dos nodes com falhas recuperáveis e os tornar disponíveis para futuras solicitações. Hypervisor também mantém registro quando o Load Balancer (balanceamento de cargas) atribui uma tarefa a um nó virtual de um servidor em particular, a fim de medir a taxa de sucesso do servidor físico.

Neste trabalho também é calculado o desempenho do servidor físico usando a taxa de sucesso. Onde Se tem n1 = número de vezes que um servidor físico dá resultados bem sucedidos e n2 = número total de vezes que os pedidos enviados para o servidor, então a taxa de sucesso é medida assim SR = N1/N2, onde n1 <= n2.

Esse modelo não só tolera falhas, mas também reduzir a chance de futuras falhas por não atribuir tarefas aos nós virtuais de servidores físicos cujas taxas de sucesso são muito baixas. Foi concluído com a ajuda de métrica de desempenho de comparação e analise da taxa de sucesso.

A Hierarchical and Autonomous IDS for Cloud Systems (HA-CIDS)

Kholidy *et al.* [28] é apresentado um sistema de detecção de intrusão hierárquica e autônoma baseado em nuvem. O HA-CIDS monitora continuamente e analisa os eventos do sistema e calcula os parâmetros de segurança e de risco.

Para atingir a tolerância a falhas essa proposta executa as seguintes técnicas:

- Replicação de elementos de processamento;
- Verificação de integridade de auto-cura;
- Detecção de falhas usando mensagens de pulsação em sistemas multiagentes;

O HA-CIDS tolerância as falhas de forma reativa. Um controlador autônomo recebe parâmetros de segurança computados pelo quadro e seleciona a resposta mais adequada para proteger a nuvem contra ataques detectados, bem como recuperar os dados corrompidos ou serviços afetados. Ao lado de resposta autônoma de ataques detectados, HA-CIDS tem várias capacidades autônomas de proporcionar auto- resistência e tolerância a falhas.

O HA-CIDS desenvolveu um testbed que avalia o desempenho e precisão do framework. Os principais componentes do HA-CIDS são: Eventos Colecionadores (*Events Collectors*) são sensores de monitoramento de software que coletam eventos de segurança, Eventos Correlator (*Events Correlator*) integra e correlaciona os eventos de acolhimento e de rede coletados de Events Correlator hospedado em diferentes VMs, Analisador de Eventos (*Events Analyzer*) usa vários mecanismos de análise para detectar eventos de acolhimento e ataques de rede e Controlador (*Controller*) seleciona a resposta mais adequada e método de proteção para proteger os anfitriões e a rede contra ataques detectados e potencial. O analisador de eventos por sua vez tem os seguintes componentes HIDS *Analyzer* é um analisador baseado em assinatura para detectar ataque baseado em trilhas deixadas pelos ataques conhecidos ou sequências predefinidas de ações do usuário que possam representar um ataque, NIDS *Analyzer*: é um analisador baseado em assinatura implementada que analisa o tráfego de rede entre as VMs nuvem, espelhando-o do switch virtual para detectar trilhas conhecidas que possam representar um ataque e DDSGA *Analyzer* é um analisador baseado em comportamento para detectar ataques de máscaras.

A metodologia utilizada pelo AH-CIDS para a fase de detecção tem-se 1- Processo de coleta, 2 – Processo de integração, 3 – Processo de correlação e 4 – Processo de avaliação de risco. E na métrica de segurança e avaliação de risco, onde a métrica de risco é uma medida do impacto potencial de uma ameaça sobre os ativos, dada a probabilidade de que ele irá ocorrer e fornece informações úteis para avaliar o estado geral de segurança da nuvem.

As características autônomas HA-CIDS, ou seja, a tolerância a falhas, incluindo o recurso de Self-Resiliência e as ações de resposta automática. Para a realização da tolerância a falha e a *self*- Resiliência utilizou-se dos seguintes mecanismos:

- Replicação dos componentes;

- Algoritmo mensagem assinada *Signed Message Algorithm* (SMA);
- *Self*-Resiliência.

Para a auto-resposta o controlador recebe alertas de segurança e parâmetros de risco a partir do componente Estimador de Impacto Ataque e seleciona o método de resposta mais adequada para proteger os anfitriões e a rede contra ataques detectados e potenciais, bem como recuperar os dados corrompidos ou serviços afetados para restabelecer o estado de funcionamento de atacado recursos. O trabalho mostra as técnicas de auto-resposta utilizada, bem como os mecanismos de Self- resiliência implantados para aumentar a disponibilidade do sistema e a sua tolerância a falhas, reduzindo significativamente o esforço humano e os conhecimentos necessários.

A Tabela 2.2 apresenta uma análise comparativa entre os sistemas descritos anteriormente.

Tabela 2.2: Comparação dos resultados obtidos pelos trabalhos relacionados analisados.

	Procedimentos aplicado para tolerar falhas	Tipos de Falhas Previstas	Tolerância a falhas por VMs	Tolerância a falhas por componente	Tolerância para a propagação de ataques na nuvem
AFTRC	Exclui o nó dependendo de sua confiabilidade	Confiabilidade	Não	Não	Não
LLFT	Replicação	Crash-cost; Trimming fault	Sim	Não	Não
FTWF	Replicação e ressubmissão de jobs	Dead line of work flow	Sim	Não	Não
VFT	Balanceamento de carga; redução de trabalho; Redundância; Check Pointing; Fault Handler.	Sobrecarga de Node; Disponibilidade	Sim	Não	Não
HA-CIDS	Redundância;	Crash-cost; Confiabilidade	Sim	Não	Não

2.8 Conclusões

Neste capítulo, alguns conceitos relacionados com a computação na nuvem, sistemas de detecção de intrusão e tolerância a falhas para o ambiente de computação na nuvem foram apresentados.

Em computação na nuvem, a sua definição, suas características e classificações foram abordadas. Também foi abordado o pila principal dessa abordagem a virtualização com os seus tipos. Na seção de sistemas de detecção de intrusão foi exposto suas características e classificações, também foi apresentado alguns sistemas de detecção para nuvem. Na seção de trabalhos relacionados, alguns projetos aplicados para nuvem e suas principais características foram apresentados. E por fim uma comparação entre as características dos trabalhos foi apresentada.

Com essa comparação concluiu-se que há uma grande necessidade do aumento de tomada de decisão e proteção dos componentes do controlador com relação aos objetivos de ataques e a possível propagação de intrusão no ambiente.

O presente trabalho apresentou a taxonomia de falhas, as falhas na nuvem e as técnicas de tolerância a falhas para nuvem. Diante da análise concluímos que, os mecanismos necessitam ser tolerante por se só e as técnicas empregadas no ambiente também, além de seguir as mudanças que ocorrem.

Apesar de muitas técnicas e sistemas proposto para essa nova abordagem, ainda há uma necessidade de melhorar cada vez mais a segurança da nuvem e todo o quadro de mecanismos inseridos a ela, uma vez que o mesmo possui dado e informações de terceiros que devem ser guardadas com privacidade e confiabilidade. Em um ambiente computacional realizado nas nuvens apresenta muitas vulnerabilidades.

3 Projeto EICIDS

Este capítulo aborda o Projeto EICIDS. As principais características são apresentadas. A arquitetura, funcionamento e todos os componentes são descritos. A funcionalidade de cada componente também é explicada neste capítulo.

3.1 Características

O EICIDS, proposto por [18] e desenvolvido na Universidade Federal do Maranhão, é um sistema de detecção de intrusão para ambientes de computação em nuvem elástico, que tem como base a técnica de virtualização. Sua arquitetura visa prover a segurança dos usuários da nuvem em nível de infraestrutura. Ou seja, protege as máquinas virtuais que compõem a camada de IaaS (base das camadas PaaS e SaaS). O trabalho visa a proteção das máquinas virtuais do ambiente de computação em nuvem contra usuários internos que possam utilizar alguma máquina virtual para realizar atividades maliciosas ou que apresentem comportamento suspeito.

O monitoramento das máquinas virtuais é feito por meio de sensores espalhados pelo ambiente de nuvem a ser protegido. Essa proteção evita que usuários maliciosos tenham acesso ao conteúdo de outras VMs ou possam fazer um uso abusivo para efetuar ataques a outras VMs [18]. A Figura 3.1 ilustra o modelo conceitual do SDI proposto por [18].

Os objetivos do EICIDS [18].

- Implementar um protótipo que atue a nível de infraestrutura (IaaS) do middleware de computação em nuvem;

- Prover segurança nas máquinas virtuais dos usuários de IaaS.

3.1.1 Arquitetura do EICIDS

Segundo o autor em [18], a arquitetura proposta tem como objetivo prover segurança aos usuários de ambientes de computação em nuvem em nível de infraestrutura, ou seja, protegendo as máquinas virtuais que compõem a camada de IaaS da nuvem que serve

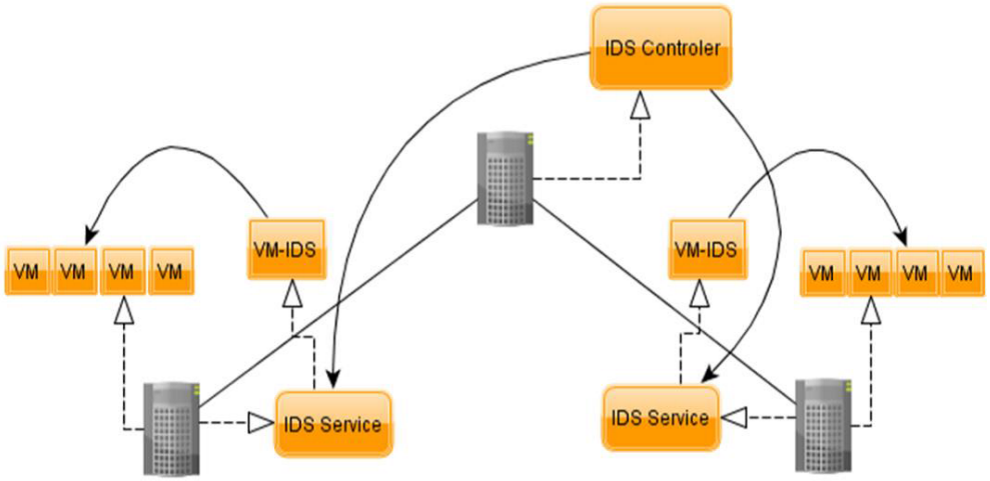

Figura 3.1: Modelo conceitual do EICIDS [18].

de base às outras camadas (PaaS e SaaS). As Figuras 3.2 e figura 3.3 mostram a arquitetura com diversos nós do SDI sendo controlado pelo IDS_Admin.

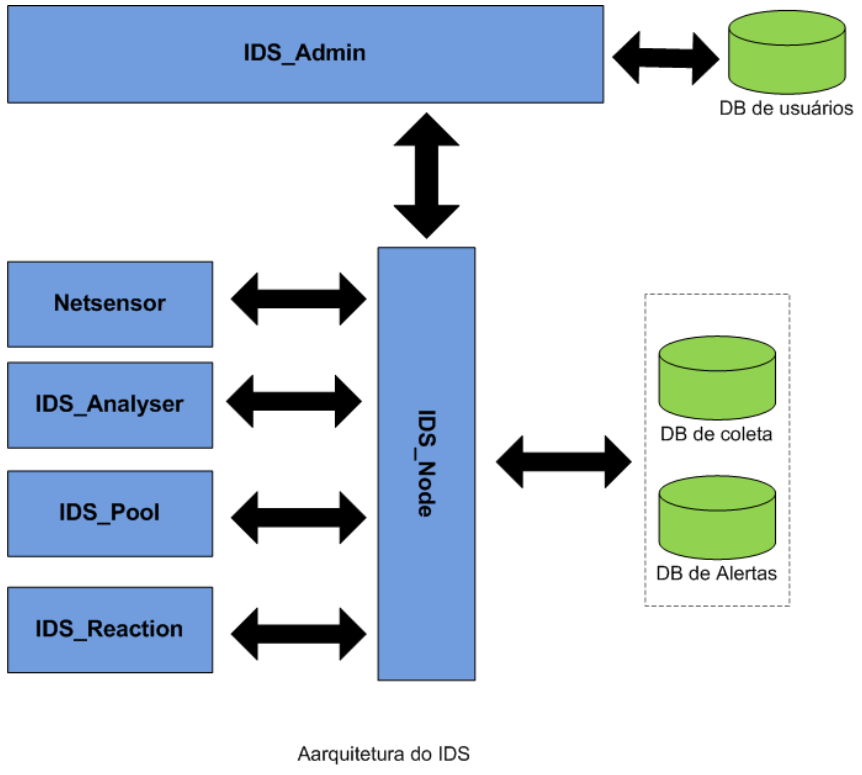

Figura 3.2: Arquitetura do EICIDS [18].

Na Figura 3.2 tem a arquitetura do SDI que se aplica a cada nó da nuvem, por meio de seus componentes coleta informações das máquinas virtuais efetuando a detecção de atividades suspeitas e posteriormente enviando alertas ao módulo de controle, denominado

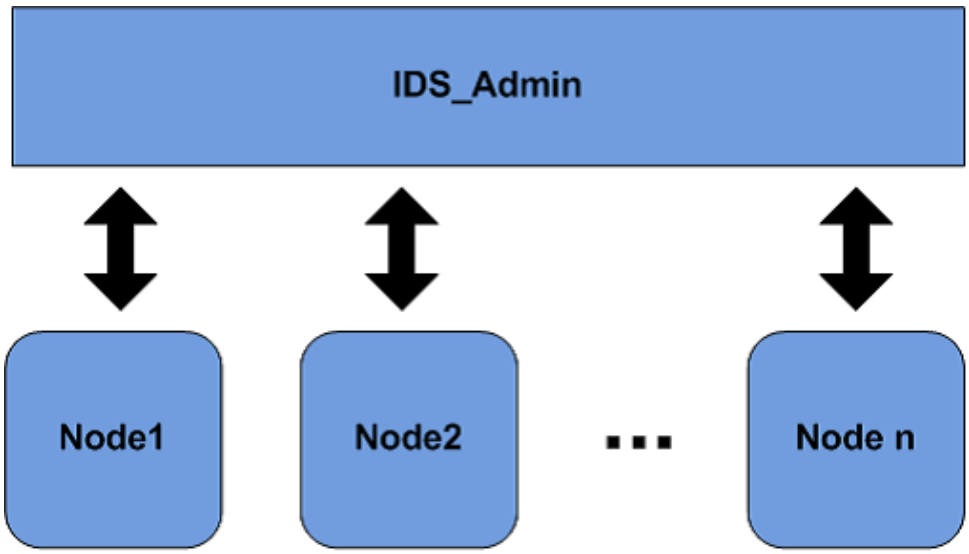

Figura 3.3: Arquitetura do EICIDS (Com vários nodes) [18].

IDS_Admin. Já na Figura 3.3 é apresentada a expansão em uma arquitetura de nuvem com vários nós, sendo que o IDS_Admin é o elemento central que controla os vários módulos do SDI e cada componente denominado Node (*Node1, Node2,...,Node*) engloba os elementos IDS_Node, *Netsensor*, IDS_*Analyser*, IDS_Pool e IDS_*Reaction*, o que torna a arquitetura do IDS mista entre centralizada e distribuída [18]. Os componentes e suas definições serão detalhados nas seções seguintes.

3.1.2 Componente do EICIDS

A seguir serão apresentados os componentes que formam o SDI proposto e logo depois a funcionalidade de cada componente.

IDS_Adim

O IDS_Admin é o componente responsável pelo gerenciamento do processo de detecção de intrusão, devido a integração de todos os componentes do SDI. O elemento central (IDS_Admin) controla os componentes e também faz uma organização, sumarização e apresentação dos alertas de segurança emitidos pelos vários nodes do sistema da nuvem.

O IDS_Admin é o controlador de todas as informações sobre os componentes do SDI, por meio do controlador se tem uma visão geral do comportamento do sistema e uma interface funcional é disponibilizada para o administrador da nuvem. Por meio da base de dados de usuários da nuvem, é possível identificar a que conta de usuários pertencem as

máquinas virtuais envolvidas em incidentes de segurança reportadas ao IDS_admin pelos diversos nós da nuvem.

IDS_Node

O IDS_Node instancia e controla os sensores e analisadores localizados nas máquinas virtuais, monitorando assim o ambiente virtual da nuvem. Logo, esse componente é o responsável por receber os alertas de segurança provenientes da análise de assinatura e encaminhá-los ao IDS_Admin. Caso a análise por assinatura não tenha encontrado um ataque padrão, o IDS_Node encaminha os pacotes capturados para uma analise de detecção de anomalias, isso é feito para se ter uma certificação de que nenhum tráfego suspeito, eventos indesejados ou padrões que indiquem intrusão possam ter passado pelo processo por assinatura, mas caso seja encontrado será identificado e encaminhado ao administrador.

O IDS_Node recebe periodicamente informações do IDS_Pool sobre a existência ou não de maquinas virtuais de usuários em execução, isso o auxilia na instanciação ou remoção dos sensores e analisadores do ambiente virtual no intuito de tanto poupar recursos da nuvem, quando da ausência de VMs executando, quanto de acompanhar o crescimento da nuvem, quando mais clientes solicitam VMs.

IDS_Pool

A função do IDS_Pool é monitorar o ambiente virtual afim de encontrar máquinas virtuais ativas ou não. Este módulo do SDI utiliza comandos tanto do sistema operacional hospedeiro, como comandos do próprio middleware para realizar esta tarefa. Diante do monitoramento, caso alguma máquina virtual seja encontrada, esse componente envia uma mensagem ao IDS_Node para que sensores e analisadores sejam ativados. Do mesmo modo, quando não é encontrada nenhuma máquina virtual, é enviada uma mensagem ao IDS_Node para poupar os recursos da nuvem, assim os sensores e analisadores são desativados.

IDS_Sensor

No componente IDS_Sensor é iniciada a detecção, esse componente captura os pacotes que trafegam na rede na qual se encontram as máquinas virtuais dos usuários da nuvem. O sensor atua como um *sniffer* e interage com a rede de forma passiva e interfere o mínimo possível, isso para que não haja comprometimento com a performance do ambiente das máquinas virtuais e não corrompa o tráfego. Em meio ao processo, é feita uma filtragem de pacotes capturados, para que apenas as informações dos pacotes que se tenha interesse sejam selecionadas para a análise a ser feita posteriormente. Os pacotes são guardados em uma base

de dados, para que a análise tanto por assinatura como anomalia possam ser feitas. O IDS_Pool captura apenas o cabeçalho dos pacotes de IP.

Isso não o faz o componente ir de encontro com as leis de privacidade como visto em [53]. As leis de privacidade podem dificultar no processo de adicionar medidas de segurança para a computação em nuvem. Tudo que é capturado é armazenado numa base de dados para ser analisado posteriormente pelos demais componentes do IDS.

IDS_Reaction

O IDS_Reação é responsável por efetuar uma contra medida diante de ações maliciosas, detectadas no sistema. É nesse componente que uma máquina virtual pode ser pausada caso apresente um comportamento suspeito, até mesmo desligada ou destruída se for constatado que apresenta ação maliciosa. Quem aciona esse componente é o IDS_Node quando uma atividade maliciosa, desencadeando uma ação programada para a intrusão e depois finalizando a sua execução.

Análise por assinatura

Esse componente é responsável pela análise dos dados gerados pelo sensor, aplicando regras de detecção, assim as ações maliciosas de algumas máquinas virtuais são identificadas ainda nele antes de serem enviadas ao componente IDS_Node. O analisador de assinatura fica localizado no ambiente virtualizado criado pelo IDS. Ao encontrar um padrão de ataque ou não essa informação imediatamente é encaminhado uma mensagem de alerta para o componente IDS_Node para que seja feita uma análise mais detalhada.

Análise por anomalia

Nessa etapa é feita uma análise dos dados gerados pelo sensor e que foram encaminhados pelo componente de análise de assinatura para ser feita uma nova análise. Qualquer comportamento fora do normal em alguma VM um alerta é emitido para o *node* ou, em caso de suspeita, as informações serão encaminhadas para o *Network Intrusion Detection System based on Inteligent Agents* (NIDIA) [33], que possui uma base de dados de intrusão de maior abrangência. O analisador de anomalia localiza-se fora do ambiente virtualizado, mais especificamente no componente *Node*.

3.1.3 Funcionamento do EICIDS

Na construção do ambiente que o EICIDS foi aplicado o EUCALYPTUS [34] que é um *open source* de infraestrutura como serviço, utilizando o hypervisor KVM [30]. A arquitetura

desenvolvida por [18] é composta de um elemento central de entrada principal da nuvem que era chamado de *Cloud controller* e os Nodes controller componentes de execução de máquinas virtuais.

O IDS que foi proposto tem uma estrutura centralizada e hierárquica. O IDS_Admin é o elemento central e inicializa os módulos em cada *node* da arquitetura de nuvem, o IDS_Node envia todas informações a ele através de conexão de rede. O monitoramento é feito por meio de sensores de IDS inseridos nas máquinas virtuais capturando informações das VMs dos usuários pelo tráfego das vlans. É feito uma análise local, por meio de assinatura, caso seja encontrado um padrão de ataque, um alerta é enviado para o controlador central e uma contra medida é efetuada e caso não seja encontrado é encaminhado para o componente *node* central da nuvem para uma análise mais completa, utilizando métodos de anomalia, ou ainda pode ser enviado para uma base de intrusão mais abrangente como o NIDIA. A alocação dos sensores acompanha o crescimento ou a redução de consumo dos recursos de cada usuários da nuvem (propriedade de elasticidade da Computação em Nuvem).

Na Figura 3.4 é ilustrado um diagrama de atividades do funcionamento da arquitetura proposta por [18]. O diagrama apresenta uma visão simplificada do funcionamento interno do EICIDS, com suas funções ou processo [49]. Quando usado para a modelagem de software, as atividades normalmente representam um comportamento invocado como resultado da chamada de um método. Quando usado para modelagem de negócios, as atividades podem ser desencadeadas por eventos externos, tais como uma ordem a ser colocada das ações desenvolvidas [42] [22].

O *netsensor* captura pacotes da rede, e o fluxo de atividades para o sistema de detecção de intrusos é iniciado. Os pacotes capturados são encaminhados para análise e o modulo responsável é o IDS_*Analyser* onde são feitas comparações com um padrão pré-definido no intuito de identificar ataques conhecidos. Caso seja encontrado um padrão, é enviado um alerta ao componente *Node* que verifica o tipo de ataque e envia um relatório ao componente IDS_Admin e ainda aciona o componente de contra medidas IDS_*Reaction*. Caso, nenhum padrão seja encontrado, os pacotes são encaminhado para análise por anomalia, onde o tráfego de rede é comparado com um padrão capturado anteriormente em um período considerado livre de ataques.

Na etapa de análise por anomalia existem três situações que podem ocorrer:

- Pode não ser identificado nenhuma anomalia, neste caso, os pacotes são descartados.

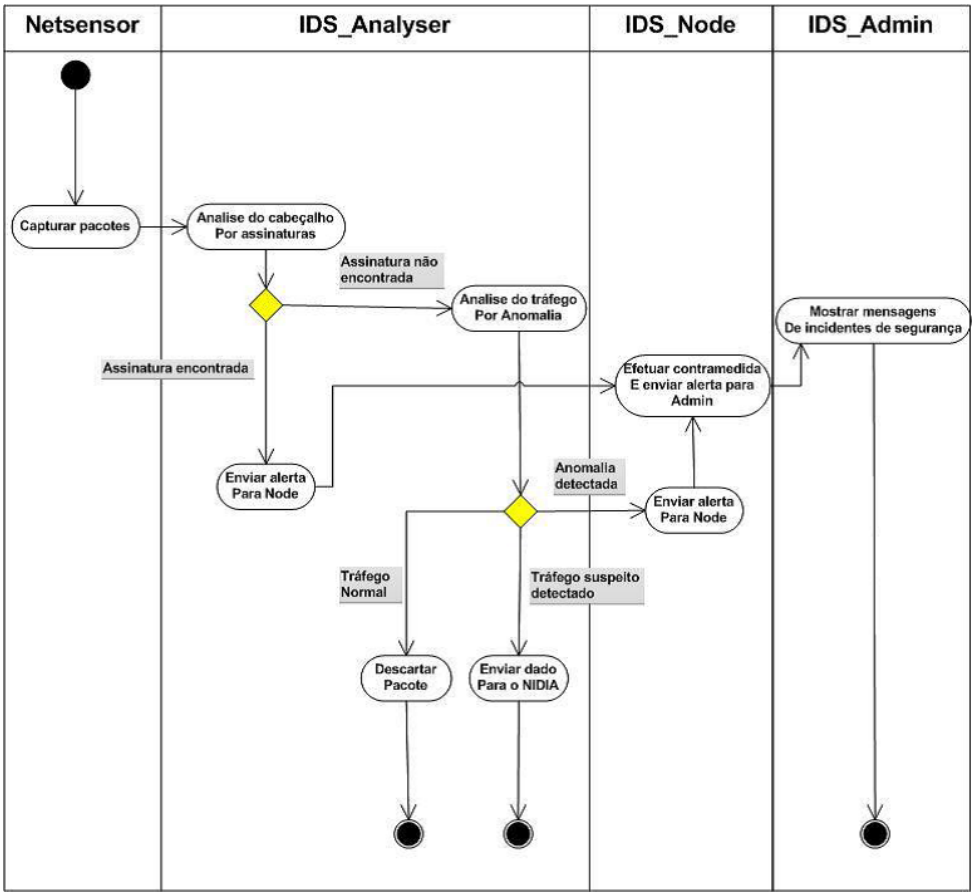

Figura 3.4: Diagrama de atividade do EICIDS [18].

- Uma anomalia pode ser encontrada, e um alerta é enviado ao *Node*, que encaminha um alerta ao IDS_Admin o elemento central do EICIDS, e posteriormente acionará uma contra medida.

- É detectada uma suspeita de ataque. Neste caso os pacotes considerados suspeitos podem ser enviados a um outro SDI, como o NIDIA [33].

O componente IDS_Node é responsável por ativar as contra medidas e enviar relatórios de incidentes de segurança ao componente IDS_Admin. O IDS_Node recebe esses resultados coletados pelos analisadores. Por sua vez o IDS_Admin exibe as informações de incidentes e ações efetuadas ao administrador do sistema.

3.1.4 Comparação entre SDIs para nuvem e o EICIDS

Com base na análise comparativa apresentada por Dias (2013), sobre os trabalhos existentes de SDIs para o ambiente de computação em nuvem, chegamos a seguinte conclusão ao fazer uma análise do EICIDS e os trabalhos estudando pelo autor [18]:

- O EICIDS possui uma arquitetura centralizada (o IDS_Admin fica localizado no *hypervisor*);

- A forma de proteção é feita tanto por nó como por VMs (os componentes do IDS são instanciados para as VMs quando for necessário);

- Captura pacotes nas VMs (Essa captura é feita por meio do *Netsensor*);

- Sua localização é tanto nas VMs como no Nó físico (temos o IDS_Admin que fica no *hypervisor* e seus componentes localizados nas VMs passando todas as informações do ambiente virtual);

- Aumenta capacidade do SDI e ao mesmo tempo poupa recursos das VMs (na arquitetura do EICIDS existe um componente chamado IDS_Pool que é responsável pela verificação de VMs ativas ou não e através dessa informação o IDS_Node pode instanciar ou não os componentes para as VMs).

A Tabela 3.1 apresenta uma análise comparativa entre os sistemas de detecção de intrusão para o ambiente de computação em nuvem apresentado no capítulo 3 e o EICIDS.

Conforme observado nos trabalhos relacionados, existe um problema com a questão de tolerância a propagação do ataque e a tolerância de falhas por componentes. Os trabalhos analisados não proporcionam ações de resposta automática com base no conjunto de políticas, e nem evitam a proliferação de um ataque dentro do ambiente proposto por cada um.

Para contornar esses problemas a metodologia proposta conta com etapas que contornam ou amenizam boa parte dos problemas observados nos trabalhos relacionados.

A etapa de melhoramento da proteção a nível de máquinas virtuais, usa técnicas de proteção e bloqueio de intrusão, tendo por base os resultados das capturas de pacotes oriundos do EICIDS, que monitoram constantemente os recursos da nuvem e acompanhe a expansão ou redução dos recursos disponibilizados pela nuvem.

A metodologia proposta se baseia em técnicas de *Safety-bag checks*[1] e IPTables[2] , que contribuem para a obtenção de tolerância a falhas em nível de máquinas virtuais. Para descrição de usuários maliciosos foi utilizado resultado da análise de pacotes capturados no ambiente. Por fim, foram aplicadas técnicas para a proteção a nível de máquinas virtuais, ou

[1]Neste processo, é feito o bloqueio de comandos que não satisfazem as características de segurança.

[2]Funciona através da comparação de regras para saber se um pacote tem ou não permissão para passar.

Tabela 3.1: Comparativo entre SDI baseados em VM para Computação em Nuvem e o EICIDS

	Arquitetura	Forma de proteção do SDI	Captura de pacotes nas VMs	Localização	Uso da elasticidade	Tolerância a Falhas
GCCIDS	Distribuído	Por nó	Não	Nó Físico	Não	Não
IDSCCE	Centralizado	Não foi definido pelo autor	Não	Nó Físico	Aumenta a capacidade do SDI	Não
IDSaaS	Centralizado	Por serviço	Sim	Nas VMs	Aumenta a capacidade do SDI	Não
MA	Não foi definido pelo autor	Por nó	Sim	Nó Físico	Aumenta a capacidade do SDI	Não
CIDS	Distribuído	Por nó e VM	Sim	No Físico e nas VMs	Aumenta a capacidade do SDI	Não
EICIDS	Centralizado	Por nó e VM	Sim	No Físico e nas VMs	Aumenta a capacidade do SDI e Poupa os recursos da nuvem	Não

seja, um eco é emitido a todas as VMs, evitando a propagação do ataque, ou seja, bloqueando o atacante e todas as aplicações que estavam sendo utilizadas por ele.

Para contornar o problema de tolerância a falhas do componente, a metodologia dispões da técnica de replicação e regras de bloqueios.

3.2 Conclusões

Neste capítulo foram apresentados o Projeto EICIDS, suas características, sua arquitetura e sua funcionalidade.

Na nuvem há uma preocupação com a segurança das VMs e dos componentes utilizados para prover a proteção, pois uma vez comprometidos podem levar o ambiente a ter sérios problemas. Assim, há um aumento da necessidade de tolerância a falhas para alcançar a proteção de máquinas virtuais, das aplicações que rodam nelas e dos componentes que a integram. Por essas razões, quanto mais precisa e rápida for a detecção, mais eficiente serão as contra medidas. A metodologia proposta aqui contribui para tal.

No próximo capítulo, um mecanismo de tolerância a falhas para o EICIDS é proposto.

4 Mecanismo Proposto

O objetivo deste capítulo é apresentar o mecanismo de tolerância a falhas proposto ao EICIDS. Primeiramente, os objetivos e requisitos do mecanismo são apresentados. Em seguida, a arquitetura do mecanismo e seus componentes são detalhados, junto com as detecções de falhas abordadas pelo mecanismo.

4.1 Objetivos

Um sistema torna-se vulnerável quando apresenta condições favoráveis de erros que, quando notadas por um atacante, podem resultar em grandes danos para o ambiente. Segundo [14], um atacante tenta utilizar a vulnerabilidade de um sistema, para conseguir executar ações maliciosas, tipo: invadir um sistema, acessar informações confidenciais, disparar ataques contra outros usuários ou até mesmo tornar o serviço do sistema inacessível.

A computação em nuvem, apesar de todas suas vantagens, ainda preocupa muitos usuários com relação à segurança e à privacidade das informações colocadas nesse ambiente. A nuvem, de uma forma geral, resume-se em um *pool* de recursos onde os usuários colocam suas informações. Porém, tal característica a torna um alvo propício a ataques tanto de usuários internos como externos. Um ataque bem-sucedido pode levar a nuvem a ter sérios problemas.

A nuvem, além de fornecer seus serviços, deve ter boas práticas de segurança, já que falhas podem vir a ocorrer, porém as consequências podem ser evitadas com o uso adequado de mecanismo de tolerância a falhas. Diante disso, uma invasão ou até mesmo a parada de uma máquina virtual dentro da nuvem podem ser resolvidas com o uso de mecanismos eficientes.

O objetivo do mecanismo de tolerância a falhas aqui proposto é detectar falhas no EICIDS, provendo uma adequada recuperação para os seus diferentes tipos detectados. Com isso, garantimos que o EICIDS continue fornecendo seus serviços, mesmo que falhas ocorram.

No mecanismo foram propostas duas técnicas: o monitoramento e a replicação para prover a tolerância a falhas. Elas têm os seguintes objetivos no EICIDS:

- Detectar VMs inativas;

- Detectar usuários com ações maliciosas;
- Detectar *crash* de aplicações (do próprio SDI);

As medidas tomadas e as técnicas utilizadas em um sistema, quando aplicadas de maneira correta, são critérios que definem se um ataque será ou não bem-sucedido. No processo de detecção de falhas, tais técnicas deverão ser aplicadas de forma precisa e completa, e a medida tomada para a recuperação deverá ser apropriada para cada tipo de falha.

4.2 Requisitos

A computação em nuvem por si só exige uma alta confiabilidade e alta disponibilidade, portanto somente a prevenção e a remoção de falhas não serão suficiente. Logo, um ambiente que deseja possuir essas características, citadas anteriormente, deve ser construído usando técnicas adequadas de tolerância a falhas, isso para que tenha seu funcionamento correto mesmo na ocorrência delas.

Em [52], apresentam-se dois pontos essenciais na hora da escolha de um mecanismo tolerante a falhas, que são: as características especiais da aplicação e as suas exigências quanto à confiabilidade e à disponibilidade.

Alguns requisitos foram definidos para o mecanismo de tolerância a falhas proposto. São os seguintes:

- O mecanismo deve ser, por si só, tolerante a falhas. Ou seja, garantir que seus serviços permaneçam em execução mesmo diante de possíveis falhas;
- O mecanismo deve ser capaz de dar respostas autônomas. Diante de um ataque o mecanismo deve alertar todo o ambiente sobre o problema, garantindo assim que a ação maliciosa não seja bem sucedida, e que seus componentes respondam a qualquer problema;
- O mecanismo de ser *self* resiliente, passar por uma situação difíceis (falhas, erros, defeitos), e mesmo assim conseguir fazer o que fazia antes sem perder o seu foco;
- Como o ambiente trabalhado é um ambiente flexível e escalável, logo o mecanismo de tolerância a falhas também deve ser flexível e escalável para suportar o crescimento do sistema em tamanho e em complexidade. A medida que uma VM é solicitada parte dos componentes de tolerância a falhas são alocados juntos com ela;

- Deve haver mudanças mínimas, ou seja, o EICIDS deve permanecer com sua forma original. O mecanismo aqui proposto permaneceu com a estrutura do EICIDS, deixando toda seus componentes com suas funcionalidades, ajustando somente os novos componente para uma melhor proteção do ambiente [18].

4.3 Arquitetura

A arquitetura proposta nesta dissertação tem como objetivo prover tolerância a falhas do EICIDS. Dessa forma, foi proposto um mecanismo de tolerância a falhas para o EICIDS que é um sistema de detecção de intrusão elástico para nuvem, por meio de Componentes de Tolerância a Falhas (CTF) e uma Base de Dados de Usuários Legítimos da Nuvem (BDUL). Os Componentes de Tolerância a Falhas encontram-se na camada de administração, pois é responsabilidade dessa camada manter a integridade do sistema.

Os CTFs são: componente sinal, componente de monitoramento e proteção do IDS_Admin, elemento principal do EICIDS, responsável pelo gerenciamento do processo de detecção de intrusão, e componente de replicação. Esses componentes cooperam entre si para o bom desempenho do mecanismo. A Figura 4.1 apresenta a arquitetura do mecanismo proposto e seus componentes.

Na Figura 4.1 é apresentada a arquitetura do mecanismo de proteção do EICIDS, que foi aplicada a cada nó da nuvem. Ou seja, a cada VM que existir, haverá os componentes App_Monitor e um IP_Analiser, que serão abordados na seção 4.5.1, e deverá haver também o elemento *Send* do componente sinal. O EICIDS, por meio de seus componentes, coleta informações das máquinas virtuais efetuando a detecção de atividades suspeitas e posteriormente enviando alertas ao módulo de controle IDS_Admin.

Os elementos do mecanismo proposto integrado na arquitetura do EICIDS são acionados e contramedidas com relação ao tipo de ataque e possíveis invasões são bloqueadas. O CTF é dividido em três componentes e cada componente por sua vez possui seus elementos. Essa divisão é dada conforme apresentada na Figura 4.1, onde se tem: componente sinal (*Send*, *Receive* e Chamada), componente de monitoramento e proteção do IDS_Admin (*Start*, Monitor, *Analyser*, Ação e Sensor) e componente de replicação (Monitor, *Instance* e Espelho).

Neste modelo, em nível de VMs, temos um conjunto de máquinas virtuais rodando em uma infraestrutura de nuvem. Cada máquina virtual contém um analisador de IPs (Ip_Analyser) e um monitor de aplicações (App_Monitor), pois por meio deles o

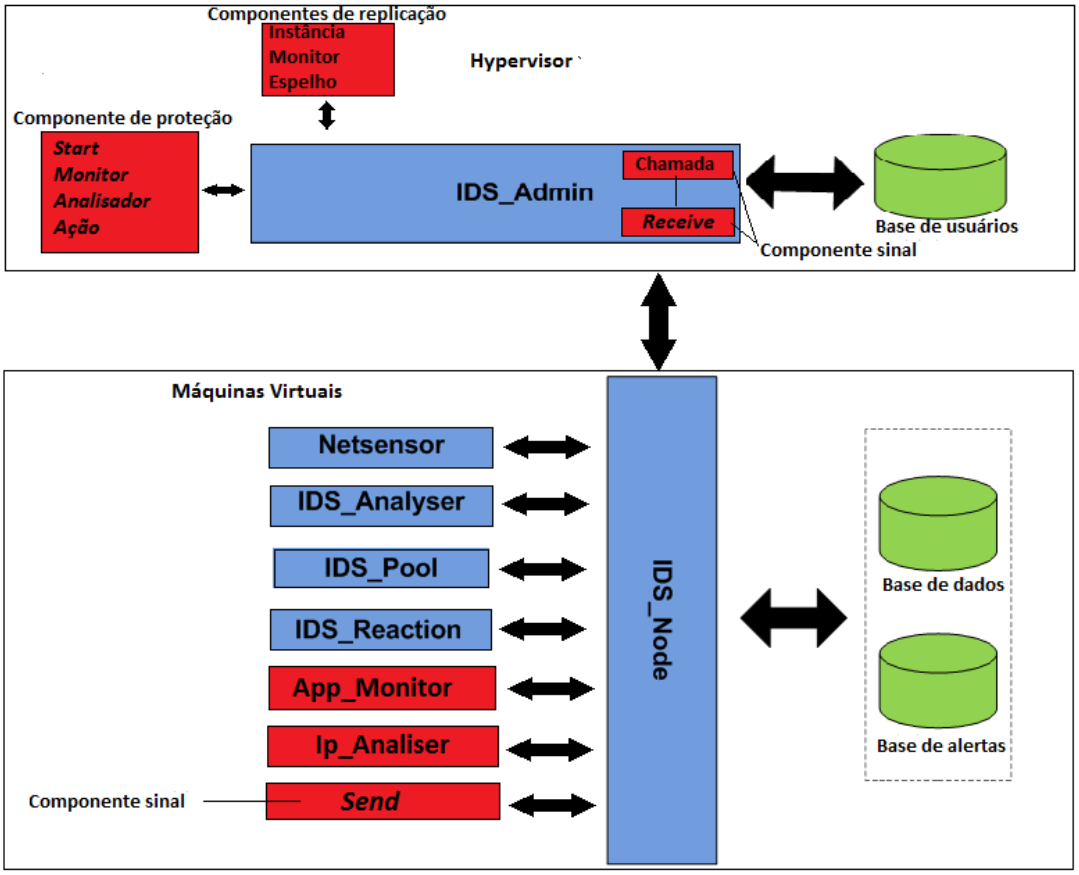

Figura 4.1: Arquitetura do modelo proposto na visão do *hypervisor*.

monitoramento, a detecção de intrusão e as regras de bloqueios são gerados ainda nas VMs. Diante de uma ação maliciosa, os componentes bloqueiam o usuário, e um eco é emitido para as demais máquinas virtuais. Vale ressaltar que cada VM possui esses dois componentes.

As características e as responsabilidades de cada componente do mecanismo são apresentadas, com detalhes, nas próximas seções.

4.4 Ambientação do Mecanismo de tolerância a falhas

A metodologia de construção do mecanismo proposto foi feita tendo por base os ambientes open-source de infraestrutura como serviço e o *EUCALYPTUS* [34] utilizando o *hypervisor* KVM [31], onde existe um ponto de entrada principal da nuvem chamado de *cloud controller* e os componentes de execução de máquinas virtuais chamados de *node controller*. Assim, um módulo central é encarregado de criar e iniciar módulos responsáveis por monitorar o comportamento das VMs em cada *node controller*, recebendo alertas e solicitações de contramedidas.

4.5 Componentes de tolerância a falhas

A seguir serão apresentados os componentes que formam o mecanismo de tolerância a falhas proposto, bem como seu funcionamento.

4.5.1 Componentes IP_Analyser e APP_Monitor

A Figura 4.2 mostra o modelo conceitual desses dois componentes do mecanismo de tolerância a falhas para o EICIDS que atuam na camada das VMs. Cada VM protege a si e as demais, gerando regras para bloquear o ataque e emitindo um eco às demais VMs do ambiente. Caso o ataque parta do ambiente interno e se for recorrente, a VM que efetuou o ataque será derrubada. O EICIDS trata o ambiente e o reconhece através de seus módulos espalhados, assim cada VM da nuvem conterá um módulo instalado, logo toda VM que não caracterizar um módulo válido será derrubada. A Figura 4.2 apresenta esses dois componentes do CTF.

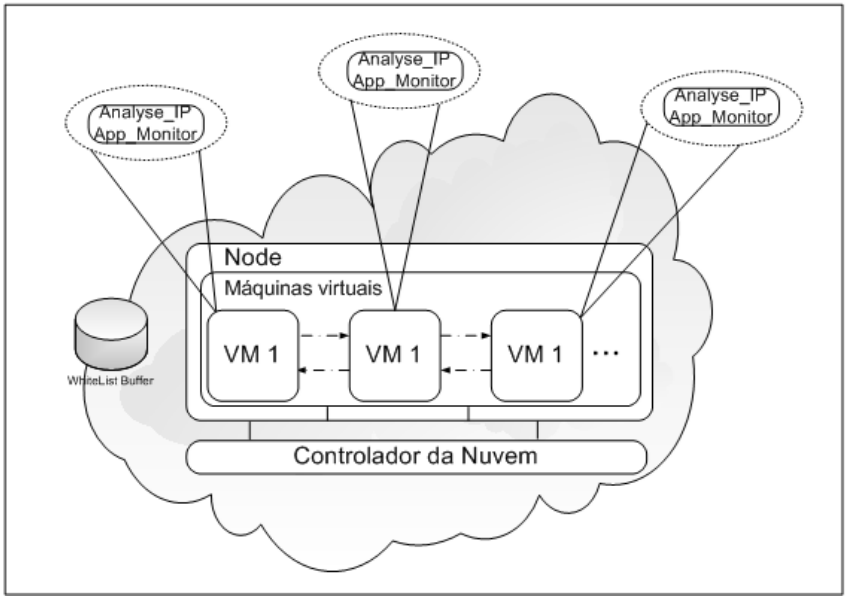

Figura 4.2: Modelo proposto nas VMs: IP_Analyser e APP_Monitor

Neste modelo, temos um conjunto de máquinas virtuais rodando em infraestrutura de nuvem. Cada máquina virtual contém um analisador de IPs *Ip_Analyser* e um monitor de aplicações *App_Monitor*.

O Analisador de IP *Ip_Analyser* faz a verificação dos IPs circulantes no ambiente e analisa-os para saber se fazem parte do catálogo. Caso não façam, ele gera uma regra de bloqueio se for um usuário externo, mas, caso seja um usuário interno com permissão, é descartado. Mas se o IP fizer parte do broadcast, é feita uma verificação para saber até

onde tem permissão de acesso. Se for permitido, ficará normalmente no ambiente (ou seja, irá circular sem problema algum no ambiente, pois é visto como tráfego normal), caso contrário será bloqueado também e a partir daí um eco é emitido para todas as demais VMs do ambiente. Na Figura 4.3 esse processo pode ser visto por meio do diagrama de atividades, usando a notação UML.

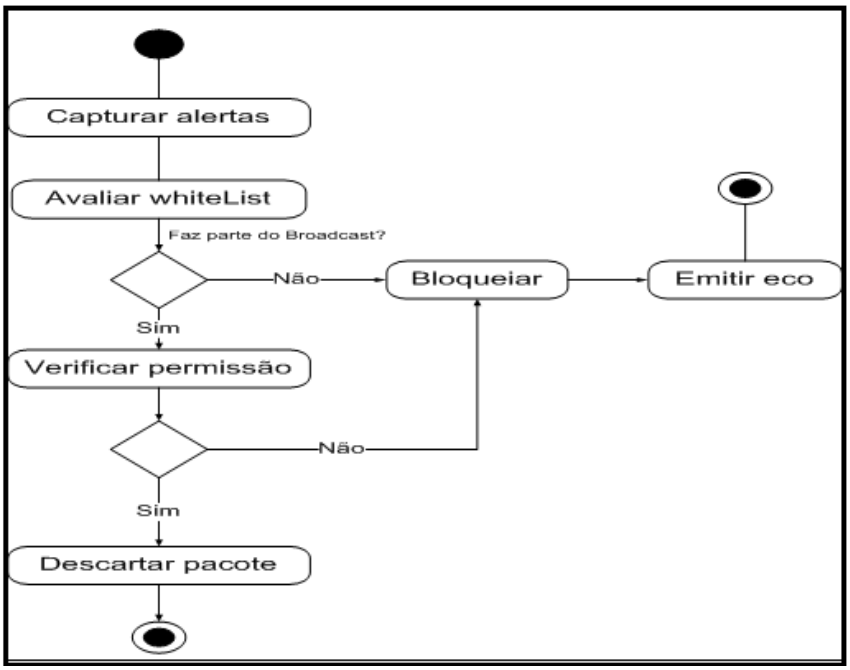

Figura 4.3: Diagrama de atividade IP_Analyser

O verificador de aplicações (*App_Monitor*) faz uma captura das aplicações em execução e também de usuários ativos na VM. Essas informações obtidas servem para comparar com os dados do catálogo, que contém a relação de usuários e aplicações que cada um pode acessar. Depois dessa verificação, se o usuário que interagiu com uma aplicação tiver permissão para executá-la, utilizará normalmente os recursos, mas, caso contrário, ele (usuário) será bloqueado e todos os processos (aplicações) dele serão destruídos, e um novo logout será feito. Esse elemento (*App_Monitor*) pode ser observado na Figura 4.4 por meio do diagrama de atividades.

4.5.2 Componente Sinal

Esse componente tem um papel muito importante no mecanismo proposto, pois ele verifica quais processos ou componentes estão em execução, ou seja, monitora o sistema. Verifica também se a comunicação entre esses componentes não foi corrompida. O componente sinal é dividido em três elementos: *Send*, *Receive* e Chamada. Esse monitoramento permite

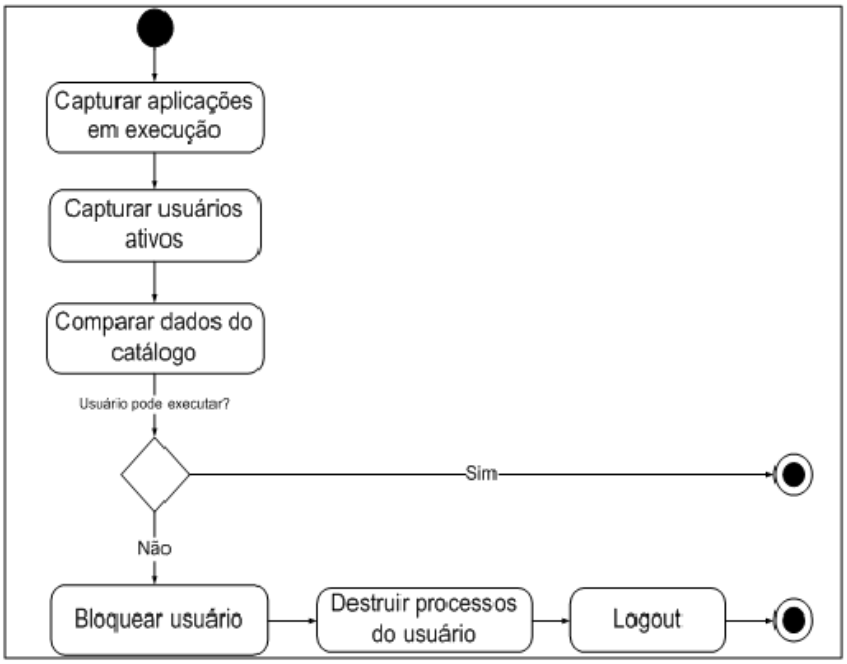

Figura 4.4: Diagrama de atividade App_Monitor

descobrir quais VMs estão ativas, ou seja, executando algum processo. Essa identificação é feita por meio de uma chave única, que serve para verificar se há comunicação entre os processos válidos do EICIDS ou qual não possui a chave de identificação. Como a chave é setada no código do programa, ou seja, codificada, apenas os nós do EICIDS que a conhecem podem transmiti-la. Isso garante que o elemento componente sinal esteja recebendo dados e conectado a uma VM que esteja rodando os componentes do EICIDS.

O EICIDS guarda consigo um arquivo de lista onde está registrada cada VM do ambiente, sendo de responsabilidade do administrador da nuvem a inclusão do nó no registro chamado *whitelist*. Para que uma VM não seja isolada do ambiente, cada nó do EUCIDS, exceto o nó administrador, é munido de um módulo de latência que foi implementado como uma *thread* de processo desse modo, ficando ativa enquanto o processo existir e enviando uma chave que apenas esse módulo conhece para o IDS_Admin.

O monitoramento das VMs com os componentes do EICIDS permite determinar se elas são autorizadas ou não. Na presença de uma identificação não autorizada, regras são geradas por meio dos componentes do mecanismo proposto. Quanto à classificação, entende-se que uma VM conceituada como maliciosa é aquela cuja execução e/ou requisição de ações não possuem autorização e suas ações poderão corromper as demais VMs e o *hypervisor*.

Elemento *Send*

É um elemento do componente sinal, implementado nas VMs, que tem a função de garantir a existência do nó. É responsável por enviar o sinal ao *Receive* que se localiza no *HYPERVISOR*. Por meio da porta 1020, é equipado com um temporizador que faz com que a rotina de envio da chave de segurança ocorra a cada 5 segundos.

Elemento Receive

Este elemento do componente sinal fica no *HYPERVISOR*; é responsável pelo recebimento e autenticação da chave de cada nó. Caso a chave esteja incorreta, a VM é direcionada para o elemento de tratamento de VMs, que será apresentado nos próximos tópicos.

No *Receive* existe um mapa que insere cada VM na *whitelist*, quando uma VM envia um sinal de latência e a chave estiver correta, o *Receive* marca 1 no mapa . Assim, para a VM que responder a cada 5 segundos, é realizada uma contagem de marcações no mapa. Caso a VM tenha marcação 0, ela é direcionada para o módulo *HYPERVISOR* que implementa o tratamento das VMs. Com isso o mapa é todo zerado novamente.

Elemento Chamada

Responsável pelo monitoramento do componente sinal, avalia a comunicação dos processos do EICIDS e aciona o componente de replicação.

O componente sinal do mecanismo de tolerância é mostrado na Figura 4.5 através do diagrama de sequência, usando a notação UML [22]. Nela podemos observar como esse componente se comporta.

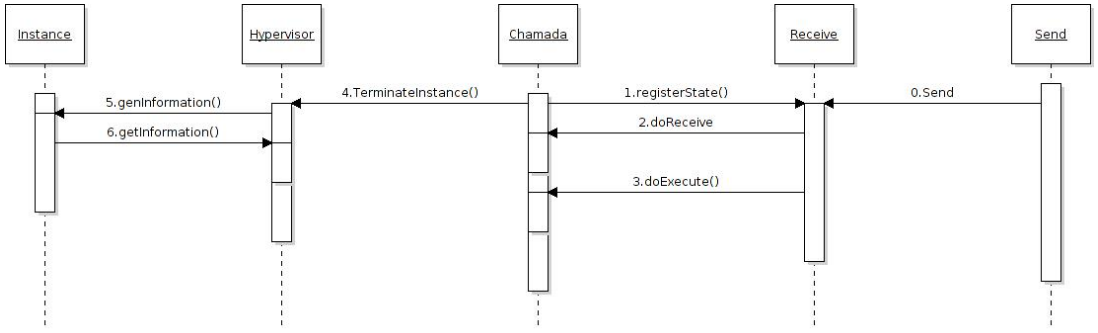

Figura 4.5: Diagrama de sequência do componente sinal.

O monitoramento dos processos feitos por esse componente é interno. A figura 4.5 apresenta os passos dessa detecção, que serão descritos:

(i) O *Send* fica mandando informações ao *Receive* a cada cinco segundos;

(ii) Ele (*Receive* com o auxilio do elemento Chamada) checa se o processo está vivo e se a chave de comunicação é autêntica para garantir que é um processo válido;

(iii) Se o processo está ativo e for válido, continua a monitoração feita pelo elemento Chamada;

(vi) Se o processo não estiver ativo ou não for autorizado, uma mensagem é enviada para o IDS_Admin que possua vez faz uma chamada do componente de replicação, mas específico ao elemento *Instance* alertando que uma VM precisa ser desativada e uma outra réplica dessa VM deve ser ativada.

4.5.3 Componente de monitoramento e proteção do EICIDS

O componente de monitoramento e proteção do EICIDS é responsável pela análise das informações coletadas pelo NAST[1], que captura pacotes e salva os dados em arquivos.

Esse componente tem uma visão geral da comunicação do sistema, pois recebe os pacotes de informações que são enviados ao hypervisor do meio externo. Isso garante a proteção do hypervisor contra ameaças externas. Os componentes são: *Start*, Monitor, *Analyser*, Ação e Sensor. Cada um desses elementos será descrito na seção 5.2.3.

4.5.4 Componente de Replicação

O componente de replicação é responsável pela organização das informações capturadas (dos ataques e das VMs) e pode tomar decisões de replicar VMs ou remover VMs, por meio dessas informações capturadas. Ele faz a identificação da VM que sofreu o ataque e que necessita ser replicada. Vale ressaltar que as informações são oriundas das VMs e do IDS_Node. Esse componente é formado pelos elementos: *Instance*, monitor e espelho.

[1]NAST: é um *packet sniffer* e um analisador de LAN baseado em *libnet* e *Libpcap*.

Instance

Executa os comandos de interação com o eucalyptus, por meio de sub-rotinas (atuadores) que irão capturar as informações do ambiente e registrá-las em arquivos a cada execução da classe. Essas informações são compostas por: o número de instâncias em execução, o identificador de cada instância (ID instâncias), o identificador de imagem de cada uma das instâncias e o IP interno e externo.

Minerador

É responsável por interpretar as informações oriundas das VMs, usando como base o modelo gerado pela classe instância. Segundo os ataques fornecidos pelo IDS_Admin, ele inicia uma réplica da instância à qual o ataque foi direcionado, sendo reconhecida pelo modelo e por meio dele obtém a imagem que representa a instância atacada. Uma vez que a VM foi identificada e sua imagem localizada, é lançada uma réplica que contém todas as aplicações registradas e configurações iniciais. Quando a réplica se torna presente no ambiente, com o IP interno da Vm comprometida, é iniciado o processo de terminação da VM antiga. Os arquivos da VM antiga encontram-se no backup incremental no *hypervisor*, quando solicitado são enviados para réplica.

Espelho

É responsável pela cópia incremental (o que torna muito mais rápido) dos dados do usuário contidos na VM. Cada VM possui uma seção de disco no *hypervisor*, onde são salvas as informações de todos os usuários da VM. Após o processo de replicação executado pelo minerador, o espelho é chamado, e uma cópia dos dados daqueles usuários é colocada de volta nas VMs.

4.6 Funcionamento do mecanismo de tolerância a falhas

As características do EICIDS são descritas em [18]. EICIDS poupa os recursos da nuvem, liberando instâncias dos sensores nas máquinas virtuais de acordo com a alocação delas pelos usuários. O funcionamento da arquitetura proposta lida com a análise feita nas VMs e no *HYOERVISOR*.

Em nível de VMs, temos os sensores do EICIDS (IDS_*Pool*, IDS_*Analyser*, IDS_*Reaction* e o *Netsensor*) que atuam no monitoramento das VMs, junto com dois componentes integrados chamados de IP_*Analyser* e App_*Monitor*, que serão vistos na seção 5.2. Dessa forma, alertas são enviados diante de uma ação maliciosa. A proteção e o bloqueio

de intrusão em máquinas virtuais são tarefas desses dois componentes, tendo por base os resultados das capturas de pacotes oriundos do EICIDS, que monitora constantemente os recursos da nuvem e acompanha a expansão ou a redução dos recursos disponibilizados pela nuvem.

O diagrama de atividade apresenta uma visão simplificada do funcionamento interno de um sistema, bem como suas funções e processos. Este tipo de diagrama é usado para modelar atividades, que podem ser um método ou um algoritmo, ou mesmo um processo completo. As atividades de um sistema são modeladas de acordo com o fluxo de ações executadas pelo sistema [22].

A Figura 4.6 ilustra o diagrama de atividades do componente sinal, com todas as atividades feitas pelas classes: Chamada, *Receive*, *Instance* e *Hypervisor*.

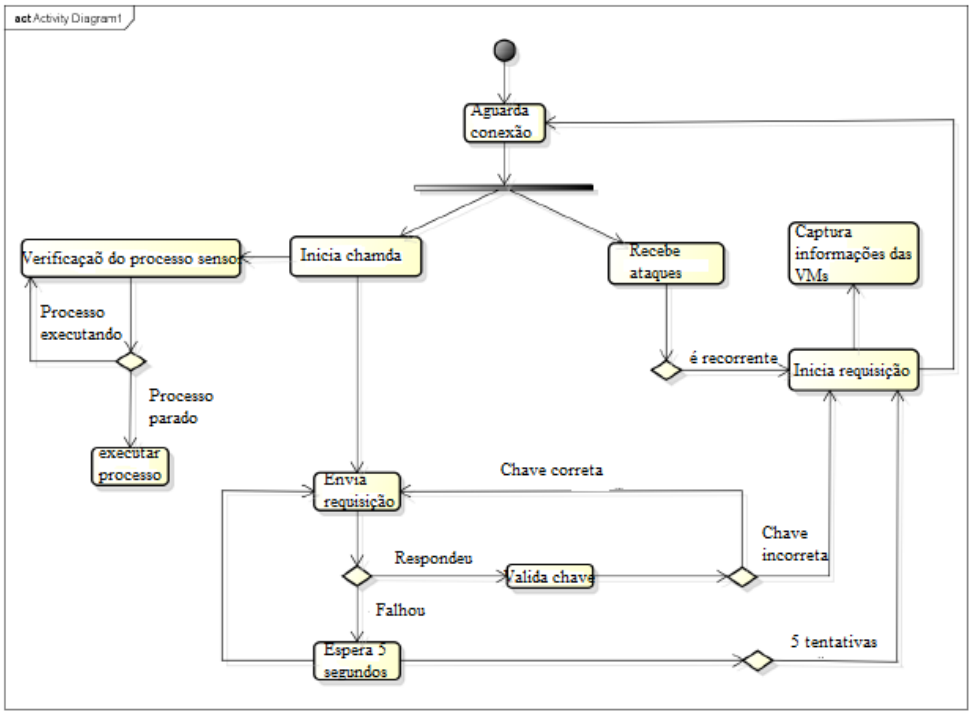

Figura 4.6: Diagrama de atividade - Componente Sinal

O CTF aguarda a conexão, logo em seguida é iniciado o fluxo de atividades para o mecanismo. Após a conexão, o fluxo de controle é divido em dois fluxos paralelos. A partir desse momento, os nós de ação: iniciar chamada e receber ataques serão executados paralelamente. Na ação receber ataques, é feita uma verificação, para identificar se o ataque é recorrente ou não. Se não for, não interessa ao sistema, mas, caso seja, ele inicia requisição que captura informações das VMs. Na ação iniciar chamada, uma verificação do processo sensor é feita para detectar se o processo está ou não executando.

A Figura 4.7 apresenta o diagrama de atividades, com as atividades feitas pelas classes: *App_Monitor, Ip_Analyser*, *Send* e o Espelho.

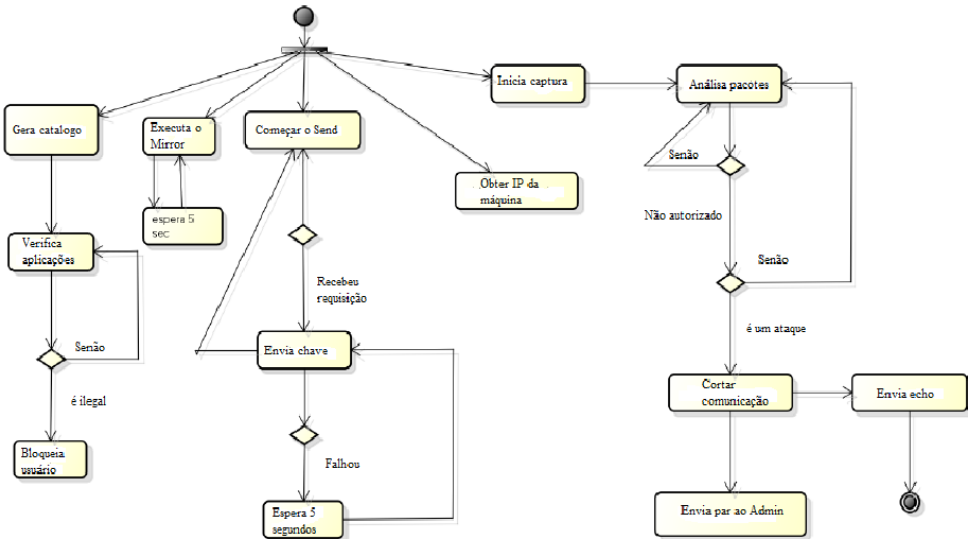

Figura 4.7: Diagrama de atividade: Detecção de ações maliciosas por meio dos elementos do CTF

O diagrama de atividade: Detecção de ações maliciosas por meio dos elementos do CTF onde o fluxo de controle é divido em vários fluxos paralelos. A partir desse momento, os nós de ação Gerar catálogo, Executar o Espelho, Começar o *Send*, Obter o IP da máquina e Iniciar captura serão executados paralelamente, já que todos se encontram em um nó de bifurgação-união[2]. O gerar catálogo verifica se as aplicações são ou não legais. E, se o analisar pacotes não possuir autorização, a comunicação é cortada e um eco é emitido às demais VMs do ambiente, as quais enviam ao IDS_Admin.

A Figura 4.8 apresenta o diagrama de atividades do processo de proteção e monitoramento do IDS_Admin. Isto é, as atividade desenvolvidas pelas classes: *Monitor*, *Analyser*, Sensor e Ação.

O diagrama de atividades do processo de monitoramento do IDS_Admin tem o fluxo de controle dividido em vários fluxos paralelos. Ao executar o NAST, o fluxo é dividido em dois e executado paralelamente. Com a inicialização paralela, os nós de ação Verificar o status dos processos e Analisar o log do NAST passarão a ser executados. O nó de Verificar o status dos processos faz uma análise para saber se os processos estão executando ou não. Caso não estejam, ele executa os processos. Já o Analisar o log do NAST, se o log não for autorizado, ele gera regras de bloqueio e corta a comunicação.

[2]bifurgação-união - é um nó de controle que pode tanto dividir um fluxo em dois ou mais fluxos concorrentes.

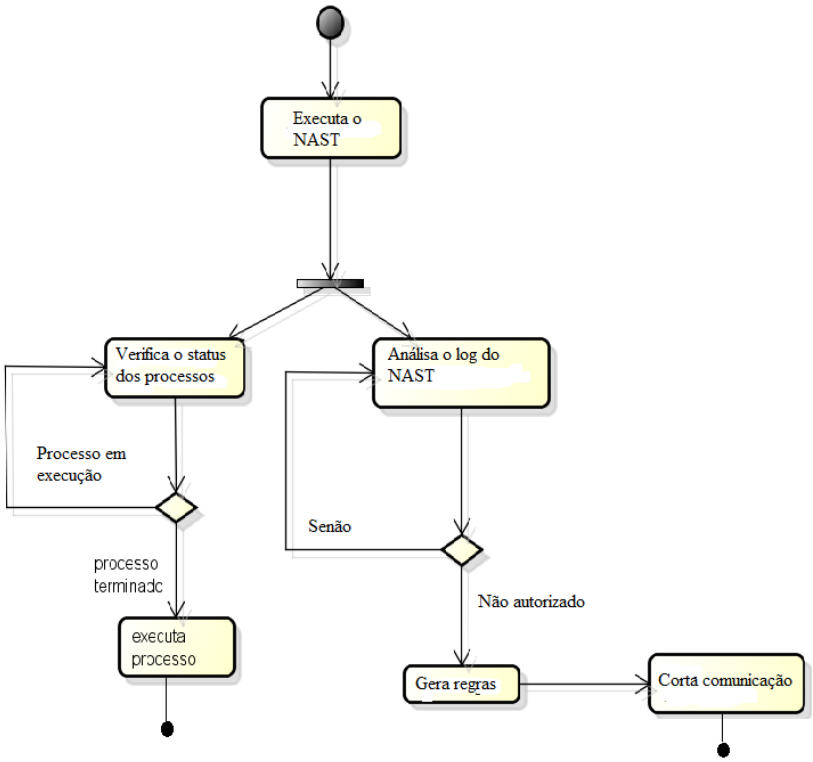

Figura 4.8: Diagrama de atividade: Processo de proteção e monitoramento do IDS_Admin

4.7 Análise comparativa

Conforme descrito no capítulo 2, quanto ao ambiente de Computação em Nuvem, existem inúmeras técnicas e resultados de implementações para prover tolerância a falhas a esse ambiente. Tais resultados levaram ao desenvolvimento dessa proposta (técnicas de tolerância a falhas propostas para o ambiente de Computação em Nuvem), visto que existe uma necessidade de tornar as técnicas de tolerância a falhas já existentes mais eficientes.

Neste capítulo, far-se-á uma análise comparativa entre os modelos tolerantes a falhas descritos no Capítulo 2 com base em: procedimentos aplicados para tolerar as falhas, tipos de falhas previstas, tolerância a falhas por VMs, tolerância a falhas por componente e tolerância a falhas para a propagação de ataques na nuvem.

A Tabela 4.1 ilustra comparação entre os modelos definidos no Capítulo 2 com base nos critérios listados anteriormente e o EICIDS.

Tabela 4.1: Comparativo entre os SDI tolerantes para nuvem e o EICIDS

Modelos proposto	Procedimentos aplicados para tolerar falhas	Tipos de falhas previstas	Tolerância a falhas por VMs	Tolerância a falhas por componente	Tolerância a falhas para a propagação de ataques na nuvem
AFTRC	Exclui o nó dependendo de sua confiabilidade.	Confiabilidade	Não	Não	Não
LLFT	Replicação	Crash-cost; Trimming fault.	Sim	Não	Não
FTWF	Replicação; Ressubimissão de job;	Dead line of work flow	Sim	Não	Não
VFT	Balanceamento de carga; Redução de trabalho; Redundância; Check Pointing.	Sobrecarga do Node; Disponibilidade.	Sim	Não	Não
HA-CIDS	Replicação; Pulsação.	Crash-cost; Trimming fault.	Sim	Não	Não
EICIDS	Replicação; Emissão de echo; Regras de IPTable; Exclui o usuários dependendo da confiabilidade.	propagação de ataques; Confiabilidade; Disponibilidade;	Sim	Sim	Sim

4.8 Conclusões

Neste capítulo foi apresentado o modelo proposto, seus objetivos, seus requisitos, sua arquitetura, seus componentes e seu funcionamento. A arquitetura do EICIDS proposta por [18] foi apresentada na seção anterior, pois esta proposta teve como base sua arquitetura e seus requisitos.

Os sistemas tolerantes a falhas se preocupam com todas as técnicas necessárias para permitir que um sistema tolere falhas de software ou hardware durante seu funcionamento.

Os provedores de serviços na nuvem buscam inovações tecnológicas que garantem proteção e privacidade das informações e dos dados de usuários.

Atualmente, há inúmeros modelos de tolerância a falhas que fornecem diferentes mecanismos tolerantes a falhas para melhorar o sistema. Todavia, ainda existem alguns desafios que precisam de alguma preocupação para cada tipo de problema. Por isso são projetadas medidas de segurança que auxiliam na diminuição de preocupações das entidades utilizadoras da computação nas nuvens, a fim de prover maior confiabilidade e

disponibilidade. Métodos de tolerância a falhas entram em jogo no momento em que uma falha entra nos limites do sistema. Dessa forma, técnicas de tolerância a falhas são usadas para corrigir falhas e permitir que o sistema permaneça em funcionamento mesmo diante de um erro. No próximo capítulo, será apresentada a implementação e os resultados dessa proposta.

5 Implementação Parcial e Resultados

O EICIDS foi implementado na nuvem *EUCALYPTUS*. O *EUCALYPTUS* é uma nuvem *open source* que usa infra estrutura computacional e de armazenamento.

Este capítulo descreve a avaliação do mecanismo proposto. Para isso, foi desenvolvido o protótipo do modelo da pesquisa. Com o objetivo de verificar o funcionamento do modelo proposto, foram implementadas técnicas, utilizando o *eucalyptus*, para melhorar a eficiência do EICIDS. Posteriormente, foram realizados testes no mecanismo do cenário de computação em nuvem. Em seguida, os resultados obtidos em cada componente do mecanismo de tolerância a falhas para o EICIDS foram detalhados.

5.1 Ambiente Proposto

Para a realização do trabalho, foram criados no LABSAC (Laboratório de Sistemas e Arquiteturas Computacionais) da UFMA (Universidade Federal do Maranhão) dois ambientes de testes que simulam uma infraestrutura de nuvem IaaS. Assim como o EICIDS, o ambiente de teste dessa proposta é similar ao usado pelo [22]. Para o ambiente, foi utilizado um microcomputador denominado CLOUDTF, no qual instalamos o sistema operacional Linux centOS, o software de virtualização KVM = *Kernel-based Virtual Machine*, a biblioteca *open source Libvirt* [3], o eucalyptus e demais dependências necessárias.

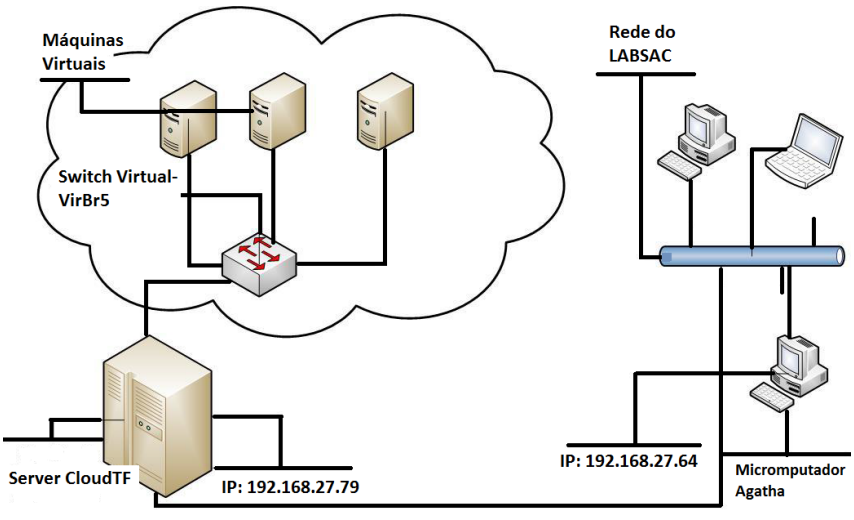

Figura 5.1: Ambiente montado para a realização dos teste

Na Figura 5.1, é apresentado esse ambiente de teste: o controlador central do ambiente e, sobre seu domínio, o conjunto de VMs e a rede externa.

A Tabela 5.1 apresenta a configurações dos computadores utilizados para a validação do mecanismo proposto.

Tabela 5.1: Configurações das máquinas no ambiente de Teste

Serve ClouTF	192.168.27.79	Core i7; 8GB RAM; 1TB de HD	Linux CentOS 12.03 server, EUCALYPTUS com os módulos cloud controller, cluster controller, storage controller e walrus
Agatha	192.168.27.89	Core i5; 8GB RAM; 500GB de HD.	Linux CentOS 12.03 server, KVM, EUCALYPTUS 3.01com o módulo node controller

5.2 Implementação do Protótipo

O principal objetivo da solução proposta é prover a tolerância a falhas ao EICIDS,para aumentar a disponibilidade do sistema e evidentemente a sua tolerância a falhas, reduzindo significativamente o esforço humano e os conhecimentos necessários para tal. O monitoramento do ambiente de rede das VMs deve ser realizado e, com as informações desse monitoramento, o EICIDS passará a acionar contramedidas de proteção. Este monitoramento se baseia em informações coletadas do tráfego direcionado às VMs em modo promíscuo, sendo que a detecção é oriunda da coleta feita pelo sistema EICIDS [18].

5.2.1 Diagrama de classe

As classes, métodos e os atributos implementados para o mecanismo proposto podem ser visualizados pelos diagramas de classes. Devido à extensão dos códigos, o projeto foi dividido em vários diagramas de classes. Os dados dos objetos são armazenados nos atributos das classes e as funções são realizadas pelos métodos ou operações implementados nas classes [22]. Na Figura 5.2 é apresentado o diagrama de classe do IDS_Admin proposto por [18], já com os componentes do mecanismo de tolerância a falhas.

O diagrama é composto pelas seguintes classes: Admin, Chamada, Receive, Hypervisor ou Espelho e Instância. Nesse diagrama podemos observar o relacionamento de 1 para 1 entre quase todas as classes apresentadas, exceto as classes Chamada e Receive, cujo relacionamento é de 1..*, ou seja, no mínimo um Receive deve estar

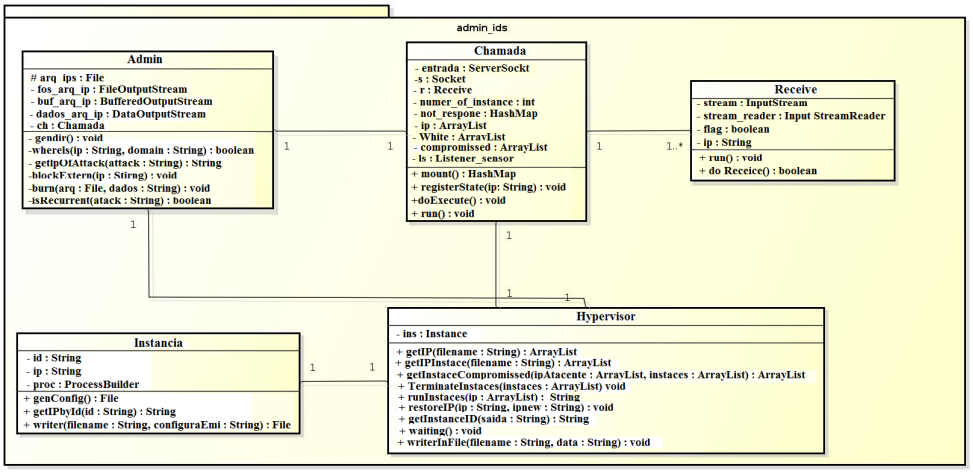

Figura 5.2: Diagrama de classe do IDS_Admin Modelo conceitual.

envolvido no relacionamento, podendo existir outros. No ambiente de nuvem, existirá apenas um IDS_Admin, que ficará responsável por gerenciar o IDS, mantendo uma visão geral da arquitetura e coordenando os vários IDS_Node, os quais ficam responsáveis pelo monitoramento em cada nó físico da nuvem.

O IDS_*Node* é apresentado na Figura 5.3 e com ele todos os componentes do EICIDS e mas os adicionais do mecanismo, como: *Send*, Espelho, *Ip_Analyser* e *App_Monitor*.

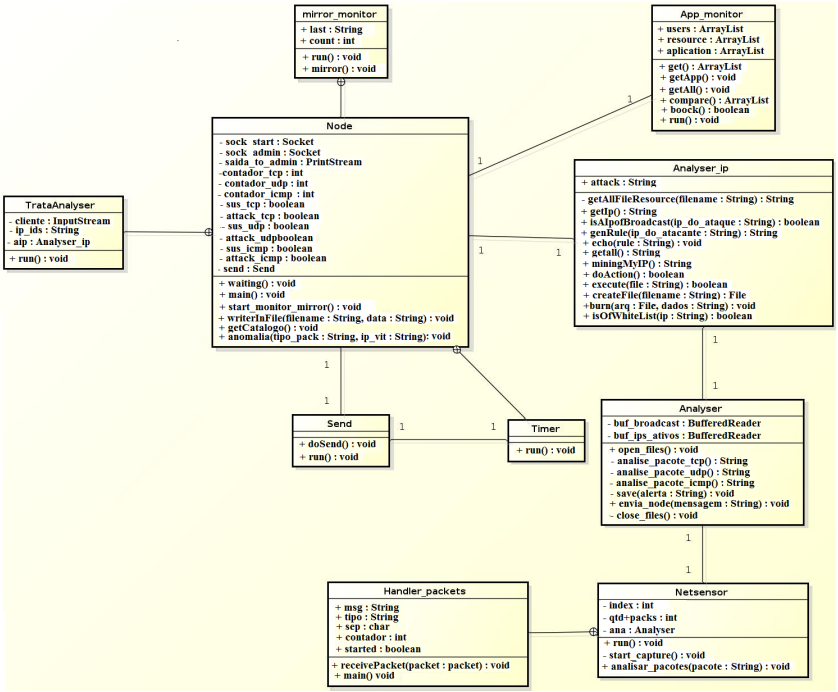

Figura 5.3: Diagrama de classe do IDS_Node Modelo conceitual

A Figura 5.3 mostra que a relação dos novos componentes inseridos no EICIDS com o *Node* é de 1 para 1, ou seja, para cada *Node* existe um elemento do componente de tolerância a falhas.

O componente de monitoramento e proteção do EICIDS, também chamado de sensor, tem uma visão geral do sistema, pois recebe os pacotes de informações que são enviados ao hypervisor, ou seja, todas as informações das VMs e do hypervisor passam pelos elementos desse componente. A Figura 5.4 apresenta o diagrama desse componente de tolerância a falhas.

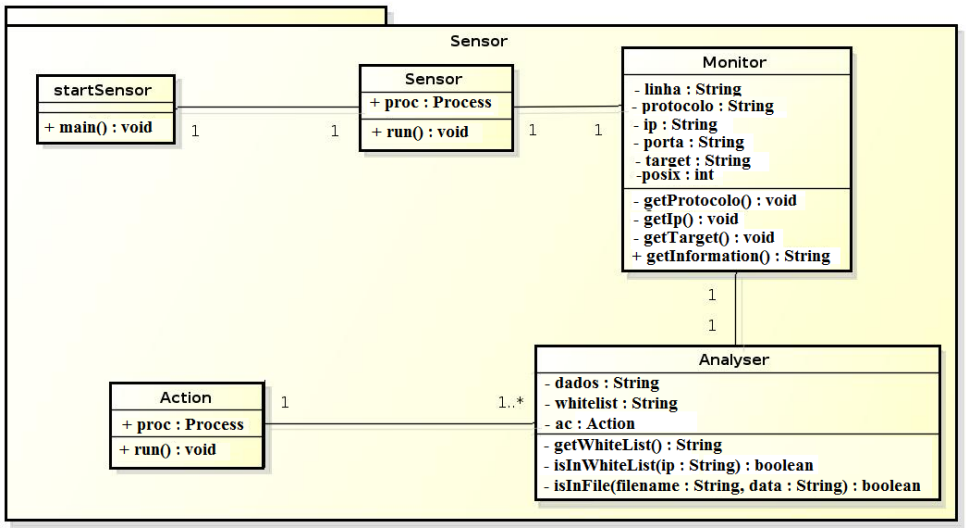

Figura 5.4: Diagrama de classe do componente sensor.

O diagrama da Figura 5.4 apresenta o componente sensor, que está dividido em: Monitor, *Analyser*, *Action* (Ação) e sensor. outra informação a ser considerada é a existência de, no mínimo, um *Analyser*, e no máximo, vários.

Para a demonstração da funcionalidade da solução proposta, foram implementadas parcialmente as classes apresentadas por meio do diagrama das Figuras 5.2, 5.3, 5.4. Foi também implementada a classe *Start*, responsável por inicializar o sistema. Os detalhes das classes implementadas são apresentados na próxima seção.

5.2.2 Implementação do componente sinal

O componente sinal apresentado na seção 4.5.1 emite sinais garantindo a existência das VMs e verifica as instâncias que enviaram o sinal. Essa verificação é feita para garantir que o usuário que pertence à nuvem está autenticado com uma chave válida. Esse componente se divide em três elementos: *Send*,*Receive* e Chamada. Serão apresentados os códigos de cada um

no decorrer desta seção. Todos os códigos obtidos por meio da implementação encontram-se nas apêndices.

O primeiro código a ser apresentado é o *Send*. A classe *Send* é responsável pela emissão de sinais para o *hypervisor*. Esse envio é para garantir que o nó está ativo.

O segundo código é o *Receive*, que é uma classe interna responsável por aguardar a resposta, oriunda do nó, e validar sua chave de comunicação. Caso não seja autêntica, aciona o elemento Chamada.

O terceiro elemento do componente sinal é o Chamada. Esse elemento inicia o *Receive* e seus processos. De acordo com os resultados obtidos pelo *Receive*, o elemento chamada irá tomar uma decisão, se irá ou não replicar a VMs. Outra de suas funções é verificar os processos do componente de monitoramento e proteção do EICIDS.

5.2.3 Implementação de monitoramento e proteção do EICIDS

Esse componente de monitoramento e proteção do EICIDS recebe o nome de Sensor_Hyp. O Sensor_Hyp é composto pelos seguintes elementos: Start, Monitor, Analisador, Ação e Sensor, que são mostrados, com detalhes, a seguir.

Start

O *Start* executa a verificação do status dos processos do componente IDS_Admin e do NAST. Com base nesse status, ele tomará a decisão de reiniciar ou não os processos. A sua função principal é instanciar a classe sensor, que é a classe principal do componente de monitoramento e proteção do EICIDS.

Esse elemento faz a interação com o ambiente usando sub-rotinas para capturar as informações dos processos, armazenado-as em arquivos temporários e usando como base a assinatura de execução de cada um dos processos ou o identificador de processos.

Monitor

O Monitor é responsável pela captura de dados do *log*.

O código gerado para esse componente tem a função de gerar um conjunto de informações, tais como: o protocolo, o IP, a porta e destino do usuário que abriu a interação.

Analyser

O *Analyser* verifica se o endereço do emissor é um endereço válido ou não, usando a *whiteList*, onde contém os IPs que são autorizados para acessar o hypervisor. Se

necessário,usando o *firewall IPTables*, gera uma regra de bloqueio para esse emissor, cria atuadores, os quais são repassados para o componente ação.

Ação

Verifica se a regra foi criada de forma correta, analisando com o uso de um padrão. E cria os atuadores, para aplicar as regras no ambiente e verifica se foram aplicadas.

Sensor

Captura as informações do log do NAST, assim que ele é modificado, pegando apenas uma linha por vez, padronizando-a para a utilização no monitor. O comando faz a passagem da informação para as demais classes.

5.2.4 Implementação do componente de replicação

Por meio desse componente, que faz a organização das informações capturadas, é feita a adição ou a remoção de uma VM, ou seja, a VM poderá ser interrompida e depois replicada.

Como final da etapa de reconhecimento do ambiente, com base nas informações encontradas por meio dos métodos da classe *instance*, é gerado um modelo do ambiente e de seus indivíduos que será utilizado, a fim de garantir que o uso do ambiente seja realizado por usuários legais. Isto quer dizer que cada VM que foi oficialmente lançada pelo administrador contará com um módulo do EICIDS.

5.3 Testes e Resultados Parciais

Para avaliar o IDS proposto, foram realizados ataques de negação de serviço (*Denial of Service* (DoS)) a partir de uma máquina virtual interna contra as outras máquinas virtuais existentes no ambiente. O EICIDS deve realizar contramedidas sempre que uma máquina virtual esteja sob ataque, tentando identificar o endereço IP do atacante e o tipo do ataque em andamento, pois tais informações ajudarão na proteção do ambiente de forma eficiente. O EICIDS possui em sua implementação métodos de ataques direcionados para a detecção de ataques de DoS[1], mas isso não impossibilita de detectar outros tipos de ataque, conforme [18].

[1]DoS: Denial of Service: Ataques de Negação de Serviços, consistem em tentativas de fazer com que computadores, servidores Web, por exemplo tenham dificuldade ou mesmo sejam impedidos de executar suas tarefas.

Para o estudo do comportamento do mecanismo proposto ao EICIDS, permaneceu o mesmo utilizado por [18]. Um atacante interno, por meio de um máquina virtual instanciada com o sistema operacional linux backtrack [44], foi inserido. O objetivo da instalação dessa VM foi realizar ataques contra as outras VMs do ambiente de teste, a fim de que as técnicas utilizadas por esse mecanismo inserido neste ambiente possam propiciar uma reação com medidas eficientes, sem a intervenção do administrador da rede.

Na máquina Agatha, está localizado o módulo IDS_Admin. No servidor de virtualização Server CLOUDTF, localizam-se os módulos IDS_Node, IDS_Analyser, Netsensor, *App_Monitor*, *Ip_Analyser* e o *Send*, que utilizam a interface de rede que o Netsensor captura, ou seja, os pacotes endereçados a *Virbr5 switch virtual*, pois ela permite que todos os tráfegos das máquinas virtuais possam ser observados. A partir da captura dos pacotes feita pelo Netsensor na *Virbr5*, os dados de alerta serão enviados ao IDS_Node 192.168.27.79, que é o próprio controlador da infraestrutura de virtualização e deverá manter uma comunicação com o módulo IDS_Admin 192.168.27.64, como demonstrado na Figura 5.5, que também traz os componentes do mecanismo de tolerância a falha proposto.

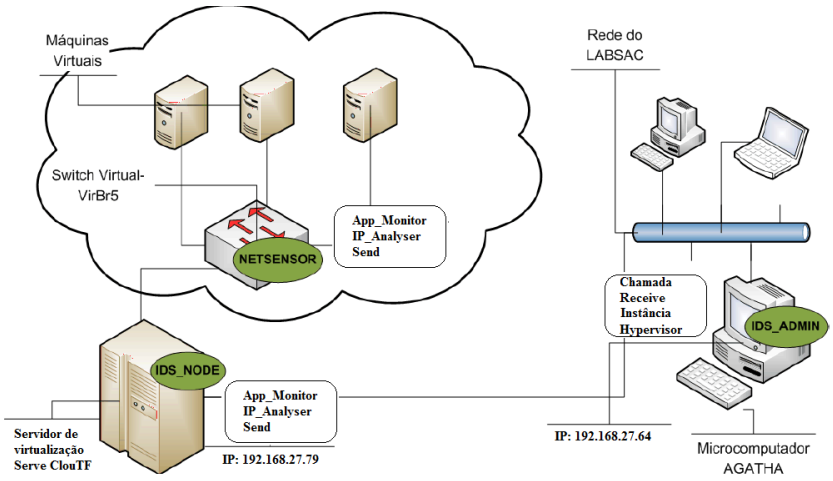

Figura 5.5: Componente do mecanismo no ambiente de teste

Na Figura 5.6, podem ser analisados na tela de console todos os arquivos do SDI.

Cada um dos arquivos de execução do SDI, conforme observado na Figura 5.6, consiste em rotinas *logs* eou regras para que o SDI possa armazenar informações e interagir com o ambiente.

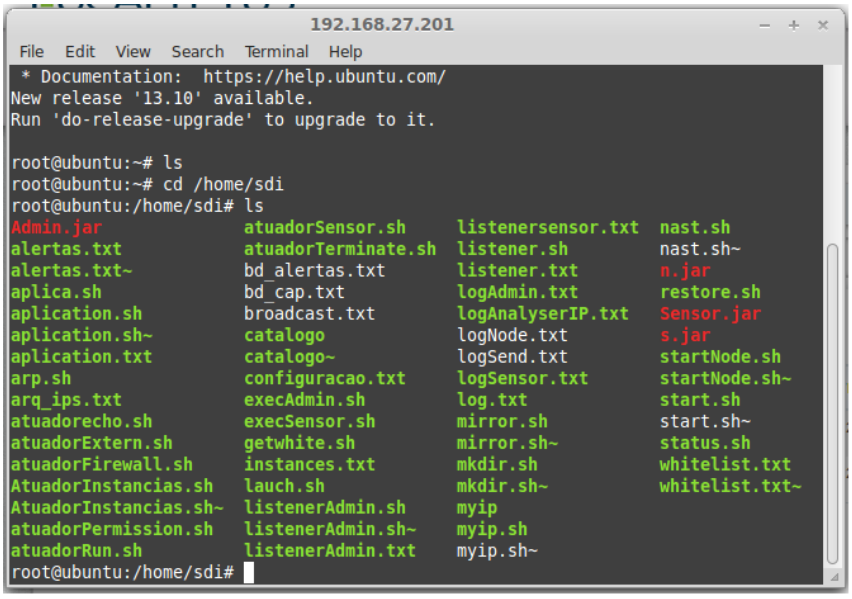

Figura 5.6: Arquivos de execução do sistema de detecção de intrusão.

Para realização dos testes, foi utilizado o programa Hping[2] [48]. Por meio dessa ferramenta, foi simulado o tráfego malicioso contra máquinas virtuais hospedadas no ambiente de teste preparado.

Seguindo a ordem de ataques feitos por [18], o primeiro ataque efetuado foi o (hping3 -V -c 1000000 -d 120 -S -w 64 -p 135 -s 135 –flood -a 192.168.27.64 172.31.254.16)[3]. Esse comando simula um ataque *SYN Flooding*, enviando pacotes TCP com *flag syn* habilitado. O endereço de IP 192.168.27.64 não pertence à faixa de endereços válidos do ambiente de teste. A sequência de requisições SYN é direcionada ao endereço 172.31.254.16.

Ao efetuar o comando a partir da VM de ataque, o ataque é prontamente identificado pelo IDS, e os componentes de replicação efetuam a medida de proteção em nível de VM emitindo um eco para todas as demais VMs do ambiente. A emissão do eco é demonstrada na figura 5.9. Antes de ele ser emitido, o *App_Monitor* e o *Ip_Analyser* fazem a verificação dos IPs e processos que interagem no ambiente, para obter informações sobre permissões. Para executar esse processo, as classe *App_Monitor* e *Ip_Analyser* examinam o catálogo de aplicações. Esse catálogo é gerado pelo Node. Este, quando realiza a primeira execução, gera um catálogo dos serviços para as VMs, automaticamente, com todas as

[2]Hping: é uma ferramenta de rede que envia pacotes TCP/IP modificados para algum endereço alvo em uma rede, servindo como um excelente teste para a segurança de uma rede.

[3]–V de verbose; -c 1000000 que envia um milhão de pacotes; -d 120 tamanho dos dados; -S que ativa o flag SYN; -w64 janela tcp; -p 135 –s 135 que indica a porta TCP de origem e de destino; –flood envia pacotes o mais rápido possível; -a mascara o endereço IP de origem

aplicações, os caminhos e os usuários que executam as aplicações. O *App_Monitor* pega o catálogo e, junto com os atuadores de ambiente,verifica-o e vai capturando do ambiente os processos que estão ativos naquele momento. Então, a verificação de todos os processos é feita e analisada para saber se estão de acordo com o catálogo gerado pelo *Node*. A figura 5.7 apresenta um trecho desse catálogo. Este é o cadastro de aplicações dos usuários que estão em execução.

Figura 5.7: Catalogo das aplicações dos usuários.

Por meio desse catálogo, é possível saber se as aplicações em execução estão permitidas. Caso o usuário não tenha permissão para executar a aplicação, o *App_Monitor* utiliza dois métodos para o bloqueio e a destruição das aplicações do usuário sem permissão.Esse processo pode ser analisado na figura 5.8.

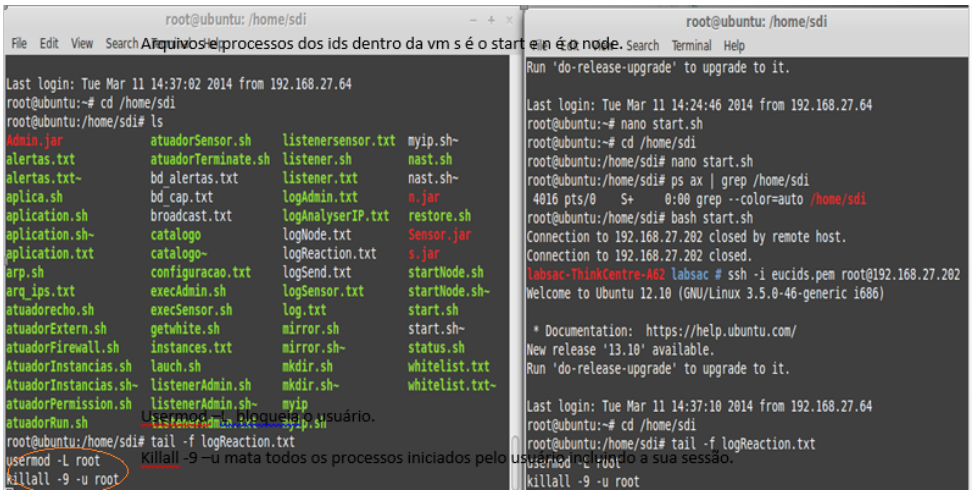

Figura 5.8: Medidas tomadas pelo App_Monitor.

Os comandos utilizados na geração da regra para bloquear o usuário e matar os processos executados por ele são respectivamente: *Usermod –L* e *Killall -9 –u*. Nessa etapa, o usuário será bloqueado e todos os processos iniciados por ele na seção serão mortos.

Isso é o resultado do *App_Monitor* e *Ip_Analyser*. E depois de usar o catálogo para registrar as aplicações permitidas e capturar periodicamente, ou seja, a cada 5 segundos, as aplicações em execução, é identificado o usuário que iniciou a aplicação e analisadas as

permissões. Por fim, verificando-se que ele não consta no catálogo, esse usuário é considerado ilegal, e a medida é tomada.

Os arquivos das VMs são copiados para o *hypervisor*, onde serão armazenados na pasta de backup, e esse processo é feito pelo *rsync*. Este é um *open source* utilitário que oferece rápida transferência de arquivos do tipo incremental [39].

Com base nas análise e verificações feitas, um eco é emitido para todas as VMs. A Figura 5.9 demostra a saída do eco.

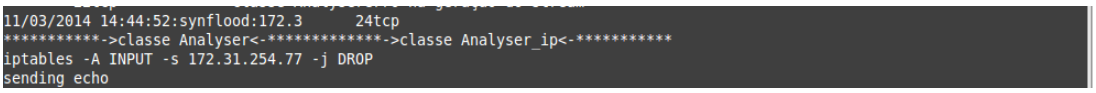

Figura 5.9: Emissão do eco as VMs

As regras aplicadas pelo eco são apresentada na Figura 5.10.

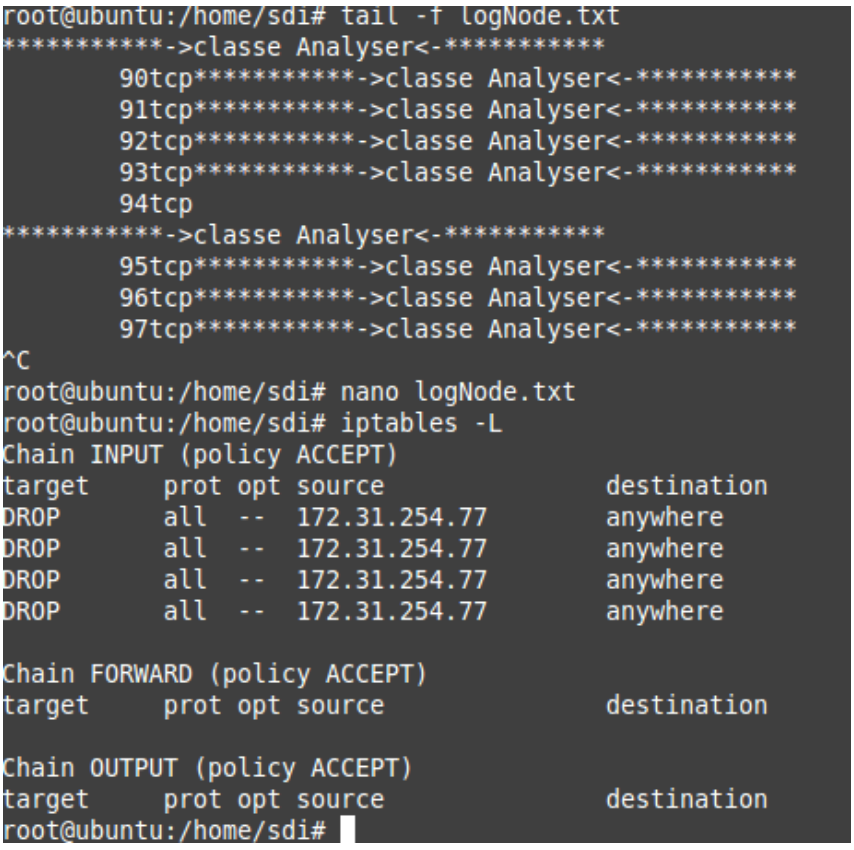

Figura 5.10: Regras do eco

Como saída, o eco emitido será: iptables -A INPUT -s 172.31.254.77 -j DROP. Onde:

- -A = acrescenta uma nova regra as existentes;
- INPUT = pacotes cujo o destino final é a própria máquina firewall;

- -s = especifica a origem do pacote. Este poder ser um host ou uma rede. Neste caso o 172.31.254.77
- DROP = não permite a passagem do pacote e abandona-o não dando sinais de recebimento.

O *HYPERVISOR* junto com a classe instância possuem uma lista com as VMs ativas no ambiente com seus identificadores e IPs. Cada um dos IPs dessas VMs emite sinal: o *log Send*.

Isso, significa que o *HYPERVISOR* enviou para aquela VM uma requisição. A VM recebe essa requisição e gera uma resposta para o *HYPERVISOR*, por meio do componente *Send* descrito na seção 5.2.2. Essa resposta é composta por uma chave que é analisada pelo componente Chamada dentro do hypervisor. O elemento Chamada trabalha em conjunto com o componente de replicação. Então, ele analisa a chave que foi recebida e analisa o sinal que a VM mandou. Caso os dados fornecidos pela VM não estejam de acordo com as regras do ambiente ou os processos não respondam, a replicação é feita.

Os componentes, diante de tal situação, entenderão que os processos das VMs, naquele momento, não estavam em execução. A execução dos processos dentro da VM é garantida por meio de um temporizador que fica tentando enviar sinais para a VM durante um intervalo de 5 segundos. Em cada segundo, ele faz uma nova tentativa e fica esperando a resposta por 5 segundos, o que leva esse processo a demorar 10 segundos por cada VM. Um sinal parte das VMs para IDS_Admin , o Trying, e significa que o elemento Chamada está enviando requisições para VM. Esse processo é apresentado na Figura 5.22.

Como demonstrado na figura 5.11, esse processo (*trying*) é repetido 5 vezes, na tentativa de abrir comunicação com aquela VM. Esta, apesar de estar registrada no ambiente, não possui os componentes do SDI sendo executados; dessa forma, para o mecanismo proposto, é conhecida como uma VM morta. Com isso, o processo de replicação tem que ser feito. O comando ***terminate*** <- 172.31.254.87 significa que a VM com IP 172.31.254.87 recebeu a ordem para ser terminada. Enquanto a VM 172.31.254.89 é lançada e se prepara para ser conectada, ao mesmo tempo outra instância deixa de responder, e o processo de terminação da VM acontece novamente. Perceba o ***Connected at*** 172.31.254.89 da Figura 5.11 significa que a VM 89 está conectada ao Admin. O ***i-E53646AF*** é o identificador da instância e as linhas que têm aquele formato se referem ao modelo do ambiente (arquivo configuração).

O componente de replicação fica dentro do Admin, são eles: Instância, Espelho (*Mirror*) e o Minerador. Essas classes são responsáveis por capturar as informações do ambiente de execução da nuvem. Esses componentes foram abordados na seção 5.2.3. A Figura 5.12

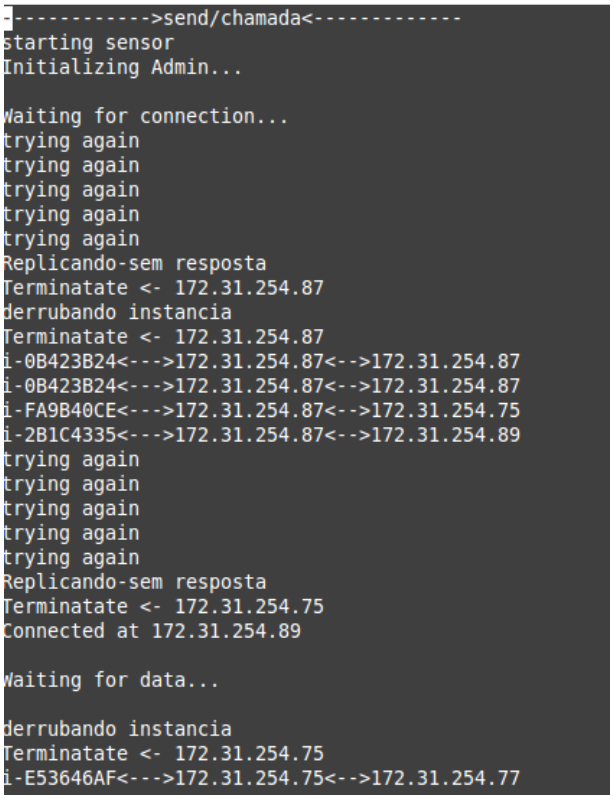

Figura 5.11: Send-Chamada

apresenta o momento em que o Espelho captura as informações da VM e manda para o *HYPERVISOR*.

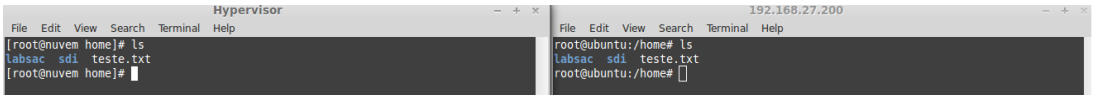

Figura 5.12: Execução do Espelho

A classe instância captura informações das VMs, e o Espelho aplica as ações contra aquelas que foram detectadas com ações maliciosas ou estejam sem resposta, ou então de acordo com os acontecimentos no ambiente, tipo: a chamada falhou 5 vezes ou o ataque foi da mesma VM várias vezes. Diante disso, o elemento chamada ativa a classe espelho. Esta classe irá pegar das VMs as informações, como: o ID - identificador e o EMI (nome da imagem que é gravada no disco da VM).

O EMI é o código de verificação de imagens dentro do EUCALYPTUS. Com a informação obtida, a classe espelho gera uma regra para derrubar a VM, a partir do ID. Depois da VM derrubada, a classe espelho volta e captura o EMI. Isso porque, quando uma VM é cadastrada dentro do EUCALYPTUS, significa que aquela VM tem programas, dados e contas

de usuários que precisam ser mantidos. Então, dentro de um EMI, é como se fosse um backup emergencial daquela VM.

O EMI é o ponto zero, isso significa que é o inicio da vida da VM. Então é usado o backup incremental do Espelho para levar do ponto zero ao ponto em que a VM foi derrubada. O EMI é importante porque dá a base para que todos os dados da VM possam ser colocados de volta no ato da replicação. Então o EMI é recuperado e a replicação é efetivada. A Figura 5.13 apresenta o console do EUCALYPTUS no momento em que a VM é derrubada e replicada.

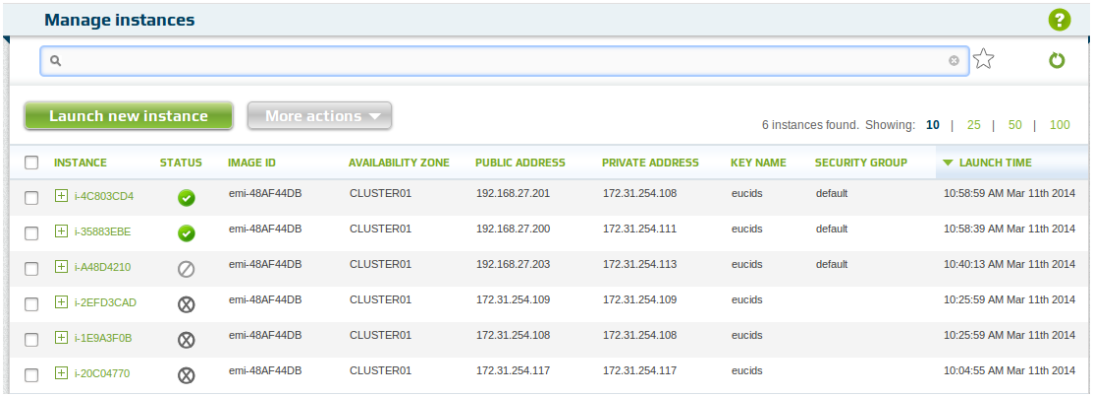

Figura 5.13: Substituição da VM.

Supondo que em uma VM contém instalado um apache, um gerenciador de identidade e um servidor FTP, caso seja necessária a derrubada dessa VM, ela terá que ser replicada. Assim, embora os 3 programas estejam supostamente instalados, se for pega uma VM sem replicar o EMI e se for colocada no lugar da VM derrubada, ela não irá conter os programas instalados. No caso, o administrador teria que ir lá e instalá-los. Mas, se a EMI daquela VM for recuperada, ela irá ser replicada e todos os programas da VM desativados serão colocados de volta na nova VM, ou seja, a réplica. Quando o backup incremental é colocado na VM, informações para que os programas possam voltar a executar são dadas. A Figura 5.14 ilustra o console do EUCALYPTUS depois da VM replicada com sucesso.

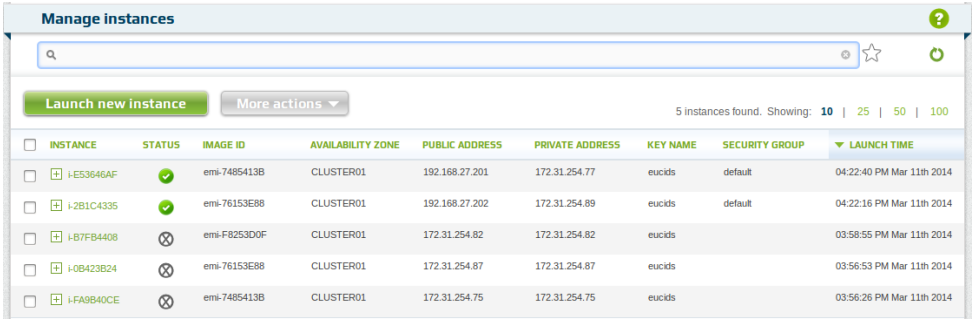

Figura 5.14: Replicação com sucesso.

No ambiente de teste citado, foram efetuados ataques do tipo: *SYN Flooding*, *Land*, *ICMP Flood*, *Smurf* e *Prob attack*. Como apresentado por [18] no sistema de detecção de intrusão proposto pelo autor, os ataques são prontamente identificados pelo SDI, que antes mostrava somente uma mensagem de alerta no console do IDS_Admin.

Portanto, verificamos que o mecanismo proposto para a detecção da intrusão no ambiente toma medidas de correção e de proteção de seus componentes. Diante de todos os ataques testados, o mecanismo se comportou da mesma forma: analisando e corrigindo falhas. Ou seja, não importa o ataque, o ambiente se recupera da mesma forma. Em todos os testes, as VMs que foram eliminadas tiveram suas criações efetivadas novamente.

5.4 Conclusão

Neste capítulo, foi apresentada a implementação do protótipo do mecanismo de tolerância a falha para o EICIDS em ambientes de Computação em Nuvem proposto nesta dissertação. Foram também realizados testes nos ambientes montados em laboratório, demonstrando a potencialidade do protótipo implementado em identificar ataques efetuados contra as máquinas virtuais da nuvem a partir de alguma VM comprometida e, com isso, replicando e protegendo o ambiente de futuras propagações de ataques. As características da virtualização foram utilizadas para proteção das VMs, bem como o uso mais eficiente dos recursos do ambiente de nuvem foi demonstrado pelo componente App_Monitor, que verifica as permissões de usuários e de suas aplicações, informando aos demais componentes do mecanismo a existência de possíveis ataques. No capítulo seguinte, apresentaremos a conclusão do trabalho, bem como as sugestões para trabalhos futuros.

6 Conclusões e Trabalhos Futuros

Este capítulo apresenta as contribuições deste trabalho, conclusões sobre os resultados alcançados e sugestões para trabalhos futuros.

6.1 Contribuição do trabalho

Os sistemas tolerantes a falhas se preocupam com todas as técnicas necessárias para permitir que um sistema tolere falhas de software ou hardware ao longo de seu funcionamento.

Os provedores de serviços na nuvem buscam inovações tecnológicas que garantam proteção e privacidade das informações e dos dados de usuários.

Atualmente, há inúmeros modelos de tolerância a falhas que fornecem diferentes mecanismos para melhorar o sistema. Todavia, ainda existem alguns desafios que precisam de alguma preocupação para cada tipo de problema. Por isso, são projetadas medidas de segurança que auxiliam na diminuição de preocupações das entidades utilizadoras da computação nas nuvens, a fim de prover maior confiabilidade e disponibilidade.

No presente trabalho, um mecanismo de tolerância a falhas para o EICIDS em ambientes de computação em nuvem foi proposto, visto que, além de proteger os usuários da nuvem de ameaças internas e também possibilitar uma abordagem de proteção baseada em superfícies de ataque, o mecanismo também efetua medidas de proteção e de correção de possíveis falhas.

A principal contribuição deste trabalho foi o desenvolvimento de um mecanismo de tolerância a falhas para o EICIDS (Sistema de Detecção de Intrusão Elástica e Baseada em Nuvem Interna). O mecanismo utiliza-se de algumas técnicas para prover a tolerância a falhas: o monitoramento, a emissão de eco e a replicação. O monitoramento feito pelos componentes do mecanismo tem como objetivo, junto com os componentes do EICIDS, detectar falhas. A emissão de eco servirá para deixar todos no ambiente cientes de que invasões estão ocorrendo e informar de onde vem o ataque, para que possam se prevenir e, assim, evitar a propagação do ataque dentro da nuvem.

Outra contribuição do trabalho é a utilização do componente App_Monitor,que verifica constantemente a existência de usuários e aplicações não autorizadas em execução no

ambiente, promovendo um uso mais eficiente dos recursos da nuvem ao bloquear um usuário mal-intencionado e matando todos os seus processos. Há ainda o componente Sinal, que envia sinais informando se os processos estão ativos ou não. Por meio desses componentes, o processo de replicação será feito de forma eficiente e correta.

Foi apresentada a implementação feita para os principais componentes do mecanismo, quando foram detalhados os principais métodos utilizados na aplicação.

Foi montado um ambiente de testes para validar o funcionamento e a capacidade de tolerar as falhas do mecanismo, e o seu objetivo principal foi alcançado: execução dos testes e visualização dos resultados.

Os resultados obtidos, através dos testes aplicados no laboratório criado para representar a infraestrutura de um ambiente de nuvem, confirmaram a capacidade de o mecanismo proposto proteger e recuperar o ambiente ao qual foi submetido. Nos testes aplicados, todos os ataques realizados a partir de uma máquina virtual maliciosa foram detectados pelo EICIDS e prontamente solucionados pelo mecanismo proposto.

Apesar de que somente ataques do tipo DoS foram utilizados, o modelo proposto possibilita que outros tipos de ataque possam ser identificados e corrigidos. Como visto no decorre do trabalho, os tipos de ataque não influem no processo de atividade do mecanismo proposto. E este estará pronto para atuar de acordo com as características da nuvem.

6.2 Limitação

Nesta proposta foi identificado a seguinte limitação:

- O mecanismo foi projetado para funcionar de modo distribuído e centralizado. Contudo, a estrutura centralizada é vulnerável, por apresentar um único ponto de falhas. Ou seja, o componente IDS_Adim pode se tornar uma vulnerabilidade, junto com os componentes do mecanismo proposto. Caso um atacante tenha conhecimento sobre ele toda estrutura pode ser parada;

- Todos os problemas que ocorrem a nível de VM, são direcionados ao *HYPERVISOR*. Onde, localizam-se parte dos componentes do mecanismo proposto e o elemento central do EICIDS. Isso é uma grande limitação já que as VM não têm autonomia para gerenciar falhas.

6.3 Trabalhos Futuros

Com o desenvolver deste trabalho, foram identificados algumas possibilidades para trabalhos futuros, dentre as quais podemos sugerir:

- Utilizar Inteligência Artificial na detecção de intrusão e adaptar os componentes do mecanismo para Sistemas Multi-Agentes.

- Fazer uma avaliação de desempenho do mecanismo. Testando sua eficiência e disponibilidade.

- Utilizar a computação autonômica e todas suas características para tornar o mecanismo mais eficiente.

- A extensão do mecanismo com o uso de agentes inteligente é um ótimo trabalho a ser desenvolvido dentro dessa área de pesquisa, inserindo novos sistemas de segurança para cooperação e, com isso, obter um aumento no nível de segurança da rede;

- Adaptar o Mecanismo proposto para dispositivos móveis;

- Tornar as máquinas virtuais autônomas. Ou seja, permitindo que elas sejam capazes de se auto corrigir diante de uma falha.

Referências Bibliográficas

[1] G. Aceto, A. Botta, W. De Donato, and A. Pescapè. Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115, 2013.

[2] T. Alharkan and P. Martin. Idsaas: Intrusion detection system as a service in public clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 686–687, 2012.

[3] C. Babcock. *Cloud Implementation To Double By 2012*, Abril 2012. Disponível em: http://www.informationweek.com/cloud/software-as-a-service/cloud-implementation-to-double-by-2012/d/d-id/1076825?. Acessado em:10 de Dezembro de 2013.

[4] R. Bace and P. Mell. Nist special publication on intrusion detection systems. Technical report, DTIC Document, 2001.

[5] A. Bisong, M. Rahman, et al. An overview of the security concerns in enterprise cloud computing. *arXiv preprint arXiv:1101.5613*, 2011.

[6] J. Brodkin. Gartner: Seven cloud-computing security risks, 2008.

[7] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 5–13. Ieee, 2008.

[8] W. Cambiucci. *Computação em Nuvem: algumas perguntas sobre desafios em projetos*, 2013. Disponível em: http://blogs.msdn.com/b/wcamb/archive/2010/05/07/computa-o-em-nuvem-algumas-perguntas-sobre-desafios-em-projetos.aspx. Acessado em: 20-10-2013.

[9] R. S. Campello, R. F. Weber, V. d. S. SERAFIM, and V. G. Ribeiro. O sistema de detecçao de intrusao asgaard. In *Workshop de Segurança de Sistemas Computacionais*, 2001.

[10] A. Carissimi. Virtualização: da teoria a soluções. *Minicursos do Simpósio Brasileiro de Redes de Computadores–SBRC*, 2008:173–207, 2008.

[11] Cisco. *Tecnologias eminentes no data center*. Cisco - Market Pulse. Disponível em http://www.2mul.com/web/BR/assets/executives/pdf/Cisco_US_4pgMktPulse_1011.pdf, Acesso em 20-01-2014.

[12] C. S. A. (CSA). *Cloud Computing Vulnerability Incidents: A Statistical Overview.*, August 2012. Disponível em : http://www.cert.uy/wps/wcm/connect/975494804fdf89eaabbdab1805790cc9/Cloud_Computing_V Acessado em: 01 de Dezembro de 2013.

[13] P. Das and P. Khilar. Vft: A virtualization and fault tolerance approach for cloud computing. In *Information Communication Technologies (ICT), 2013 IEEE Conference on*, pages 473–478, 2013.

[14] C. de Segurança para Internet. *Cert.br - Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil*. Cert.br, 2012. Disponivel em http://cartilha.cert.br/ataques/. Acessado em 20-12-2013.

[15] E. P. de Souza and J. A. S. Monteiro. *Estudo sobre Sistema de Detecção de Intrusão por Anomalias Uma Abordagem Utilizando Redes Neurais*. PhD thesis, Master's thesis, Salvador University, Salvador, Brazil, 2008.

[16] S. N. Dhage and B. Meshram. Intrusion detection system in cloud computing environment. *International Journal of Cloud Computing*, 1(2):261–282, 2012.

[17] C. Dias. *SEGURANÇA E AUDITORIA DA TECNOLOGIA DA INFORMAÇÃO*. 2000.

[18] J. Dias. *Um modelo de detecção de intrusão para o ambiente de computação em nuvem. In Dissertação de mestrado. Coordenação de Pós-Graduação em Engenharia de Eletricidade.* UFMA - Universidade Federal do Maranhão, 2013.

[19] P. M. e Timothy Grance. *The NIST Definition of Cloud Computing*. NIST - National Institute of Standards and Technology U.S. Department of Commerce, 2011. Disponível em: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. Acessado em: 12-10-2013.

[20] S. V. B. EVANGELISTA. Sistemas de detecção de intrusos e sistemas de prevenção de intrusos: Princípios e aplicação de entropia. *Petrópolis: IST*, page 19, 2008.

[21] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.

[22] G. T. Guedes. *UML: uma abordagem prática*. Novatec Editora, 2008.

[23] Y. Jadeja and K. Modi. Cloud computing - concepts, architecture and challenges. In *Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on*, pages 877–880, 2012.

[24] D. Jamil and H. Zaki. Cloud computing security. *International Journal of Engineering Science and Technology*, 3(4):3478–3483, 2011.

[25] S. Jayadivya, S. Jaya Nirmala, and S. Mary Saira Bhanu. Fault tolerant workflow scheduling based on replication and resubmission of tasks in cloud computing. *IJCSE) ISSN*, pages 0975–3397, 2012.

[26] P. Kaur, D. Rattan, and A. K. Bhardwaj. An analysis of mechanisms for making ids fault tolerant. *International Journal of Computer Applications*, 1(24):22–25, 2010.

[27] H. Kholidy and F. Baiardi. Cids: A framework for intrusion detection in cloud systems. In *Information Technology: New Generations (ITNG), 2012 Ninth International Conference on*, pages 379–385, 2012.

[28] H. Kholidy, A. Erradi, S. Abdelwahed, and F. Baiardi. Ha-cids: A hierarchical and autonomous ids for cloud systems. In *Computational Intelligence, Communication Systems and Networks (CICSyN), 2013 Fifth International Conference on*, pages 179–184, 2013.

[29] E. Klein. *Modelos de Serviço de Computação em Nuvem*. Mobiltec - mobilidade em negócios, 2013. Disponível em: http://www.mobiltec.com.br/blog/index.php/modelos-de-servico-de-computacao-em-nuvem/. Acessado em: 20-10-2013.

[30] KVM. *Kernel Basead Virtual Machine - KVM.* Red Hat. Write Paper., 2009. Disponível em: http:/www.redhat.com. Acessado em: 13/09/2013.

[31] KVM. *KVM - Kernel Based Virtual Machine*, 2013. Disponivel em: <http://www.linux-kvm.org/page/Main_Page>. Acessado em: 03/10/2013.

[32] Y. Li, W. Li, and C. Jiang. A survey of virtual machine system: Current technology and future trends. In *Electronic Commerce and Security (ISECS), 2010 Third International Symposium on*, pages 332–336. IEEE, 2010.

[33] C. F. L. Lima. Agentes inteligentes para detecçao de intrusos em redes de computadores. *São Luís*, 2002.

[34] S. Liu, Y. Liang, and M. Brooks. Eucalyptus: a web service-enabled e-infrastructure. 2007.

[35] S. Malik and F. Huet. Adaptive fault tolerance in real time cloud computing. In *Services (SERVICES), 2011 IEEE World Congress on*, pages 280–287. IEEE, 2011.

[36] D. M. F. Mattos. Virtualização: Vmware e xen. *Universidade Federal do Rio de Janeiro*, 2008.

[37] P. Mell and T. Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800(145):7, 2011.

[38] D. A. Menascé. Virtualization: Concepts, applications, and performance modeling. In *Int. CMG Conference*, pages 407–414, 2005.

[39] V. o Linux. *Transferindo arquivos com o rsync*, 2014. Disponivel em: <http://www.vivaolinux.com.br/artigo/Transferindo-arquivos-com-o-rsync>. Acessado em: 03/09/2013.

[40] P. H. Pedrosa and T. Nogueira. Computação em nuvem. *artigo disponível em http://www. ic. unicamp. br/˜ ducatte/mo401/1s2011 T*, 2.

[41] H. PEREIRA. sistema de detecção de intrusão. 2011.

[42] D. Pilone and N. Pitman. *UML 2.0 in a Nutshell*. O'Reilly Media, Inc., 2009.

[43] G. Ramachandran and D. Hart. A p2p intrusion detection system based on mobile agents. In *Proceedings of the 42nd annual Southeast regional conference*, pages 185–190. ACM, 2004.

[44] P. Renato. *Backtrack: Solução Open Source para pen test.*, 2012. Disponível em: . Acessado em: 25/11/2013.

[45] W. G. I. Robert, M. B. Kadonoff, J. F. Maddox, and R. A. Wendt. Intrusion detection system, Sept. 20 1988. US Patent 4,772,875.

[46] S. Roschke, F. Cheng, and C. Meinel. Intrusion detection in the cloud. In *Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on*, pages 729–734. IEEE, 2009.

[47] R. Sandhu and P. Samarati. Authentication, access control, and intrusion detection. *The Computer Science and Engineering Handbook*, 1:929–1, 1997.

[48] Sanfilippo. *Hping*. Disponível em: http://hping.org. Acessado em: 16/12/2013.

[49] J. Schmuller. *Sams teach yourself UML in 24 hours*. Sams publishing, 2004.

[50] F. B. Shaikh and S. Haider. Security threats in cloud computing. In *Internet technology and secured transactions (ICITST), 2011 international conference for*, pages 214–219. IEEE, 2011.

[51] M. P. C. Silva and M. N. S. Sampaio. Estudo de sistemas de detecção e prevenção de intrusões uma abordagem open source, 2006.

[52] L. Siqueira and Z. Abdelouahab. A fault tolerance mechanism for network intrusion detection system based on intelligent agents (nidia). In *Software Technologies for Future Embedded and Ubiquitous Systems, 2006 and the 2006 Second International Workshop on Collaborative Computing, Integration, and Assurance. SEUS 2006/WCCIA 2006. The Fourth IEEE Workshop on*, pages 6–pp. IEEE, 2006.

[53] D. Svantesson and R. Clarke. Privacy and consumer risks in cloud computing. *Computer Law & Security Review*, 26(4):391–397, 2010.

[54] X. Tan and B. Ai. The issues of cloud computing security in high-speed railway. In *Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference on*, volume 8, pages 4358–4363. IEEE, 2011.

[55] C. Taurion. *Cloud Computing - Computacao em Nuvem - Transformando o mundo da tecnologia*. 2009.

[56] C. Taurion. *A adoção de cloud computing é apenas uma questão de velocidade e intensidade.*, 2013. Disponivel em: <http://imasters.com.br/infra/cloud/a-adocao-de-cloud-computing-e-apenas-uma-questoa-de-velocidade-e-intensidade/>. Acessado em: 03 março de 2013.

[57] A. Tchana, L. Broto, and D. Hagimont. Fault tolerant approaches in cloud computing infrastructures. In *ICAS 2012, The Eighth International Conference on Autonomic and Autonomous Systems*, pages 42–48, 2012.

[58] R. Vanathi and S. Gunasekaran. Comparison of network intrusion detection systems in cloud computing environment. In *Computer Communication and Informatics (ICCCI), 2012 International Conference on*, pages 1–6. IEEE, 2012.

[59] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, Dec. 2008.

[60] K. Vieira, A. Schulter, C. B. Westphall, and C. M. Westphall. Intrusion detection for grid and cloud computing. *It Professional*, 12(4):38–43, 2010.

[61] T. Wadlow. *Segurança de Redes*. 2000.

[62] T. S. Weber. Tolerância a falhas: conceitos e exemplos. *Intech Brasil, Distrito*, 4:32–42, 2003.

[63] M. Y. A. Younis and K. Kifayat. Secure cloud computing for critical infrastructure: A survey. *Liverpool John Moores University, United Kingdom, Tech. Rep*, 2013.

[64] W. Zhao, P. Melliar-Smith, and L. E. Moser. Fault tolerance middleware for cloud computing. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 67–74. IEEE, 2010.

Parte I

Apêndice

A . Apêndice - Códigos implementados

Com o objetivo de demonstrar o funcionamento do mecanismo proposto, alguns dos principais trechos de código dos componentes que compõem o mecanismo de tolerância a falhas foram selecionados.

A.1 Componentes IP_Analyser e APP_Monitor

A seguir são apresentados através da figura, trechos do código implementado para o IP_*Analyser*.

```
 9  public class Analyser_ip {
10      String attack;
11      public Analyser_ip(String attack){
12          this.attack = attack;
13      }
14      //captura dados do arquivo
15      private String getAllFileResource(String filename){
16          BufferedReader br;
17          FileReader fr;
18          String resource = "";
19          try {
20              fr = new FileReader(filename);
21              br = new BufferedReader(fr);
22              while(br.ready()){
23                  resource += br.readLine();
24              }
25              br.close();
26              fr.close();
27              return resource;
28          } catch (FileNotFoundException e) {
29              System.out.println("ERRO AO abrir ARQUIVO, ANALYSER IP");
30          } catch (IOException e) {
31              System.out.println("erro ler o arquivo, analyser_ip");
32          }
33          return resource;
34      }
```

Figura A.1: Trecho de código do IP_Analyser. Parte 1

```
35      //retorna o ip do ataque
36      private String getIp(){
37          StringTokenizer stk;
38          String ip_do_ataque="";
39          stk = new StringTokenizer(attack,":",false);
40          while(stk.hasMoreElements()){
41              ip_do_ataque = (String) stk.nextElement();
42          }
43          return ip_do_ataque;
44      }
45      //verifica se o ip atacante pertence ao broadcast
46      private boolean isAIpOfBroadcast(String ip_do_ataque){
47          StringTokenizer stk,stk_ips;
48          String ips_ativos = getAllFileResource("/home/sdi/broadcast.txt");
49          String ips="";
50          stk_ips = new StringTokenizer(ips_ativos,"\n",false);
51          while(stk_ips.hasMoreElements()){
52              ips = (String)stk_ips.nextElement();
53              if(ips.contains(ip_do_ataque)){
54                  return true;
55              }
56          }
57          return false;
58      }
```

Figura A.2: Trecho de código do IP_Analyser. Parte 2

```
59      //gera regra de bloqueio do acesso malicioso
60      public String genRule(String ip_do_atacante){
61          String rulemodel1="iptables -A INPUT -s ";
62          String rulemodel2=" -j DROP";
63          if(ip_do_atacante.equals("")){
64              return "não há atacante";
65          }
66          return rulemodel1+ip_do_atacante+rulemodel2;
67      }
68      //emite echo caso uma acao maliciosa seja detectada,alertando demais vm no espaço
69      private void echo(String rule) throws UnknownHostException{
70          Socket echo_socket;
71          String ip,ip_of_ataque;
72          String ips = getAllFileResource("/home/sdi/arq_ips.txt");
73          StringTokenizer stk = new StringTokenizer(ips,"\n",false);
74          ip_of_ataque = getIp();
75          OutputStream os;
76          PrintStream ps;
77          while(stk.hasMoreElements()){
78              ip = (String)stk.nextElement();
79              if(!ip.equals(ip_of_ataque)&&!ip.equals(minigMyIP())){
80                  try {
81                      echo_socket = new Socket(ip,222);
82                      os = echo_socket.getOutputStream();
83                      ps = new PrintStream(os);
                            .
                            .
                            .
```

Figura A.3: Trecho de código do IP_Analyser. Parte 3

```
12  public class Analyser {
13      // buffer do arquivo de ips broadcast
14      private BufferedReader buf_broadcast;
15      // buffer do arquivo de ips ativos na rede
16      private BufferedReader buf_ips_ativos;
17      //abre arquivos
18      public void open_files() {
19          try {
20              // abre o arquivo de ips de broadcast para leitura
21              buf_broadcast = new BufferedReader(new FileReader(
22                      "/home/sdi/broadcast.txt"));
23
24              // abre o arquivo de ips ativos da rede para leitura
25              buf_ips_ativos = new BufferedReader(new FileReader(
26                      "/home/sdi/arq_ips.txt"));
27          } catch (IOException ie) {
28              System.out.println("Falha na abertura de arquivo.");
29              ie.printStackTrace();
30          }
31      }
```

Figura A.4: Trecho do código APP_Monitor. Parte 1.

```
33      /**
34       * Analisa pacotes do tipo TCP.
35       */
36      @SuppressWarnings("unused")
37      public String analise_pacote_tcp(String[] pack) {
38          String src_ip, src_port, dest_ip, dest_port;
39          String tcp_syn, tcp_ack, tcp_fin, tcp_data;
40          String ip_da_rede = null; // ip atual da rede interna
41          boolean eh_ip_rede = false; // if src_ip eh ip ativo da rede intern
42
43          open_files();
44
45          String data = "dd/MM/yyyy";
46          String hora = "HH:mm:ss";
47          String data1, hora1, date;
48          java.util.Date agora = new java.util.Date();
49          SimpleDateFormat formata = new SimpleDateFormat(data);
50          data1 = formata.format(agora);
51          formata = new SimpleDateFormat(hora);
52          hora1 = formata.format(agora);
53          date = String.format(data1 + " " + hora1);
54
55          try {
56              if (pack.length > 9) {
57                  System.out.println("Tamanho do TCP packet invalido.");
58                  return null;
59              }
```

Figura A.5: Trecho do código APP_Monitor. Parte 2.

```
60              src_ip = pack[2];//quem ataca
61              src_port = pack[3];
62              dest_ip = pack[4];//quem é atacado
63              dest_port = pack[5];
64              tcp_syn = pack[6];
65              tcp_ack = pack[7];
66              tcp_fin = pack[8];
67              // tcp_data = pack[8];
68
69              // verifica se o ip de origem eh ip da rede ativo da rede interna
70              if (buf_ips_ativos.ready()) {
71                  while ((ip_da_rede = buf_ips_ativos.readLine()) != null) {
72                      if (ip_da_rede != null) {
73                          if (src_ip.equals(ip_da_rede)) {
74                              eh_ip_rede = true;
75                              break;
76                          }
77                      }
78                  }
79              }
```

Figura A.6: Trecho do código APP_Monitor. Parte 3.

Parte II

Apêndice

A.2 Componentes Sinal

A seguir são apresentados através das figuras, trechos do código implementado para o *Send*, *Receive* e Chamada.

```
5  public class Send extends Thread {
6      public void node(int count){
7          if(count < 5){
8              try {
9                  String line="";
10                 ServerSocket s = new ServerSocket(1020);
11                 Socket aux = s.accept();
12                 Scanner scanner = new Scanner(aux.getInputStream());
13                 line = scanner.nextLine();
14                 System.out.println(line);
15                 burn(new File("/home/sdi/logSend.txt"),line);
16                 s.close();
17                 aux.close();
18                 while(send() == false){
19                     send();
20                 }
21             } catch (IOException e) {
22                 count ++;
23                 node(count);
24             }
25         }else{
26             send();
27         }
28     }
```

Figura A.7: Trecho do código do Send. Parte 1

```
29      private void burn(File arq,String dados){
30          if(arq!=null){
31              try {
32                  arq.createNewFile();
33                  FileWriter fw = new FileWriter(arq);
34                  fw.write(dados);
35                  fw.flush();
36                  fw.close();
37              } catch (IOException e) {
38                  System.out.println("erro ao escreever no arquivo");
39              }
40          }
41      }
42      public boolean send(){
43          Socket socket;
44          try {
45              socket = new Socket("192.168.27.79",1021);
46              PrintStream p = new PrintStream(socket.getOutputStream());
47              p.println("gEPmA3USJdI");
48              p.close();
49              socket.close();
50          } catch (IOException e) {
51              return false;//app
52          }
53          return true;
54          }
```

Figura A.8: Trecho do código do Send. Parte 2

```
20      private class Receive extends Thread{
21          ServerSocket s;
22          Socket sock;
23          public Receive(ServerSocket socket){
24              s = socket;
25          }
26          public void run() {
27              try {
28                  sock = s.accept();
29                  Scanner s = new Scanner(sock.getInputStream());
30                  if(s.nextLine().equals("gEPmA3USJdI")){
31                      System.out.println("ok <- "+sock.getInetAddress().getHostAddress());
32                  }else{
33                      System.out.println("replicando-chave incorreta");
34                      Hypervisor hv = new Hypervisor();
35                      hv.run(sock.getInetAddress().getHostAddress());
36                      hv.waiting();
37                      hv.terminate(sock.getInetAddress().getHostAddress());
38                      gendir();
39                  }
40                  s.close();
41                  sock.close();
42                  this.s.close();
43              } catch (IOException e) {
44                  run();
45              }
```

Figura A.9: Trecho do código do Receive.

```
133     public void doReceive(String ip,int count){
134         Receive r;
135         if(count < 5){
136             try {
137                 Socket sender = new Socket(ip,1020);
138                 ServerSocket ss = new ServerSocket(1021);
139                 PrintStream p = new PrintStream(sender.getOutputStream());
140                 p.println("requisitando...");
141                 p.close();
142                 sender.close();
143                 r = new Receive(ss);
144                 r.start();
145                 waiting(20);
146                 if(r.isAlive()){
147                     r.interrupt();
148                     r.yield();
149                     count ++;
150                     doReceive(ip,count);
151                 }
152             } catch (IOException e) {
153                 System.out.println("trying again");
154                 count++;
155                 doReceive(ip,count);
156             }
```

Figura A.10: Trecho do código Chamada.

Parte III

Apêndice

A.3 Componentes de monitoramento e proteção do EICIDS

A seguir são apresentados através das figuras, trechos do código implementado para o *Sart*, Monitor, *Analyser*, Ação e Sensor.

```
5  public class StartSensor {
6      static void waiting(int time){//confere a quantidade de
7      //segundos á esperar por uma conexão
8          String data,next="";
9          Date agora = new Date();
10         SimpleDateFormat formata = new SimpleDateFormat("ss")
11         data = formata.format(agora);
12         Date agora1;
13         SimpleDateFormat formata1;
14         while(time!=0){
15             agora1 = new Date();
16             formata1 = new SimpleDateFormat("ss");
17             next = formata1.format(agora1);
18             if(!data.equals(next)){
19                 time--;
20                 data = next;
21             }
22         }
23     }
24     public static void main(String[] args) {
25         int flag = 0,flag_admin = 0;
26         Sensor sensor;
27         Process proc,proc1,proc_nast;
28         File arq = new File("/home/sdi/listener.txt");
29         File arq1 = new File("/home/sdi/listenerAdmin.txt");
30         FileReader fr,fr1;
```

Figura A.11: Trecho do código Start.

```
3  public class Monitor extends Thread{
4      private String linha;
5      private String protocolo;
6      private String ip;
7      private String porta;
8      private String target;
9      private int posix;
10     public Monitor(String linha){
11         this.linha = linha;
12         posix = 0;
13         protocolo = "";
14         ip = "";
15         porta = "";
16         target = "";
17         getProtocol();
18         getIp();
19         getPort();
20         getTarget();
21     }
22     private void getProtocol(){
23         String temp = "";
24         int i = 0;
25         if(linha.equals("")){
26             System.out.println("linha nula");
27             return;
28         }
               .
               .
               .
```

Figura A.12: Trecho do código do Monitor.

```
5  public class Analyser {
6      private String[] dados;
7      private String whitelist;
8      private Action ac;
9      public Analyser(String[] linha){
10         dados = linha;
11         whitelist = getWhiteList();
12     }
13     private String getWhiteList(){
14         String temp="";
15         File arq = new File("/home/sdi/whitelist.txt");
16         FileReader fr;
17         BufferedReader br;
18         try {
19             fr = new FileReader(arq);
20             br = new BufferedReader(fr);
21             while(br.ready()){
22                 temp += br.readLine();
23                 temp += "|";
24             }
25             br.close();
26         } catch (IOException e) {
27             System.out.println("erro ao ler do arquivo");
28         }
29         return temp;
30     }
```

Figura A.13: Trecho do código Analyser.

```
3  public class Action implements Runnable{
4      String temp="";
5      public Action(String rule){
6          temp = rule;
7      }
8      Process proc;
9      public void run() {
10         try {
11             proc = Runtime.getRuntime().exec(temp);
12             proc.waitFor();
13         } catch (InterruptedException e) {
14             System.out.println("Erro");
15         } catch (IOException e) {
16             System.out.println("Erro");
17         }
18     }
19
20 }
21
```

Figura A.14: Trecho do código Ação.

```
4  public class Sensor extends Thread{
5      Process proc;
6      @Override
7      public void run() {
8          File arq = new File("/home/sdi/log.txt");
9          BufferedReader br;
10         FileReader fr;
11         Monitor monitor;
12         String temp="",linha = "";
13         Analyser ana;
14         long tam=0;
15         try {
16             proc = Runtime.getRuntime().exec("bash /home/sdi/nast.sh");
17             if(!arq.exists()){
18             System.out.println("erro ao iniciar o nast..verifique se ele está instalado ");
19                 return;
20             }
```

Figura A.15: Trecho do código Sensor.

Parte IV

Apêndice

A.4 Componentes de replicação

A seguir são apresentados através das figuras, trechos do código implementado para o *Instance*, Minerador e Espelho.

```
7  public final class Instancia {
8      @SuppressWarnings("unused")
9      private String id;
10     @SuppressWarnings("unused")
11     private String ip;
12     @SuppressWarnings("unused")
13     private Process proc;
14     private ArrayList<String>ips;
15     public Instancia() {
16         try {
17             proc = Runtime.getRuntime().exec("bash /home/sdi/AtuadorInstancias.sh");
18             ips = new ArrayList<String>();
19             proc.waitFor();
20         } catch (IOException e) {
21                 System.out.println("Erro no efetor de instancias");
22         } catch (InterruptedException e) {
23             e.printStackTrace();
24         }
25     }
                    .
                    .
                    .
```

Figura A.16: Trecho do código Instance.

```
8  public class  Minerador  {
9      Instancia ins;
10     public  Minerador  (){
11         ins = new Instancia();
12     }
13     @SuppressWarnings("resource")
14     public ArrayList<String> getIP(String filename){
15         ArrayList<String> ips = new ArrayList<String>();
16         File arq = new File(filename);
17         FileReader fr;
18         String temp="",ip="";
19         StringTokenizer stk;
20         try {
21             fr = new FileReader(arq);
22             BufferedReader br = new BufferedReader(fr);
23             while(br.ready()){
24                 temp = br.readLine();
25                 stk = new StringTokenizer(temp,":",false);
26                 while(stk.hasMoreElements()){
27                     ip = (String)stk.nextElement();
28                     //System.out.println((String)stk.nextElement());
29                 }
30                 if(!ips.contains(ip)){
31                     ips.add(ip);
32                 }
33             }
```

Figura A.17: Trecho do código Minerador.

```
44     @SuppressWarnings("resource")
45     public ArrayList<String> getIPInstance(String filename){
46         Instancia i = new Instancia();
47         i.genConfig();
48         ArrayList<String> ips = new ArrayList<String>();
49         FileReader fr;
50         BufferedReader br;
51         String temp="",ip="";
52         try {
53             fr = new FileReader(filename);
54             br = new BufferedReader(fr);
55             while(br.ready()){
56                 temp = br.readLine();
57                 if(temp.contains("->")){
58                     temp = br.readLine();
59                     while(!temp.contains("<-")){
60                         ip+=temp+":";
61                         temp=br.readLine();
62                     }
63                     ips.add(ip);
64                 }
65                 ip="";
66             }
                   .
                   :
```

Figura A.18: Trecho do código Espelho.