PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MITIGAÇÃO DE ATAQUES DE NEGAÇÃO DE SERVIÇO EM RESTS AUTENTICÁVEIS NA NUVEM

RÉGIO ANTONIO MICHELIN

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Avelino F. Zorzo

Porto Alegre
2015

Dados Internacionais de Catalogação na Publicação (CIP)

M623m Michelin, Régio Antonio
Mitigação de ataques de negação de serviço em rests autenticáveis na nuvem / Régio Antonio Michelin. – Porto Alegre, 2015.
89 f.

Dissertação (Mestrado) – Faculdade de Informática, PUCRS.
Orientador: Prof. Dr. Avelino F. Zorzo

1. Informática. 2. Computação em Nuvem.
3. Arquitetura de Computador . 4. Segurança de Dados.
I. Zorzo, Avelino F. II. Título.

CDD 004.36

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

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

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Mitigação de Ataques de Negação de Serviço em RESTs Autenticáveis na Nuvem" apresentada por Régio Antonio Michelin como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, aprovada em 22/01/2015 pela Comissão Examinadora:

Prof. Dr. Avelino Francisco Zorzo – Orientador — PPGCC/PUCRS

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

Prof. Dr. Antônio Marinho Pilla Barcellos – UFRGS

Homologada em 04/02/2015, conforme Ata No. 002 pela Comissão Coordenadora.

Prof. Dr. Luiz Gustavo Leão Fernandes
Coordenador.

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

“It is known that there are an infinite number of worlds, simply because there is an infinite amount of space for them to be in. However, not every one of them is inhabited. Therefore, there must be a finite... Any finite number divided by infinity is as near to nothing as makes no odds, so the average population of all the planets in the Universe can be said to be zero. From this it follows that the population of the whole Universe is also zero, and that any people you may meet from time to time are merely products of a deranged imagination.”
(Douglas Adams)

AGRADECIMENTOS

Agradeço à minha esposa Mirian pelo incentivo, paciência e compreensão, e ao meu filho Murilo que veio para adicionar emoção ao último semestre do mestrado.

Aos meus amigos pelas ideias trocadas, ajuda e motivação.

Ao meu orientador Avelino pelo incentivo, suporte, conselhos e ensinamentos.

A HP pela disponibilização da bolsa de estudos através da Lei de Informática.

A todos outros que esqueci e que fizeram parte e me apoiaram ao longo deste período.

E por fim agradeço a mim mesmo por ter conseguido chegar até aqui.

MITIGAÇÃO DE ATAQUES DE NEGAÇÃO DE SERVIÇO EM RESTS AUTENTICÁVEIS NA NUVEM

RESUMO

Nos dias de hoje sistemas disponíveis na Internet estão expostos aos mais diversos tipos de ataques. Estes ataques têm diferentes finalidades tais como roubar dados dos sistemas e implantação de códigos maliciosos, podendo até mesmo deixar um sistema totalmente indisponível, o que em sistemas de alta disponibilidade seria um grande problema.

Ataques de negação de serviço merecem uma atenção especial, pois atualmente vemos muitos ataques sendo executados por diferentes razões, como: política, concorrência industrial, protestos ou até mesmo apenas por diversão. Esse tipo de ataque consiste em esgotar os recursos computacionais do seu alvo, deixando-o lento ou até mesmo indisponível.

Quando o alvo de um ataque de negação de serviço é um ambiente de nuvem, não apenas um sistema pode ser comprometido, mas sim todos os sistemas que estão disponíveis no ambiente em questão. Desse modo, um ambiente de nuvem é um bom alvo para que usuários mal intencionados, através de um ataque ao mesmo, comprometam um grande número de sistemas nele rodando.

Ambientes de nuvem costumam expor, através do modelo arquitetural de REST, uma API para que usuários do seu serviço possam, programaticamente, integrar seus sistemas. Sabendo disso, essa REST API exposta acaba por se tornar um potencial vetor de ataques aos ambientes de nuvem. Com isso, faz-se necessário a utilização de um mecanismo de autenticação, com intuito de permitir acesso apenas aos usuários legítimos. Assim, o presente trabalho foca em analisar o problema de ataque de negação de serviço que explora o mecanismo de autenticação de REST API de ambientes de nuvem. A este problema, será proposta uma solução que atua em nível de aplicação, traçando um perfil

dos clientes que fazem uso desta REST, e desse modo evitando que o ambiente de nuvem seja sobrecarregado com operações desnecessárias.

Palavras-Chave: Ataque de negação de serviço, REST, Ambiente de Nuvem, Segurança de Nuvem.

MITIGATING DENIAL OF SERVICE TO AUTHENTICATED CLOUD REST

ABSTRACT

Computer systems available on the Internet are used for, basically, everyone. This widespread use has facilitated their exposure to several different types of attacks. These attacks are intended to, for example, steal information, deploy malicious code and even to make a system slow to respond, or worst, to become completely offline.

Denial of service attacks is a type of attack that, currently, needs a special attention, since they may be performed for different reasons, such as political purposes, industrial competition, protests or even just for fun. This kind of attack has as its main purpose to slow response time or even to make a computer system unavailable, for example, consuming all target computational resources.

When a denial of service attack targets a cloud environment, it could compromise not only one system, but also all systems that are hosted in the cloud. Therefore, a cloud environment is a convenient target for malicious users, since that with a single attack they are able to hit multiple systems.

A cloud environment usually exposes, through a REST architecture model, an API to allow its users to write their own service to be integrated with the cloud environment. Hence, this exposed REST API becomes a potential threat to the cloud environment. In order to reduce the risk of attacks, usually a cloud environment uses an authentication mechanism to allow only legitimate users to access the system. Despite that, some attacks may still be possible.

This research is focused on the analysis of denial of service attacks that exploit the authentication mechanism through REST API calls in a cloud environment. In this work, we propose a solution that works in the application level. Our solution creates client profiles in order to verify whether a client is posing any threat to the cloud or not. When a threat is detected, then our solution starts to filter calls from users that were identified as malicious.

We applied our solution to Openstack, an open cloud management system, and showed that we have improved response time for legitimate users when the cloud is under attack.

Keywords: Denial of Service Attack, REST, Cloud, Cloud Security.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API – Application Program Interface

CAPTCHA – Completely Automated Public Turing test to tell Computers and Humans Apart

CMS – Cloud Management Systems

CR – Carriage Return

CSA – Cloud Security Alliance

CVE – Common Vulnerabilities and Exposures

DDOS – Distributed Denial of Service

DOS – Denial of Service

EC2 – Amazon Elastic Compute Cloud

HTTP – Hypertext Transfer Protocol

IAAS – Infrastructure as a Service

KVM – Kernel-based Virtual Machine

LF – Line Feed

MADRA – Mitigação de Ataques DoS em RESTs Autenticáveis

NTP – Network Time Protocol

PSN – Playstation Network

REST – Representational State Transfer

RUDY – Are You Dead Yet

SOAP – Simple Object Access Protocol

TCP – Transmission Control Protocol

TI – Tecnologia da Informação

SUMÁRIO

1. INTRODUÇÃO

Nos dias de hoje sistemas disponíveis na Internet estão expostos aos mais diversos tipos de ataques, como por exemplo SQL *Injection*, *Cross-site scripting*, etc. [HHL03]. Esses ataques têm diferentes finalidades, tais como: roubar dados dos sistemas, implantação de códigos maliciosos ou até mesmo deixar um sistema totalmente indisponível, o que em sistemas de alta disponibilidade seria um grande problema.

Dentre os diferentes tipos de ataques, este trabalho dará especial atenção para os que têm como finalidade tornar um sistema indisponível, pois além de deixa-lo indisponível, um usuário mal intencionado também pode tentar se passar pelo alvo. Esses ataques são conhecidos como ataques de negação de serviço (DoS - *Denial of Service*) e o objetivo principal é deixar os usuários do sistema sem acesso, por exemplo, através do esgotamento dos recursos computacionais (processador, rede, memória, etc.) [Ste14]. Conseguindo que os clientes legítimos do sistema não tenham acesso ao mesmo pode dar um grande poder aos usuários mal intencionados. Através desse poder, eles podem fazer uso desse tipo de ataque para tirar algum sistema importante do ar em uma guerra cibernética, ou mesmo para praticar algum tipo de extorsão do seu alvo. Em um exemplo desse tipo de situação, recentemente o site agregador de notícias Feedly [Fee14] teve seus servidores sendo alvo de um poderoso ataque de negação de serviço, deixando o mesmo inacessível para os seus usuários, e para que o ataque cessasse, os atacantes exigiam um pagamento.

Outro fator que tem contribuído para o aumento do número de ataques é que está cada vez mais simples disponibilizar um sistema na Internet. Para que isso seja possível, atualmente os desenvolvedores contam com ambiente de nuvem, que consiste de uma arquitetura onde através de máquinas físicas e virtuais é criada uma infraestrutura para seu uso. As empresas fornecedoras de nuvem conseguem disponibilizar um ou mais servidores, onde o desenvolvedor paga apenas pelos recursos que usar. Desse modo, o desenvolvedor se preocupa apenas com a criação do seu serviço, ao passo que a gerência da infraestrutura fica delegada à empresa fornecedora da nuvem (este modelo de negócio é chamado de Infraestrutura como Serviço - *Infrastructure as a Service* - IaaS) [DTM10].

Um dos aspectos que favoreceram a popularização e barateamento de computação em nuvem foi a utilização de virtualização [PSL13]. Graças à virtualização é possível que o hardware de um sistema seja compartilhado por diversas máquinas virtuais. Para que seja feita a criação e gerenciamento destas máquinas virtuais faz-se necessário um software chamado *hypervisor*, responsável pela virtualização do hardware [PSL13]. Dentre os principais *hypervisors* podemos citar VMWare [VMW14], Hyper-V [Mic14], XEN [Xen14] e KVM [KVM14].

Entretanto, não basta apenas um *hypervisor* para que um ambiente de nuvem seja criado. Ainda é necessário utilizar um sistema responsável por orquestrar a utilização de

diferentes *hypervisors*, máquinas físicas, equipamentos de armazenamento de dados, etc. Além disso, um ambiente de nuvem deve ter uma interface para que os diferentes clientes do serviço consigam gerenciar suas próprias máquinas. Para esta finalidade existem sistemas chamados *Cloud Management Systems* (CMS), tais como OpenStack [Ope14], CloudStack [Clo14], Eucalyptus [Euc14], etc.

Em nível de aplicação, os CMSs, disponibilizam um mecanismo através do qual os clientes podem conectar seus próprios softwares, e assim gerenciar suas máquinas virtuais e serviços no ambiente de nuvem. Essa interconexão pode ser realizada através do modelo arquitetural definido via REST API ou também via SOAP[1]. No entanto, ao disponibilizar esse mecanismo de interconexão, faz-se necessário tomar uma série de medidas para evitar, principalmente, que um cliente mal intencionado consiga causar algum tipo de dano ao ambiente de nuvem, e desse modo comprometer as informações e sistemas de diferentes clientes [GSL13]. Uma dessas medidas é a criação de um mecanismo de autenticação através do qual o cliente deve informar um usuário e senha, ou mesmo um *token*, para que assim ele possa fazer uso das operações disponibilizadas.

Contudo, uma vez disponibilizado um mecanismo de acesso externo, aqui no caso via REST, há necessidade de garantir a segurança desta operação além do usuário e senha. Conforme será apresentado na Seção 4.2.3, a operação envolvida no mecanismo de autenticação pode ser utilizada indevidamente. Essa utilização ilícita pode acontecer durante a validação dos dados que o serviço recebe para autenticação (usuário e senha ou *token*), pois essa validação envolve uma consulta ao banco de dados. Desse modo, um usuário mal intencionado pode sobrecarregar o serviço através da utilização de uma operação, fazendo assim com que o sistema comece a deteriorar o tempo de resposta para os clientes legítimos.

Para resolver o problema mencionado no parágrafo anterior, o presente trabalho propõe um mecanismo de mitigação de ataques de negação de serviço, que opere em nível de aplicação, analisando as requisições feitas por clientes para uma REST, e com base no conteúdo da sua chamada limitar o acesso do mesmo. Esse mecanismo baseia-se no IP de origem dos clientes e em uma fila de controle do tempo em que cada cliente ficará impossibilitado de enviar novas chamadas. Como forma de demonstrar a viabilidade da solução proposta, será feita uma análise do componente Keystone, que é responsável pelo mecanismo de autenticação e controle de acesso do sistema de gerenciamento de nuvem OpenStack.

O presente trabalho está organizado da seguinte forma. O Capítulo 2 descreve vulnerabilidades que são encontradas em ambientes de nuvem. O Capítulo 3 apresenta uma análise de outros trabalhos que visam classificar e implementar um mecanismo de defesa

[1]O modelo de interconexão via SOAP é pouco utilizado. Atualmente os grandes provedores de nuvem, como por exemplo: Amazon, Microsoft Azure, Google, OpenStack, etc., adotam a utilização de interconexão através de REST API [Kle14, Cre12, PMP10].

contra ataques de negação de serviço. No Capítulo 4 são explorados os diferentes tipos de ataque de negação de serviço, dividido pelas camadas onde são executados. O Capítulo 5 apresenta como acontece o problema de um ataque de negação de serviço em uma serviço de REST que utiliza *token* para autenticação dos clientes, bem como mostra a solução desenvolvida neste trabalho. No Capítulo 6 é feito um estudo de caso utilizando o componente Keystone do OpenStack para mostrar a aplicação da proposta. Por fim, no Capítulo 7 são expostas as conclusões do presente trabalho bem como levantados apontamentos de trabalhos futuros.

2. AMEAÇAS À NUVEM

Apesar do conceito de computação em nuvem ter se consolidado no mercado há pouco tempo [Mel13], ele agrega uma série de ideias já conhecidas na área da Computação [Tau09]. Desse modo, além das ideias, são trazidos junto também seus pontos fracos, os quais podem e são explorados por pessoas mal intencionadas. As vulnerabilidades de uma nuvem acabam por se tornar um dos pontos fundamentais na hora de uma empresa decidir se vai adotar a utilização de um sistema em nuvem ou não.

Trabalhos como os de Kandukuri [KPR09], Zhou [ZZX+10] e Subashini [Sub11] conduzem diferentes enfoques de análise sobre problemas de segurança relacionados a ambientes de nuvem. Ainda, a *Cloud Security Alliance* (CSA) realizou um questionário envolvendo especialistas em Computação em nuvem com intuito de identificar as principais ameaças em ambientes de nuvem. Baseado nas respostas obtidas foi elaborado um relatório, chamado “As Notórias Nove Principais Ameaças a Computação em Nuvem de 2013” [All13], para auxiliar as organizações a entenderem as principais ameaças existentes em relação a ambientes de nuvem, e com isso ajudar na análise de riscos.

Com base nesse relatório, neste capítulo, será feito uma análise dessas nove ameaças que são: vazamento de dados, perda de dados, sequestro de contas ou tráfego de serviços, interfaces inseguras, negação de serviços, usuário mal intencionado, abuso de serviços da nuvem, investigação insuficiente e vulnerabilidades de recursos compartilhados. O principal objetivo deste capítulo é situar o problema a ser resolvido neste trabalho.

2.1 Vazamento de dados

Desde antes das empresas migrarem seus dados para os computadores, vazamento de dados era um grande problema, pois o estrago causado por uma informação caindo nas mãos de uma empresa concorrente pode ser catastrófico.

Considerado pelos especialistas como a principal vulnerabilidade dos ambientes de nuvem, vazamento de dados sempre esteve presente na realidade das empresas. Entretanto, com o crescimento da tecnologia da informação (TI), essa ameaça tende a crescer, pois os dados em meio digital aumentam, e com isto também acabam por virar alvo de criminosos virtuais.

Podemos notar esse problema ao vermos grandes empresas sofrendo com o vazamento de dados. Isso tem sido reportado frequentemente, por exemplo, em abril de 2011 a Sony [Bak13] sofreu um grande golpe ao ter vazado dados de aproximadamente 77 milhões de contas de usuários. Suspeita-se que o vazamento de dados aconteceu a partir do computador infectado de um dos administradores da rede *PlayStation Network* (PSN),

com isso, entre os dias 17 e 19 de abril foram comprometidos nomes, endereços, e-mails, usuários, senhas, perguntas de segurança, dentre outras informações. Assim, a Sony tirou do ar, emergencialmente, a PSN pelo período de 20 de Abril à 14 de Maio para implementar um mecanismo que garantisse a segurança da sua rede e de seus usuários. Segundo o instituto Ponemon [Dig13], esse vazamento de dados custou à Sony aproximadamente 171 milhões de dólares. Outro caso de um vazamento considerável de dados foi o ocorrido com a T.J. Maxx (uma grande rede de lojas de departamentos norte americana) em 2007. Durou de 2005 à 2007 e acabou por comprometer aproximadamente 40 milhões de transações de crédito. Estima-se que o custo desse incidente tenha sido na ordem de 25 milhões de dólares [Mil13]. Albert Gonzalez [Zet13] foi o *hacker* que coordenou esse ataque através da quebra do mecanismo de criptografia da rede sem fio (WEP). Uma vez acessando a rede da empresa, os *hackers* criaram uma conta no sistema e a utilizaram para conseguir capturar todas as transações de crédito que estavam sendo feitas [Esp13].

Diante destes casos, podemos notar que normalmente os vazamentos de dados acabam por ocorrer em conjunto com alguma outra vulnerabilidade do sistema, como por exemplo um computador infectado, um mecanismo de criptografia fraco ou até mesmo usuários despreparados, acabando por culminar num vazamento considerável de informações. Pesquisas como a apresentada por Du Meng [Men13] analisam os principais problemas relacionados a segurança de dados em ambientes de nuvem, desde o armazenamento dos dados até o tráfego das informações pela rede. No artigo de Yinqian Zhang [ZJR12] é mostrado um ataque a máquinas virtuais para assim capturar chaves de criptografia de dados.

Em se tratando de proteger as informações dos usuários é necessário saber quais informações devem ser protegidas. O *National Institute of Standards and Technology* (NIST) [MGK13] define a informação pessoal identificável (*Personally identifiable information* - PII [Wil07]) como sendo qualquer informação sobre um indivíduo que é mantida pela empresa e que possa ser vinculada a um usuário, permitindo assim que o mesmo seja identificado. Exemplos de informações definidas na PII: nome, numeração de documentos, data de nascimento, nome da mãe, registros médicos, registros financeiros, educação e referências a vínculos empregatícios. Ainda, há uma série de leis e padrões que são utilizados e também servem para auditoria, com intuito de garantir a segurança das informações pessoais: *Payment card industry - data security standards* - PCI DSS [Cou13]; *Federal Information Security Management Act* - FISMA [RKT05]; *Sarbanes Oxley Act* - SOX [Hal07]; *Health Insurance Portability And Accountability Act* - HIPAA [Cra09];

Sabendo dos problemas associados aos vazamentos de dados e quais são estas informações, então como mitigá-los e/ou minimizá-los? A resposta nem sempre é óbvia, pode envolver a utilização de alguma tecnologia como criptografia dos dados, para que somente quem possua a chave consiga acessar os dados, criando assim um ponto principal de segurança relacionada a chave do algoritmo de criptografia. Além de técnicas de pro-

teção de dados, contratos também devem ser considerados principalmente quando uma empresa é contratada como provedora do serviço de nuvem, pois nesse caso a provedora da nuvem tem responsabilidades sobre os dados armazenados em seu sistema.

2.2 Perda de dados

Uma vez que as empresas decidem por utilizar uma solução de computação em nuvem, esta inicia a migração dos seus dados para a nuvem. Estes dados como vimos anteriormente podem vazar de diversas maneiras e além do vazamento dos dados é necessário ter atenção especial também quanto à perda dos mesmos.

A perda dos dados pode acontecer de várias maneiras diferentes, porém podem ser enquadradas em dois tipos de perda: acidentais e intencionais. Como o próprio nome sugere, ocorrendo de maneira acidental, pode acontecer quando um usuário ou mesmo uma aplicação apaga, sobrescreve, altera os dados acidentalmente, sem intenção de prejudicar o proprietário das informações. Perda de dados de maneira intencional dá-se quando uma pessoa ou aplicação apaga os dados de maneira proposital, seja essa motivação para causar algum tipo de dano ao detentor da informação ou para tirar algum proveito da perda dos dados.

Um caso recente aconteceu com Mat Honan, escritor da revista Wired [Hon13], que teve suas contas do Gmail, Apple e Amazon invadidas, e como resultado dessa invasão todos os seus dados armazenados na nuvem, bem como os guardados em seus dispositivos de hardware, vinculados à Apple, apagados.

Sabendo dos tipos de perdas de dados, como uma empresa pode se preparar para evitá-las? O principal mecanismo para se evitar a perda de dados é a cópia de segurança. Um sistema eficiente de cópias de segurança garante que caso a empresa perca dados consiga minimizar ou mesmo evitar a perda. Quando uma empresa pensar em levar suas informações para um sistema de nuvem é imprescindível saber se o mesmo conta com um mecanismo de cópias de segurança, qual o tempo necessário para restaurar os dados perdidos, qual a frequência que as informações são copiadas e também se os mesmos são armazenados de maneira segura. Pesquisas como a apresentada por Cardellini [CI12] propõem um modelo de redundância de volumes para armazenamento dos dados.

A perda de dados pode trazer impactos negativos para a empresa, manchando a imagem, pois ela se mostra incapaz de guardar com segurança os dados de seus clientes. Além de denegrir a sua imagem, a empresa poderá enfrentar consequências judiciais, com isso acarretando prejuízos financeiros [Liu10b].

2.3 Sequestro de contas/tráfego de serviços

Sequestro de conta ou tráfego de serviço é um tipo de ataque que acaba por ser potencializado com o uso dos ambientes de computação em nuvem. Isso acontece porque num ambiente de nuvem pode haver diversos serviços e contas sendo utilizados, fazendo assim com que um usuário mal intencionado tenha um ponto único para explorar [BFZ07].

O sequestro de uma conta pode envolver outros ataques, como *phishing*, fraudes ou exploração de vulnerabilidades nos serviços já existentes, para alcançar seu objetivo. Uma vez que o atacante consiga acesso à conta, ele pode começar a monitorar transações, manipular os dados, distribuir informações falsificadas e até mesmo redirecionar clientes para sites maliciosos. Uma vez comprometida uma conta de serviço do ambiente da nuvem, esta pode servir de ponto inicial para diferentes tipos de ataques, desde vazamento de dados de outras contas que estejam na mesma nuvem, até mesmo para um ataque interno no próprio ambiente de nuvem, dentre outros [All13].

Em 2013 uma das grandes empresas de armazenamento de arquivos na nuvem chamada Dropbox foi alvo deste tipo de vulnerabilidade [Zor13]. Pesquisadores mostraram que a partir da decompilação do código do software cliente do Dropbox, que é instalado localmente, é possível interceptar o tráfego SSL oriundo do servidor e com isso o mecanismo de autenticação não é executado, tornando assim possível sequestrar a conta do usuário em questão. Em outro exemplo, podemos citar o caso da Amazon em 2010 [Goo13], onde foi encontrado o *bot* Zeus (cavalo de troia que rouba informações da máquina infectada) rodando no ambiente de nuvem da Amazon. Este *bot* conseguiu ser instalado a partir de uma outra conta que foi sequestrada, e com isso começou a se alastrar pelos clientes a medida que conectavam-se na Amazon. Após esta infecção inicial, o *bot* começava a armazenar informações e reportava os dados para os servidores maliciosos que estavam na nuvem.

Para prevenir esse tipo de ameaça é necessário que seja feito um controle com relação às credenciais das contas. Esse controle passa pela utilização de políticas de segurança que obriguem o usuário a alterar as senhas em períodos curtos. Deve também instruir os usuários sobre as ameaças e suas consequências caso as credencias sejam compartilhadas. Outro mecanismo que pode ajudar é a utilização de técnicas de autenticação forte em duas etapas, como por exemplo além do usuário informar a sua senha, também deve informar um *token* aleatório que é utilizado para uma operação e descartado.

2.4 Interfaces inseguras

Diversos provedores de ambientes de computação em nuvem disponibilizam interfaces (APIs) para que seus clientes as utilizem de modo a tirar o melhor proveito da

nuvem, muitas vezes através dessas APIs outras empresas conseguem implementar uma nova camada de software sobre a nuvem de modo a agregar valor ao seu serviço [All13].

Se por um lado essas APIs dão flexibilidade para que possa ser integrada uma nova camada de software a nuvem, por outro a sua implementação requer esforço extra para garantir seu correto funcionamento. Lu [LZB+13] em sua pesquisa apresenta um estudo das diversas falhas encontradas nas APIs disponibilizadas pela Amazon. Em seu trabalho é feito um detalhamento das falhas encontradas na Amazon, onde 60% dos problemas são relacionados a paradas com falhas, 19% são falhas de conteúdo (mensagens de erro, falta de dados, conteúdo errado e conteúdo não esperado), 12% são falhas por respostas demoradas e 9% relacionados a erros genéricos.

Sabendo dos problemas relacionados a API da Amazon, Lu [LZB+13] investigou as causas dessas falhas, onde concluiu que estas podem ser enquadradas em três principais categorias, relacionadas a problemas no desenvolvimento, falhas dos dispositivos físicos e falhas em interações entre os sistemas.

Diante dessa série de falhas encontradas em APIs disponibilizadas pela Amazon, um usuário mal intencionado poderia explorar estas falhas de modo a por exemplo sobrecarregar o servidor fazendo uma série de operações que gerem algum tipo de erro.

2.5 Negação de serviços

Negação de serviço consiste basicamente em fazer com que um sistema pare de responder ou mesmo responder de forma lenta. Para que o sistema deixe de responder, o ataque pode explorar problemas de implementação, através do qual o sistema fique preso em um estado, ou ainda fazer com que todos recursos disponíveis sejam consumidos. Esses recursos podem ser: processador (sobrecarregando o mesmo através de inúmeros processos sendo criados e executados, efetuando tratamento de erro, etc.), memória, espaço em disco e o principal recurso consumido: a rede. O consumo da rede acaba por ser o principal alvo de um ataque de negação de serviço, pois uma vez sobrecarregada a rede, o sistema terá um acesso muito lento ou até mesmo perda de acesso [All13].

Existem diversos tipos de ataques voltados ao consumo da banda de rede de um sistema, porém devido ao aumento da capacidade de processamento e de técnicas de defesa, os atacantes passaram a utilizar o ataque de negação de serviço distribuído [PMP10]. Desse modo acaba dificultando para o sistema alvo conseguir distinguir o tráfego malicioso do conteúdo verdadeiro que é recebido. Independente do ataque ser centralizado ou distribuído, há variantes em como o ataque é executado. Os tipos mais comuns desse ataque são: ICMP *flood*, SYN *flood*, SlowLoris, dentre outros [Che10].

Estes ataques de negação de serviço aumentaram a medida em que mais sistemas estão disponíveis *online*. Uma vez que um dos princípios da computação em nuvem é o conceito de recursos computacionais virtualmente infinitos, usuários podem criar contas em provedores de nuvem pública para desse modo gerar tráfego e fazer o ataque de negação em um sistema [Lem13]. Segundo O'Neil [ONe13], usuários mal intencionados também estão explorando as APIs com problemas, para assim comprometer a disponibilidade de um sistema hospedado em uma nuvem. Liu [Liu10a] em sua pesquisa apresenta uma forma de negação de serviço gerada por uma máquina de dentro de um ambiente de nuvem, onde a máquina mal intencionada gera tráfego de modo a sobrecarregar os roteadores onde a mesma se encontra ligada, desse modo todas outras máquinas que utilizam este roteador acabam por ser afetadas. CSA [All13] também considera o aspecto de que uma máquina em um ambiente de nuvem, durante um ataque de negação de serviço, acaba por consumir muitos recursos, desse modo o dono da máquina terá gasto financeiro maior, dado que na nuvem normalmente a cobrança é baseada na utilização dos recursos.

O principal malefício causado por um ataque de negação de serviço é relacionado com a insatisfação dos clientes do mesmo, uma vez que o sistema comprometido pode ficar lento e até mesmo fora do ar enquanto o mesmo estiver sob ataque.

2.6 Usuário mal intencionado

Segundo *Carnegie Mellon University's Computer Emergency Response Team* (CERT) [CER13] um usuário mal intencionado (*malicious insider*) é um empregado, parceiro ou terceirizado de uma empresa que possui acesso a rede, sistemas e/ou dados e intencionalmente utiliza-se deste acesso para afetar a confidencialidade, integridade ou disponibilidade dos dados ou sistemas da organização.

Duncan [DCG12] em seu trabalho apresenta os possíveis atores que podem atuar como usuários mal intencionados em um ambiente de nuvem. Este indivíduo mal intencionado pode estar atuando junto ao cliente que contratou o serviço de nuvem, pode estar trabalhando diretamente dentro da empresa que oferece o serviço de nuvem ou até mesmo pode atuar no provedor de Internet capturando informações não protegidas em trânsito.

O dano causado por um usuário mal intencionado atuando dentro da empresa que fornece a solução de computação em nuvem pode ser inestimado, pois esse usuário tendo acesso físico às máquinas, onde estão rodando os *hypervisors* (sistemas de virtualização), pode comprometer dados dos mais diversos clientes.

Para auxiliar na identificação de um possível usuário mal intencionado é necessário que sejam implementados mecanismos de auditoria dos sistemas, desse modo tentando inibir a ação do mesmo.

2.7 Abuso dos serviços da nuvem

Um dos grandes benefícios da utilização de computação em nuvem é a sua escalabilidade, sendo necessário para um cliente apenas pagar pelo que consumir. Com esse modelo de negócio qualquer usuário pode fazer uso de grande poder computacional oferecido pela nuvem apenas pagando pela utilização [All13].

Nesse cenário surge uma ameaça para o ambiente de computação em nuvem que é a possibilidade de um usuário criar uma conta na nuvem e com isso usar seus recursos para fins ilícitos, tais como distribuir algum tipo de *malware*, software pirata e até mesmo originar um ataque DDoS. Este tipo de ameaça acaba por impactar em grande parte o provedor da nuvem, visto que é sua responsabilidade evitar o abuso do uso dos serviços para fins ilícitos por seus usuários.

Este tipo de prática acaba por tornar difícil para um provedor de nuvem distinguir o uso ilícito de seus recursos de quem efetivamente está utilizando de maneira coerente. Pensando sobre este viés, Szefer [Sze13] em seu trabalho propõe um sistema de proteção através de medidas econômicas.

O modelo de Szefer é baseado em um depósito inicial feito pelos usuários ao contratar um serviço de nuvem. Este valor depositado inicialmente ficará como uma forma de garantia que o serviço não será utilizado para fins ilícitos. Caso o provedor de serviço detecte que o usuário está violando a política de utilização, este valor é mantido pelo provedor e o serviço sumariamente parado. Caso o usuário termine seu serviço e não tenha infligido as normas, o valor previamente depositado é creditado de volta para o mesmo.

2.8 Investigação insuficiente

Esta vulnerabilidade está associada às empresas migrarem seus sistemas para ambientes de nuvem sem ter o devido conhecimento das características específicas do ambiente de nuvem. Isto acaba por acontecer devido à pressão das companhias para adoção muito cedo do ambiente de nuvem [All13].

Muitas vezes, na pressa de ser a pioneira em determinada área, algumas empresas acabam por deixar de lado aspectos importantes na adoção de uma tecnologia. Por exemplo, aspectos relacionados aos acordos legais com o provedor do serviço de nuvem, que podem no futuro gerar impasses sobre as responsabilidades atribuídas a cada uma das partes quanto aos cuidados com as informações, disponibilidade dos serviços, dentre outros. Outro item é relacionado ao conhecimento da plataforma de nuvem. Essa plataforma possui diferenças em relação ao ambiente da empresa, e essas diferenças, como configurações de redes, compartilhamento de recursos, etc., devem ser levadas em consideração

quando a empresa decide migrar uma aplicação que rode no seu ambiente tradicional para um ambiente de nuvem.

Diante disso, quando uma empresa decide por adotar uma solução de nuvem para seus sistemas, ela deve investigar a fundo a tecnologia envolvida e tentar entender ao máximo os riscos que ela pode estar assumindo nessa transição.

2.9 Vulnerabilidades de recursos compartilhados

Uma das principais técnicas empregadas na criação de um ambiente de nuvem é o compartilhamento de recursos. Através dele é possível que diferentes clientes e aplicações possam ser executadas simultaneamente no mesmo processador, seus dados trafeguem em uma mesma rede, estejam armazenados no mesmo disco. Graças a esse compartilhamento de recursos, um ambiente de nuvem consegue dividir os custos envolvidos com a infraestrutura de hardware, entre os seus clientes [All13].

Como a virtualização é a principal tecnologia empregada na criação da nuvem, ela traz uma série de ameaças que existem pelo fato de que muitas das arquiteturas utilizadas não foram feitas esperando um uso compartilhado. O grande alvo de um ataque de recursos compartilhados acaba por ser o *hypervisor*, que é o software responsável por gerenciar as máquinas virtuais. Ele acaba sendo o foco dos ataques, pois uma vez comprometido, um usuário mal intencionado consegue atingir todas as máquinas virtuais que estão sendo executadas no mesmo *hypervisor* [PSL13].

Dawoud [DTM10], em seu trabalho, diferencia as ameaças a uma máquina virtual quando parte da máquina que hospeda o *hypervisor* tem capacidade de monitorar as informações da máquina virtual, bem como a comunicação que há entre as máquinas virtuais e o *host.* Com base neste caso, a partir do *host*, um usuário mal intencionado poderia ser capaz de capturar a comunicação entre as máquinas virtuais. Dawoud ainda analisa quando uma ameça é oriunda de outra máquina virtual, no caso uma máquina virtual tentar monitorar ou mesmo interceptar a comunicação da outra através da rede que acaba por ser compartilhada. Ainda, há a ameaça relacionada a mobilidade das máquinas virtuais, que podem migrar entre os diferentes *hosts* utilizados pelo *hypervisor*, onde um usuário mal intencionado venha a comprometer a integridade das mesmas.

2.10 Considerações

Diante de todas as ameaças aqui analisadas, e sabendo que cada uma delas pode afetar os princípios básicos da segurança que são: confidencialidade, integridade, autenti-

cidade, não repúdio e disponibilidade [TG14], a Tabela 2.1 apresenta de forma sumarizada quais princípios são afetados por cada uma das ameaças.

Tabela 2.1 – Princípios afetados por ameaças.

Princípio\Seção	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9
Confidencialidade	X		X	X		X		X	X
Integridade			X	X		X		X	X
Autenticidade			X	X		X		X	X
Não repúdio		X	X			X		X	X
Disponibilidade		X	X		X	X		X	X

Para cada uma das vulnerabilidades apresentadas existe algum tipo de ataque que pode explorá-la. Baseado nessas vulnerabilidades, a CSA [All13] apresenta uma série de ataques que podem utilizar as ameaças analisadas. Na Tabela 2.2 é apresentada de forma sumarizada qual ataque é aplicado a cada vulnerabilidade.

Tabela 2.2 – Ataques utilizados para cada vulnerabilidade.

Ataque\Seção	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9
Spoofing			X			X		X	
Adulteração			X	X		X		X	
Repúdio		X	X					X	
Vazamento de informações	X		X	X		X		X	X
Negação de serviço		X		X	X			X	
Elevação de privilégios			X	X				X	X

As vulnerabilidades de sequestro de contas (Seção 2.3), usuário mal intencionado (Seção 2.6), investigação insuficiente (Seção 2.8) e recursos compartilhados (Seção 2.9), de acordo com a Tabela 2.1, são as principais vulnerabilidades, pois acabam por afetar todos os princípios da segurança de um sistema de nuvem. E ainda, dos princípios apresentados, a confidencialidade e disponibilidade são os principais alvos das vulnerabilidades analisadas.

Podemos notar que o principal tipo de ataque voltado para ambientes de nuvem é o de vazamento de informações, ele é explorado por 6 das 9 vulnerabilidades apresentadas na Tabela 2.2. Também vemos que a vulnerabilidade de investigação insuficiente (Seção 2.8) é a que pode ser suscetível a qualquer um dos ataques, visto que o usuário do ambiente de nuvem não tem conhecimentos dos procedimentos adotados no provedor de serviço.

Devido ao recente crescimento dos ataques de negação de serviço em diferentes organizações, somado ao fato de que uma vez comprometido o CMS, todos os sistemas nele rodando tendem a sofrer o impacto, levando-os por exemplo à perda de dados, o presente trabalho foca-se em analisar este problema de negação de serviço.

3. TRABALHOS RELACIONADOS

Existem diferentes tipos de ataques de negação de serviço, desde ataques que acontecem em nível de rede até mesmo ataques em nível de aplicação. Esses ataques podem ser classificados de maneiras diferentes, levando em conta as características de cada tipo de ataque de negação de serviço.

3.1 Ataques em nível de transporte/rede

Kargl [KMW01] fez uma análise dos primeiros ataques de negação de serviço, bem como sua variação para ataques distribuídos, executados através de máquinas infectadas por *daemons,* as quais permitem que um usuário malicioso consiga controlá-las remotamente. Como proposta para defesa contra este tipo de ataques, ele desenvolveu uma alteração no *kernel* do Linux incluindo um mecanismo de roteamento de pacotes através de *round robin* (os pacotes são distribuídos dentre os servidores que atendem o serviço) e *least connection* (os pacotes são distribuídos com base no estado das conexões que o mesmo mantêm, pois cada conexão pode ficar aberta por diferentes períodos de tempo). Com essa alteração ele utiliza este Linux operando com intuito de fazer um balanceamento de carga, para assim distribuir as requisições homogeneamente entre os servidores. Esta solução opera em nível de rede validando os pacotes recebidos e descartando-os caso o cliente faça parte da fila de clientes suspeitos. Este cliente é marcado como suspeito uma vez que seu comportamento seja similar aos comportamentos definidos nos ataques de negação.

Um sistema de defesa colaborativo contra ataques de negação de serviço é proposto por Tariq [TMA11]. Esse sistema de defesa é implantado nos diferentes nodos que compõem a rede, desse modo ao detectar algum tipo de tráfego malicioso, o nodo em questão emite um alerta para os outros nodos, e assim os pacotes acabam por ser filtrados, evitando que o ataque consiga chegar no computador alvo. Esse controle de pacotes é feito com base em uma análise do tráfego durante uma janela de tempo. Os dados coletados são comparados com as assinaturas de ataques de negação de serviço já conhecidas, como por exemplo um ataque do tipo SYN *Flood*, o qual tem como característica a não finalização do TPC *handshake* na abertura da conexão, conforme será visto na Seção 4.1. A limitação deste trabalho é que por atuar analisando pacotes em nível de rede, ataques que abram uma conexão válida com o computador alvo e então iniciem o envio de pacotes maliciosos acabam por atingir seu objetivo.

3.2 Ataques em nível de aplicação

O trabalho feito por Beitollahi [Bei12b] apresenta um mecanismo de defesa contra ataques de negação de serviço que operam em nível de aplicação. Em seu artigo, Beitollahi cria um mecanismo que atribui diferentes pontuações às conexões, baseado em uma análise histórica e estatística do estado da mesma. No entanto, a solução apresentada por Beitollahi necessita que o computador cliente seja operado por um humano, pois o computador alvo ao detectar que está sendo atacado envia um desafio através de CAPTCHA [VBL04] (*Completely Automated Public Turing test to tell Computers and Humans Apart*) para o cliente, o que não seria aplicável quando dois sistemas computacionais estivessem interagindo via uma interface REST.

Apesar do mecanismo de CAPTCHA ser amplamente utilizado em sistemas computacionais para diferenciar os usuários seres humanos dos usuário sistemas (robôs), existem pesquisas que mostram já a possibilidade de sistemática de comprometer esse mecanismo de defesa. Yan [Yan08] em seu trabalho mostra como através da análise das cores da imagem é possível buscar pelo padrão dos caracteres.

O mecanismo de defesa contra ataques de negação de serviço, proposto por Schmerl [SCG+14], atua na camada de aplicação. Em seu trabalho ele definiu uma lista de potenciais táticas a serem aplicadas diante do ataque de negação de serviço. A Tabela 3.1 apresenta esta lista de táticas, que são utilizadas como base para a criação das estratégias de defesa. A estratégia de defesa consiste na combinação das táticas identificadas e sua aplicação durante o ataque. Nesta combinação são utilizadas 3 diferentes estratégias: Desafio, utiliza as táticas de CAPTCHA e forçar autenticação; Eliminação, utilizando as táticas de *Blackhole* e regulação de acesso; e Superação, combinando aumento da capacidade e diminuição do serviço. A escolha da estratégia aplicada é feita com com base no resultado que se deseja obter, pois dependendo da escolha, pode ser criado um efeito colateral indesejado para os usuários legítimos. Por exemplo, se for adotada a estratégia de uso de Desafio, quando quem faz uso da aplicação é um outro sistema, faria com que o mesmo parasse de funcionar, tendo em vista que essa solução atua de modo a diferenciar humanos de máquinas.

Em busca de identificar os problemas relacionados às REST APIs dos sistemas de nuvem, Lu [LZB+13] em seu trabalho conduz um estudo empírico sobre as APIs utilizadas pela *Amazon Elastic Compute Cloud* (EC2). Sua pesquisa foi motivada pelo grande número de relatos de usuários enfrentando problemas com o uso das APIs disponibilizadas pela EC2 (53% dos problemas relatados eram relacionados com o uso da API). Lu classificou os problemas em: falhas de conteúdo, atraso no tempo de resposta, falhas que ocasionam paradas e falhas erráticas. Dentre estas falhas identificadas, o percentual de cada tipo de falha foi respectivamente: 19%, 12%, 60% e 9%. Com esse trabalho podemos notar que

Tabela 3.1 – Lista de potenciais táticas para defesas de ataques de negação de serviço

Aumento da capacidade	Adição de recursos de modo a superar o ataque.
Diminuição do serviço	Simplificação das consultas para permitir mais requisições.
Captcha	Evitar ataque vindo de robôs.
Forçar autenticação	Evitar ataque vindo de robôs.
Blackhole	Ignora requisições oriunda de atacantes.
Regulação de acesso	Limita o número de requisições de cada cliente.

a grande maioria das falhas apresentadas pelo serviço EC2 estavam relacionadas com o ocasionamento de paradas no sistema, deixando o usuário da API preso em um estado, impossibilitado de realizar outras operações. Isso nos leva a dar atenção especial para as APIs que são utilizadas no gerenciamento de um ambiente de nuvem.

Barna [BSS+14] propõe uma solução considerando uma arquitetura adaptativa baseada em modelos pré-definidos. Ele descreve um modelo para prever o impacto do tratamento das requisições, com base na análise do hardware, e com isso um conjunto de regras de filtragem é instanciado de modo a analisar as requisições suspeitas. A definição deste modelo é feita através de um sistema chamado OPERA [Lit07], que tem como finalidade monitorar o uso do hardware consumido pelas requisições, e desse modo, sua solução consegue identificar pontos de sobrecarga no hardware. Ao receber as requisições o *firewall* dinâmico verifica se a requisição irá sobrecarregar o serviço. Ao identificar o tráfego suspeito, o mesmo aciona o mecanismo de analise do usuário, onde este mecanismo envia um desafio (CAPTCHA) para que o usuário prove que ele não é um robô.

3.3 Classificação

Um ataque de negação de serviço pode acontecer de diferentes maneiras e visando diferentes vítimas. Com base nas diferentes características envolvidas em um ataque de negação de serviço, Mikovic [Mir04] propõe uma taxonomia para classificação dos tipos de ataques.

A Figura 3.1 apresenta uma simplificação desta taxonomia. Conforme a classificação, por exemplo, um ataque pode ter uma origem falsificada, como quando o atacante forja os pacotes de modo a tentar esconder a sua verdadeira identidade, ou a origem pode ser válida. A fraqueza explorada pelo ataque pode ser semântica, quando o ataque visa explorar uma vulnerabilidade do protocolo/aplicação, ou força-bruta, quando o ataque utiliza mais recursos computacionais que o sistema alvo, visando assim esgotar seu poder de resposta. Durante um ataque o número de pacotes enviados é utilizado para determinar

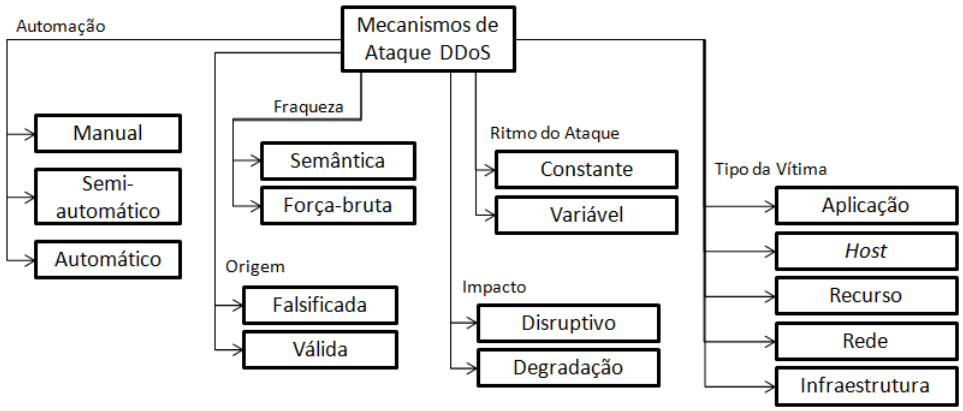

Figura 3.1 – Taxonomia de um ataque DDoS adaptada do trabalho de Mirkovic [Mir04]

sua classificação, que pode ser constante, quando o atacante envia os pacotes a uma taxa de envio igual, sem variações ou então variável, se o atacante oscilar randomicamente o tempo de envio de cada pacote. O ataque gerado pode ser considerado disruptivo quando deixa o seu alvo inacessível para os clientes legítimos, ou o ataque pode simplesmente degradar a performance do seu alvo deixando-o lento. Quanto ao alvo, as vítimas podem ser classificadas quanto à aplicação, quando o ataque visa comprometer apenas uma aplicação específica, *host* quando se busca comprometer o servidor que está hospedando o sistema, um recurso específico ao tentar, por exemplo, consumir o espaço de disco do alvo, vitimando a rede para um ataque que busque causar o congestionamento da rede.

3.4 Considerações

Apesar de haver um grande número de propostas de trabalhos que visem solucionar problemas relacionados a ataques de negação de serviço, como por exemplo: Brustoloni [Bru02], Moustis [Mou13], Patel [PMP10], Damon [DDL+12], Xueping [Che10], Udhayan [UA09], Al-Duwairi [ADM05], etc., este trabalho se focou nas pesquisas apresentadas por possuírem mecanismos que embasaram o desenvolvimento da solução proposta. Esta pesquisa de trabalhos se deu através da metodologia de *snowball,* identificando trabalhos relacionados a ataques de negação de serviço em serviços web de REST, bem como ameaças envolvidas a ambientes de nuvem, e a partir desta pesquisa definido este estudo.

A Tabela 3.2 apresenta de maneira sumarizada os trabalhos analisados, e divide-os de acordo com o nível onde o ataque é feito, a forma de defesa proposta bem como o tipo de vítima e a fraqueza explorada de acordo com a classificação proposta por Mirkovic. Desse modo, após uma análise dos trabalhos identificamos uma série de técnicas aplicadas nos mesmos, que podem ser aplicadas no presente trabalho. A criação de uma lista de suspeitos auxilia a traçar o comportamento dos usuários e assim dar prioridade aos que não cometem algum tipo de ação maliciosa. Entretanto, essa lista de clientes suspeitos deve

Tabela 3.2 – Sumário dos trabalhos analisados

Autor	Nível	Defesa	Tipo de vítima	Fraqueza
Kargl	Rede/Transporte	Balanceamento de carga entre os servidores, e descarte dos pacotes dos clientes da lista de suspeitos	Rede	Semântica
Tariq	Rede/Transporte	Composto por nodos da rede, analisa os clientes durante uma janela de tempo para filtrar os pacotes.	Rede	Semântica
Beitollahi	Aplicação	Atribui pontuações as conexões, cliente suspeito recebe um desafio de CAPTCHA.	Aplicação	Força-bruta
Schmerl	Aplicação	Baseado em uma tabela de táticas de ataque que utiliza uma estratégia de defesa associada.	Aplicação	Semântica
Barna	Aplicação	Algoritmos adaptativo que analisa o modelo de uso do hardware. Usa CAPTCHA quando encontra suspeitos.	Aplicação	Semântica

armazenar apenas informações sobre as operações realizadas pelo cliente durante um determinado período de tempo, para tal ação, é utilizada uma janela deslizante de tempo. Nos trabalhos de Barna e Beithollahi é lançado um desafio de CAPTCHA quando identificado um cliente suspeito. Contudo, esse desafio torna-se inviável quando o cliente do sistema for outro sistema.

4. ATAQUE DE NEGAÇÃO DE SERVIÇO

A segurança de sistemas de informação é fundamentada nos seguinte princípios: confidencialidade, integridade, autenticidade, não repúdio e disponibilidade [TG14]. Assim, o ataque de negação de serviço, para deixar um sistema não acessível para clientes legítimos, acaba por buscar comprometer o princípio da disponibilidade do sistema. Esse ataque pode ser feito de uma série de maneiras diferentes, tais como: consumir a banda de conexão de um sistema, utilizar todas as conexões disponíveis, sobrecarregar o processador do alvo, dentre outros. Conforme mencionado na Seção 3.3, Mirkovic [Mir04] propôs uma taxonomia para classificar os ataques com base nas características apresentadas por eles (veja Figura 3.1).

Apesar da taxonomia de Mirkovik levantar uma série de diferentes características dos ataques, não foi considerada uma diferenciação quanto ao nível da camada de rede onde o ataque é executado. Desse modo, no presente trabalho os ataques de negação de serviço serão diferenciados quanto ao nível em que o mesmo é executado. Sendo assim, será feita a divisão de ataques quando executados nos níveis de rede/transporte e de aplicação.

4.1 Nível de Transporte/Rede

Ataques de negação de serviço realizados no nível de transporte/rede são realizados na camada de nível 4 e 3 do modelo OSI [Zim80]. Esse tipo de ataque acontece quando o usuário malicioso gera um grande número de requisições para o sistema alvo, consumindo assim todos os recursos disponíveis nesse sistema.

Atualmente, os sistemas que são disponibilizados *online* são hospedados em computadores de grande porte ou com um grande poder computacional, ou mesmo em ambientes de nuvem. Uma característica do ambiente de nuvem é que a medida que houver uma maior demanda por recursos (processador, disco, banda de rede, etc.), estes são alocados dinamicamente gerando apenas um custo maior para o proprietário do sistema. Desse modo, essas medidas quando adotadas na implantação de um sistema seriam o suficiente para evitar um ataque de negação de serviço.

Para conseguir que um ataque de negação de serviço seja bem sucedido nesse cenário, para o usuário mal intencionado, não basta apenas gerar requisições a partir de seu próprio computador. Assim, surge o conceito de ataque de negação de serviço distribuído (*Distributed Denial of Service* - DDoS) [Bei12a], que consiste neste usuário mal intencionado recrutar uma série de computadores para gerar tráfego suficiente para efetuar o ataque.

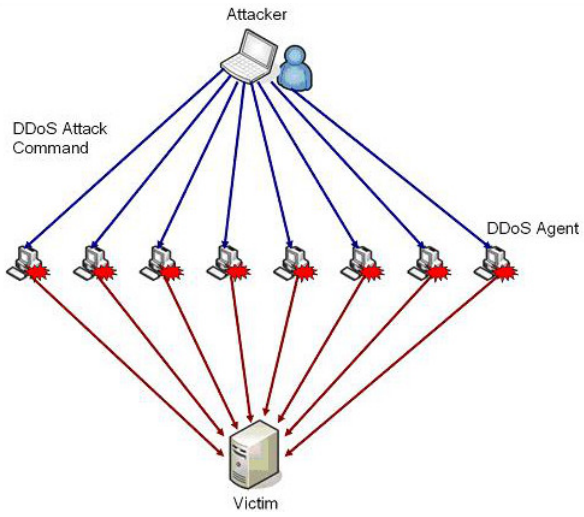

Figura 4.1 – Cenário de um ataque de negação de serviço distribuído

A Figura 4.1 apresenta o cenário onde o usuário mal intencionado utiliza uma série de computadores zumbi para atacar um sistema. Um computador zumbi é um computador que é infectado com algum tipo de software malicioso, que permite ao atacante controlar esse sistema remotamente. Desse modo, basta que o usuário mal intencionado instrua os computadores zumbis para iniciarem o ataque que os mesmos irão começar o ataque ao sistema alvo.

A necessidade da utilização desse grande número de computadores zumbis pode ser um empecilho para perpetrar um ataque de negação de serviço. Tentando facilitar a vida dos atacantes de um sistema, é apresentado o conceito de *botnet*, que consiste de uma rede de computadores interligados, nos quais está instalado algum tipo de software agente, o qual permite que o computador seja controlado remotamente. Essas *botnets* costumam ser ofertadas como um serviço, onde qualquer interessado pode pagar para utilizá-lo. Sendo assim, basta que o usuário mal intencionado pague pelo serviço de uma *botnet* para que o mesmo consiga disparar um ataque de negação de serviço contra um sistema alvo.

Independente do atacante de um sistema utilizar o serviço de uma *botnet*, controlar uma série de computadores zumbis ou até mesmo disparar o ataque de sua própria máquina, estes ataques podem explorar diferentes vulnerabilidades do sistema alvo através da camada de rede. Dentre estes ataques podemos citar SYN *Flood* [YZ08], refletido [ADM05], ICMP *Flood* [UA09], *teardrop* [Bru02], etc.

4.1.1 SYN *Flood*

Neste tipo de ataque [YZ08] o usuário mal intencionado se vale do funcionamento da abertura de uma conexão utilizando o protocolo TCP. A Figura 4.2 mostra como funciona

o estabelecimento de uma conexão através do protocolo TCP. Uma conexão, no protocolo TCP, é aberta após executar o chamado *three way handshake*, que consiste no cliente enviar um pacote do tipo SYN, que serve para sinalizar ao servidor que o cliente deseja abrir uma conexão. O servidor, ao receber este pacote, responde para o cliente com um pacote SYN-ACK, para informar ao cliente que o servidor recebeu o pacote inicial e o mesmo está agora aguardando o estabelecimento da conexão. A conexão é efetivamente inicializada quando o cliente, após receber o SYN-ACK, responde para o servidor através de um pacote ACK. Nesse ponto a conexão TCP está aberta e começa a troca de informações entre cliente e servidor.

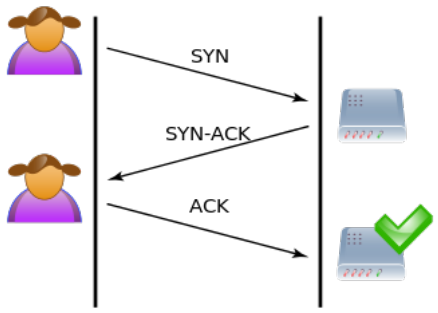

Figura 4.2 – Conexão TCP

O ataque de negação de serviço do tipo SYN *Flood* acontece quando o usuário mal intencionado explora o limite de conexões que o servidor consegue manter abertas aguardando pelo pacote de ACK do cliente.

No cenário do ataque, o usuário mal intencionado envia pacotes do tipo SYN para o servidor, se passando por um cliente legítimo, como se estivesse requisitando a abertura de novas conexões. O servidor responde para o cliente com o SYN-ACK, e nesse momento o protocolo TCP prevê que o servidor deve aguardar por um período de tempo para que o cliente responda com o pacote ACK. Sabendo disso, o atacante, depois de receber o pacote SYN-ACK do servidor, descarta-o e não envia o pacote ACK, fazendo com que o servidor fique aguardando pelo estabelecimento da conexão. Ao utilizar um grande número de máquinas zumbi para enviar apenas pacotes do tipo SYN, o atacante consegue esgotar os recursos de conexão do servidor, e quando um cliente legítimo tentar abrir uma conexão, o mesmo terá sua requisição negada.

A Figura 4.3 apresenta a sequência de pacotes enviados durante o ataque do SYN *Flood*. O usuário malicioso sobrecarrega o servidor com uma sequência de pacotes SYN, e uma vez que o servidor está aguardando pelo pacote ACK, os clientes legítimos não são atendidos.

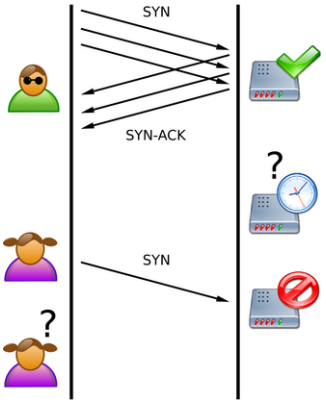

Figura 4.3 – Ataque SYN *Flood*

4.1.2 Refletido

O ataque de negação de serviço refletido [ADM05] acontece quando o usuário mal intencionado cria um pacote contendo no endereço de resposta do pacote o endereço da sua vítima. A Figura 4.4 mostra o cenário onde o ataque de negação refletido é efetuado.

Uma vez que os computadores zumbis recebem este pacote, eles respondem para o computador identificado no pacote, que neste caso é o endereço da vítima. Assim o sistema alvo recebe uma inundação de pacotes que ele não havia solicitado, e o computador alvo acaba por ser sobrecarregado precisando fazer o tratamento destas requisições.

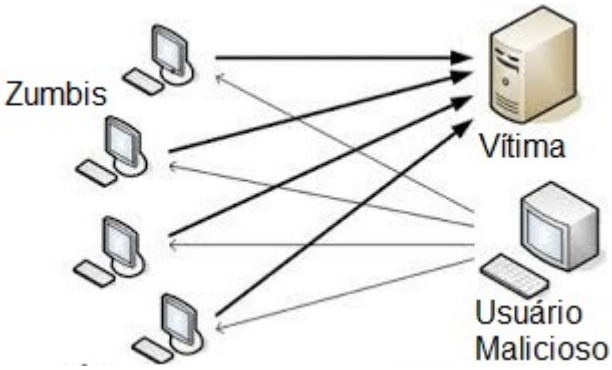

Figura 4.4 – Ataque refletido

Uma das variações de um ataque refletido é o ataque via *Network Time Protocol* (NTP). O protocolo NTP é utilizado para sincronização do relógios dos computadores. Esse protocolo conta um comando chamado *monlist*, onde o servidor retorna uma lista com os últimos 600 sistemas que o contactaram.

A Figura 4.5 mostra a variação deste ataque utilizando o protocolo NTP. Nele o usuário malicioso envia um pacote forjado para uma série de servidores NTP de forma a executar o comando de listagem. Nesse pacote, ele altera a informação referente ao remetente do pacote com o endereço do computador alvo. Assim, ao efetuar a consulta

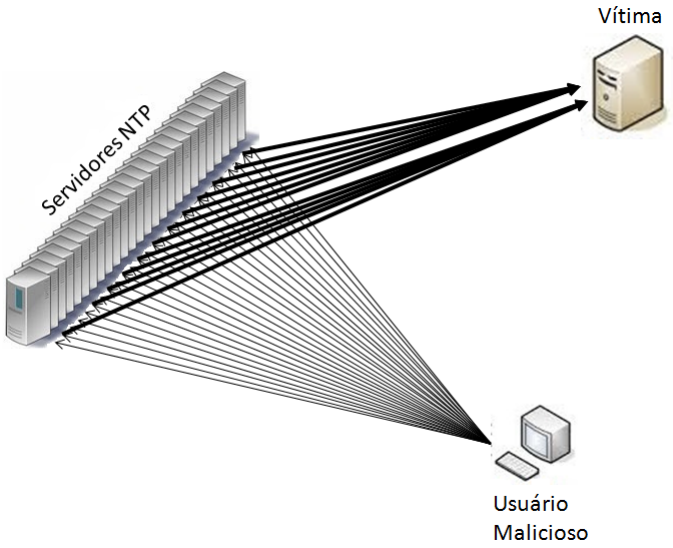

Figura 4.5 – Ataque refletido NTP

NTP nos servidores, os mesmos passam a responder todos para a vítima, fazendo assim com que o sistema alvo seja sobrecarregado.

Normalmente os ataques efetuados em nível de rede podem ser mitigados ou mesmo evitados através da utilização de regras para controle de pacotes recebidos através de um *firewall,* ou mesmo aplicação de um sistema de detecção de intrusão configurado para reconhecer as assinaturas de cada um destes tipos de ataques.

4.2 Nível de Aplicação

Ataques efetuados em nível de aplicação são difíceis de detectar através do uso de *firewalls*. Filtragem de pacotes neste nível é computacionalmente caro e difícil de ser escalável, e ainda, pode gerar muitos falsos positivos, bloqueando assim clientes legítimos.

Um ataque efetuado neste nível normalmente não requer um grande número de máquinas envolvidas, por exemplo utilizando uma *botnet*, pois este tipo de ataque visa comprometer especificamente uma aplicação, seja explorando alguma vulnerabilidade no protocolo utilizado ou mesmo na própria aplicação. Dentre os ataques mais comuns de negação de serviço em nível de aplicação, podemos citar Slowloris [Mou13], *are you dead yet* (RUDY) [DDL+12], etc.

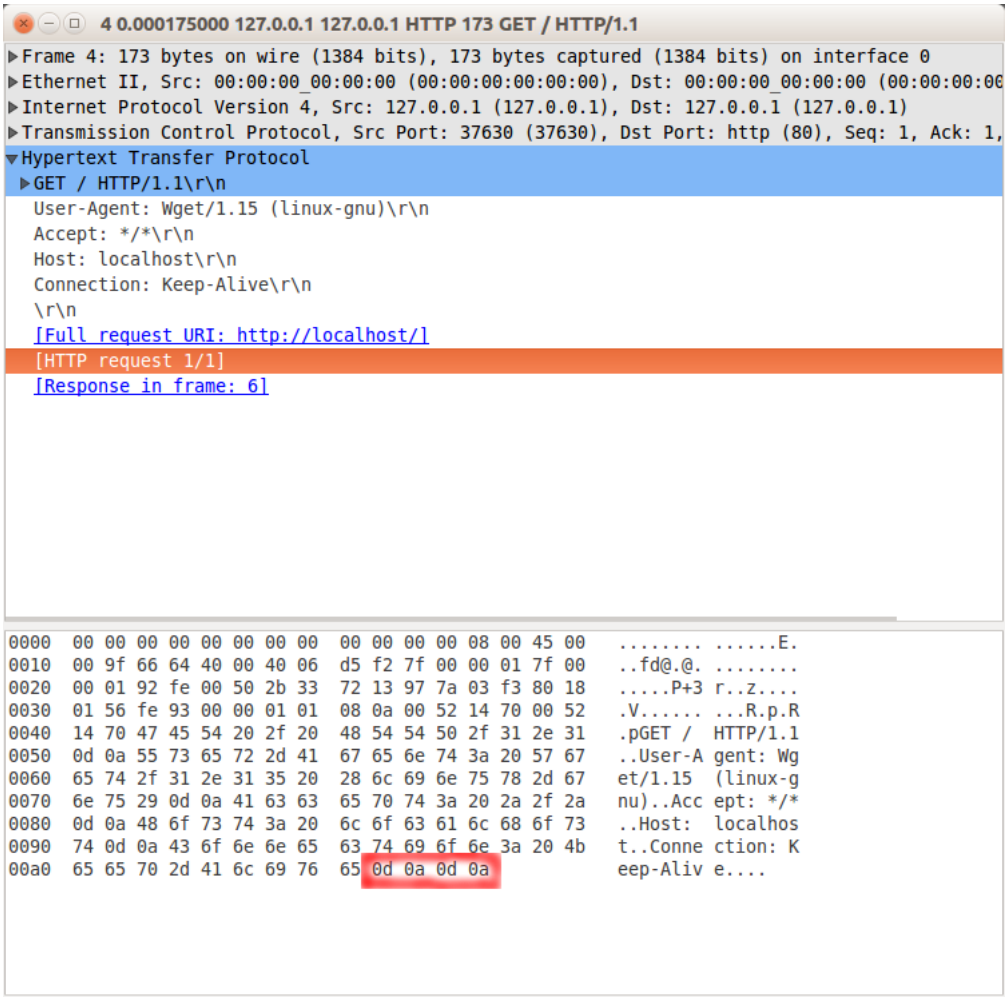

Figura 4.6 – Pacote HTTP original

4.2.1 Slowloris

O ataque Slowloris [Mou13] utiliza uma vulnerabilidade na operação de GET do protocolo HTTP, pois os servidores HTTP esperam sempre até receber um *header* da requisição completa. Essa requisição completa é marcada pela sequência de caracteres 0d 0a 0d 0a.

A Figura 4.6 apresenta uma requisição com o cabeçalho devidamente formado e enviado a um servidor HTTP. No final da requisição temos a sequência 0d 0a 0d 0a, que representa respectivamente *Carriage return*(CR - \r) e *Line Feed*(LF - \n). Ao ser repetido duas vezes CRLFCRLF indica a completude da requisição. Sabendo disso, o ataque não envia a requisição completa. Ao final do pacote é inserido apenas uma sequência de CRLF conforme mostrado na Figura 4.7. É também mostrado um exemplo de um pacote, capturado através da ferramenta WireShark, gerado durante o ataque Slowloris. Desse modo, o servidor mantêm uma sessão aberta, aguardando pela finalização do cabeçalho desta requisição. Assim, o ataque consiste em enviar uma série de tantas requisições quanto o

servidor HTTP consiga atender, esgotando todas as sessões disponíveis e mantendo-as abertas, fazendo assim com que o servidor deixe de atender a clientes legítimos.

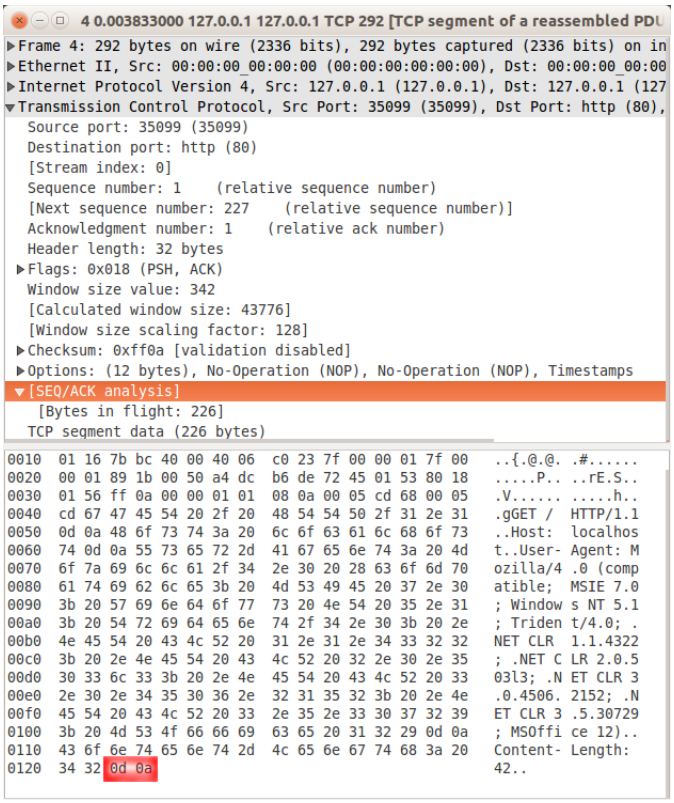

Figura 4.7 – Pacote gerado pelo ataque Slowloris

Uma possível forma de evitar, ou mesmo mitigar, este tipo de ataque é a configuração de um mecanismo de balanceamento de carga, que ao receber as requisições analise-as, e apenas as que possuírem um *header* completo sejam repassadas ao servidor.

Outro meio de mitigação pode ser através da configuração do módulo *netfilter* para sistemas Linux, de modo a limitar o número de conexões abertas por cada cliente.

4.2.2 *Are you dead yet* (RUDY)

Este ataque, ao contrário do Slowloris, que se baseia no funcionamento do método GET do protocolo HTTP, utiliza o método POST para comprometer seu alvo. E este ataque é difícil de ser detectado, pois ele gera chamadas válidas para o servidor em uma taxa de envio muito baixa com tamanho pequeno.

O método POST do protocolo HTTP é utilizado por formulários na Internet para enviar informações para os servidores. Desse modo, o ataque é voltado para servidores web que esperam receber os dados de um formulário. A Figura 4.8 mostra o cenário do ataque, nele o atacante fragmenta a informação que seria enviado pelo formulário em requisições

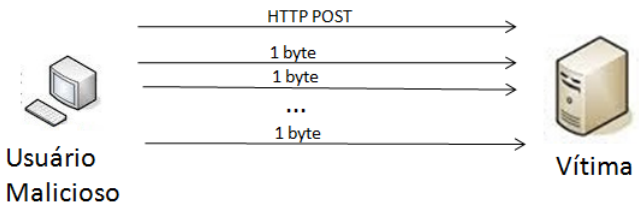

Figura 4.8 – Cenário do ataque RUDY

de 1 byte, estas são enviadas em intervalos de tempo randômico. Assim, o servidor web não encerra a conexão por acreditar se tratar de um cliente com uma conexão intermitente ou mesmo lenta.

Como possível mecanismo de defesa contra este tipo de ataque, pode-se configurar o servidor web para que defina o tamanho mínimo de cada requisição oriunda do método POST.

4.2.3 Ataque via REST API

Conforme a definição da REST API (*Representational State Transfer*) proposta na tese de doutorado de Fielding [Fie00], a REST é um estilo abstrato de elementos arquiteturais em um sistema distribuído. REST não foca na implementação dos componentes ou mesmo do protocolo, mas sim foca no papel dos componentes e suas restrições.

Atualmente, muitas requisições feitas na Internet se baseiam no estilo arquitetural REST [Fie00]. Chamadas REST são uma parte fundamental na arquitetura e implementação de sistemas, pois através delas diferentes operações são expostas e, dessa forma, diferentes aplicações podem utilizá-las. Uma vez criado um serviço REST, ele deve ser disponibilizado para acesso através de uma rede, e assim esta REST acaba por ficar exposta a usuários mal intencionados que queiram por algum motivo deixar este sistema indisponível. Um dos meios que estes usuários utilizam é através de ataques de negação de serviço.

Os serviços expostos via REST podem ser divididos em: serviços que não exigem nenhum mecanismo de autenticação, onde o usuário pode executar alguma operação disponível sem a necessidade de usuário e senha[1], e serviços que exigem algum mecanismos de autenticação, normalmente feita através de usuário e senha. Este último cenário é o modo como a grande parte dos serviços web disponibiliza acesso a sua REST API. Por esse motivo, o presente trabalho é focado em analisar problemas envolvendo o mecanismo de autenticação.

A Figura 4.9 apresenta a sequência de passos que acontecem durante uma operação típica via REST com autenticação. Inicialmente o cliente acessa o endereço do servidor

[1] Atualmente é muito difícil alguma empresa disponibilizar algum serviço web que não requer algum tipo de autenticação, desse modo o presente trabalho não discutirá a respeito deste tipo de serviço REST.

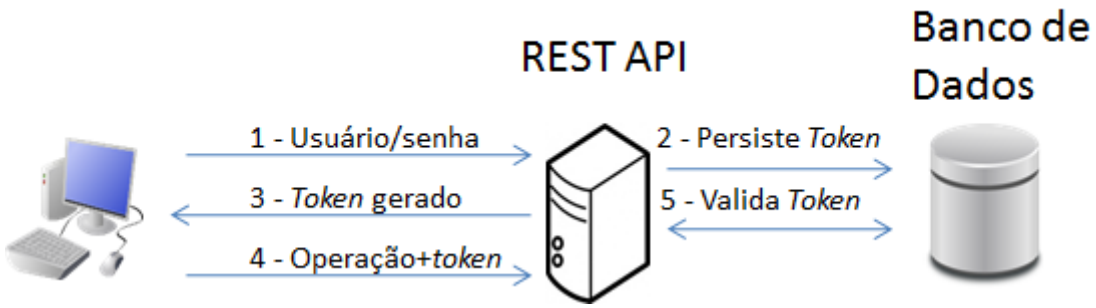

Figura 4.9 – Fluxo normal de chamada da REST API com autenticação

que expõe a REST e efetua uma chamada (1) enviando usuário e senha através de uma conexão segura. O servidor por sua vez valida estas informações, e uma vez estando corretas, gera um *token*, que é persistido no seu banco de dados (2). A seguir retorna esse *token* para o usuário (3), desse modo evita que o mesmo tenha que enviar usuário e senha a cada nova requisição. Assim, toda vez que o usuário necessitar executar uma das operações disponíveis na REST ele deve enviar este *token* (4), para que o serviço avalie se o usuário tem permissão para executar a operação ou não. Essa avaliação é feita consultando inicialmente se o *token* realmente existe (5). Caso o *token* exista no banco e ele tenha permissão, então a operação é executada.

Estes serviços REST acabam por disponibilizar uma série de operações para as quais é necessário que o usuário esteja autenticado. Esta ação de validação do *token* se torna um alvo relativamente atrativo para um usuário mal intencionado, pois ele pode enviar *tokens* inválidos com o tamanho máximo suportado pela REST, assim fazendo com que o servidor seja sobrecarregado com uma série de validações desnecessárias, levando a uma degradação dos serviços disponibilizados para os clientes legítimos.

O problema da sobrecarga do servidor acontece porque a cada requisição recebida, o mesmo deve efetuar uma consulta ao banco de dados para validar o *token* recebido. Esta consulta acaba por consumir os recursos de processamento do servidor. Mecanismos tradicionais de detecção e bloqueio de tráfego de ataques de negação de serviço acabam por não surtir efeito neste cenário, pois todas as chamadas são consideradas como tráfego real sendo gerado, e os sistemas de detecção atuam em nível de rede (ver Seção 4.1) analisando os pacotes recebidos. Mesmo trabalhos onde a detecção é feita em nível de aplicação (ver Seção 4.2), em geral, o mecanismo de defesa espera que do outro lado (cliente) exista um usuário [Bei12b].

A Figura 4.10 apresenta um cenário onde diversos usuários mal intencionados geram tráfego malicioso de forma a consumir os recursos do servidor, e fazendo assim que um cliente real tenha as suas chamadas deterioradas ou até mesmo negadas. Neste caso, os clientes maliciosos, sabendo do endereço do serviço, fazem chamadas de uma operação, e nessa operação é enviado um *token* inválido. O serviço da REST, para cada uma das chamadas, precisa fazer uma consulta ao banco de dados para identificar a validade

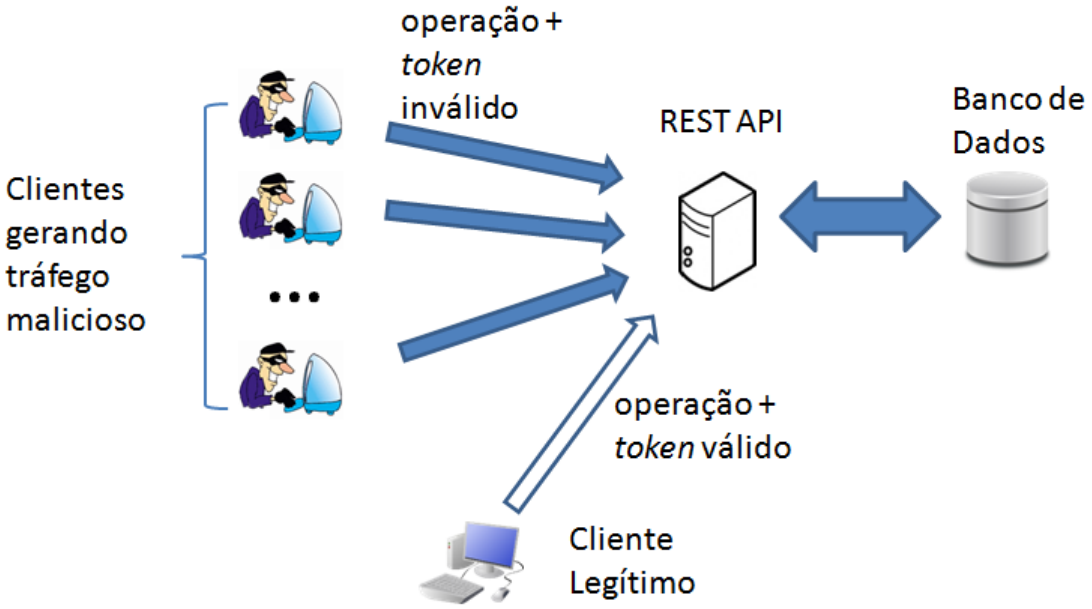

Figura 4.10 – Cenário do ataque ao servidor que disponibiliza a REST API.

do *token*, que nesta situação estará fazendo uma série de validações desnecessárias pelos *tokens* serem inválidos. Um cliente legítimo do serviço, ao executar a chamada com um *token* válido será penalizado com um tempo de resposta muito alto ou mesmo tendo sua requisição não atendida, pois o serviço pode estar sobrecarregado.

4.3 Considerações

É difícil existir uma única e simples solução para conter um ataque de negação de serviço. Conforme visto, os ataques acontecem em diferentes camadas ou ainda explorando diferentes problemas que os protocolos de comunicação possuem. Assim, faz-se necessário utilizar diferentes soluções combinadas de modo a buscar por uma proteção próxima à ideal.

A Figura 4.11 mostra um cenário onde um usuário mal malicioso pode utilizar diferentes tipos de ataques de negação de serviço para comprometer sua vitima. Desse modo, é mostra como podem ser combinadas as soluções de utilização de um *firewall* para contenção de ataques que operem em nível de rede, como o SYN-*Flood*; em um nível mais elevado é utilizado o módulo *NetFilter* ou *LoadBalancer*, com o intuito de barrar ataques de negação que utilizem a técnica do SlowLoris; ao se efetuar uma configuração adequada do *web server* consegue-se evitar que ataques do tipo RUDY consumam os recursos de um sistema alvo; e, por fim, ao implementar a utilização da proposta do MADRA (Capítulo 5), conseguiríamos proteger o sistema alvo contra ataques com sobrecarga de *tokens* inválidos.

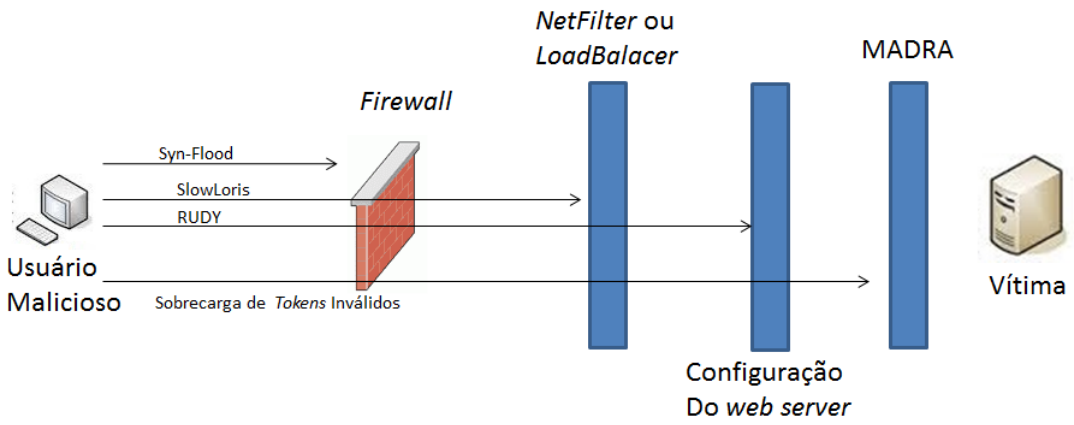

Figura 4.11 – Combinação de soluções para defesa de ataque de negação de serviço

O MADRA atua como uma camada de defesa, protegendo a aplicação contra os ataques que visam explorar o mecanismo de validação de *tokens* da vítima.

5. MITIGAÇÃO DE ATAQUES DOS EM RESTS AUTENTICÁVEIS (MADRA)

Este capítulo apresenta a solução para mitigar os ataques de negação de serviço que exploram a vulnerabilidade da REST API. Esta solução será chamada de MADRA, e tem por objetivo, através de diferentes estados de análise de informações, definir uma fila para controlar o acesso dos clientes ao sistema baseado no perfil gerado.

Conforme descrito no Capítulo 4, a sobrecarga de requisições levando ao esgotamento de recursos do sistema é chamada de ataque de negação de serviço. Através da taxonomia proposta por Mirkovic [Mir04], a Figura 5.1 apresenta como o MADRA seria classificado. O foco do presente trabalho está tratando de ataques de negação de serviço que operam via força bruta, com endereços IP de atacantes válidos tendo como alvo uma aplicação. Porém, se o ataque fosse efetuado utilizando IPs falsificados como origem, o MADRA também iria atuar corretamente protegendo a aplicação, pois não é feita nenhuma validação quanto a validade do IP. Entretanto, ataque em nível de aplicação requer que a conexão seja válida.

Já o meio de defesa é reativo com base na detecção de uma anomalia, sendo implantado no computador alvo com uma estratégia de respostas baseada na identificação dos agentes maliciosos. Em relação a automação do ataque, o MADRA está apto a tratar de ataques de negação de serviço rodando manualmente, automático ou mesmo semi-automático. Do mesmo modo, ele é capaz de lidar com os ataques cujo impacto atinja o sistema de modo a forçar uma parada total ou apenas degradem a performance do mesmo.

Quanto ao ritmo do ataque, o MADRA é capaz de lidar com ataques que tenham um ritmo constante ou até mesmo variável. Entretanto, ataques de ritmo variável podem escapar do sistema de defesa. Isso pelo fato de que no sistema proposto há definição de uma janela de operação, e essa janela define o tempo durante o qual as informações dos clientes são válidas. Uma vez que a janela se desloque e o cliente não faça nenhuma operação inválida durante este período, as informações em relação ao percentual de requisições inválidas poderá ser nulo.

Diante do problema apresentado e sua classificação, está sendo proposta uma implementação em nível de aplicação com intuito de minimizar o número de validações de *tokens* que são necessárias realizar no servidor, evitando que o mesmo execute chamadas desnecessárias ao banco de dados. A implementação parte da premissa que um cliente válido não irá tentar enviar constantemente *tokens* inválidos, desse modo, o sistema deve aceitar um conjunto de requisições (mesmo que estas possuam *token* inválido). Porém, ao suspeitar que esteja diante de um ataque de negação de serviço, que aqui analisamos pela combinação de *tokens* inválidos com o serviço da REST estar estressando a CPU, o

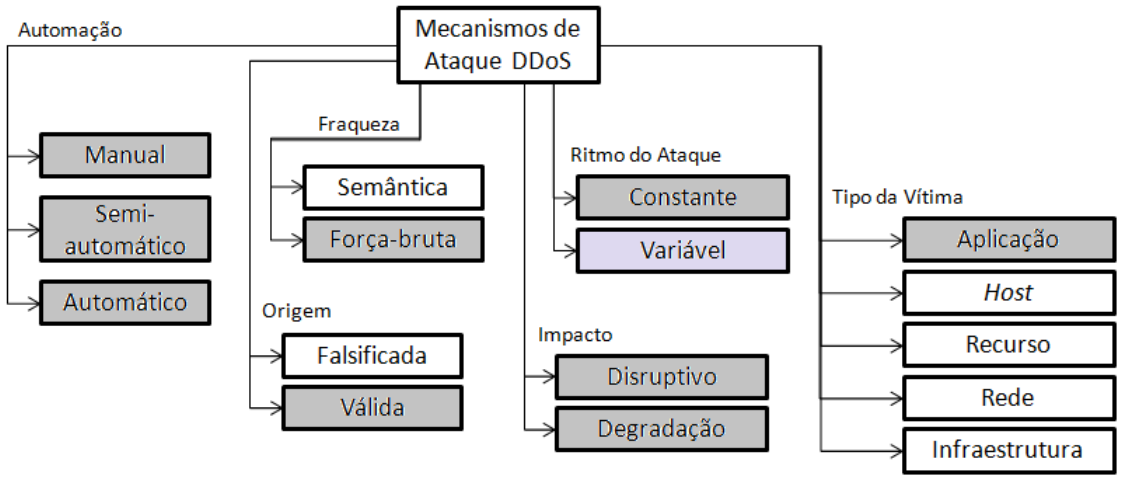

Figura 5.1 – Classificação do MADRA segundo Mirkovic (retângulos preenchidos)

sistema deve começar a descartar as requisições oriundas do cliente que está enviando as sucessivas requisições com *token* inválido.

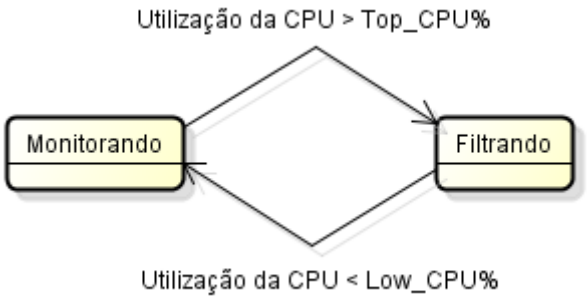

Figura 5.2 – Transição entre os estados de operação

A Figura 5.2 mostra os estados nos quais o sistema atua. O sistema MADRA é constituído por dois diferentes estados: Monitorando - aqui o sistema está apenas analisando o comportamento dos clientes para identificar quais são os que possam causar algum tipo de mal a aplicação e Filtrando - neste estado o sistema entra em ação de modo a descartar as requisiões originadas de clientes que foram identificados como potenciais suspeitos. O sistema, quando ativo, funciona monitorando as requisições dos clientes. A transição entre os estados se dá de acordo com a utilização da CPU do sistema, ou seja, ao detectar um uso de CPU superior ao percentual definido por *Top_CPU* o sistema passa do estado de monitoração para o estado de filtragem, e nesse estado deixa de atender as requisições dos clientes suspeitos.

O sistema se mantêm no estado de filtragem enquanto a utilização da CPU não baixar do valor percentual definido por *Low_CPU*. Essa diferença dos valores de transição entre os estados se dá para evitar que o sistema fique constantemente alternando entre os estados, e com isso bloqueando apenas poucas requisições vindas dos cliente suspeitos, desse modo ainda prejudicando o sistema.

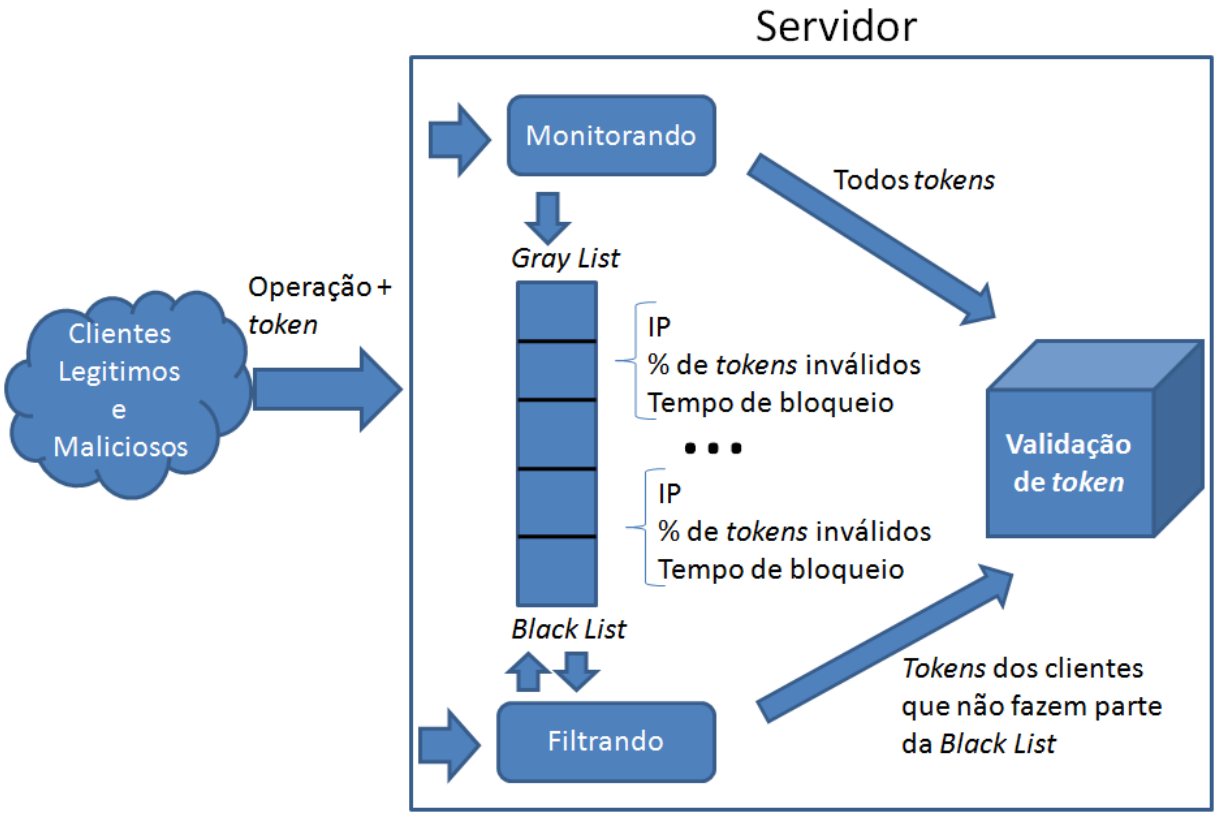

Figura 5.3 – Arquitetura do MADRA para controle de clientes

A Figura 5.3 apresenta a arquitetura da solução proposta. Seu princípio de funcionamento se dá analisando a utilização da CPU em conjunto com as requisições que são atendidas pelo serviço (na figura a validação de *tokens*). Uma vez que um cliente malicioso efetue chamadas com *tokens* inválidos, o sistema em modo de monitoramento irá criar um perfil deste cliente, contendo seu IP bem como o número percentual de *tokens* inválidos enviados no período definido por *Window*, o qual é definido em segundos. Note-se que durante esta fase de execução, as requisições são todas repassadas até o serviço que está sendo protegido. Esse perfil do cliente será armazenado em uma lista cinza (*gray list*).

Assim que o MADRA identificar que a CPU está com uma utilização superior a *Top_CPU*, o estado é alterado para filtragem. Agora a lista cinza se torna uma lista negra (*black list*), nesse momento todos os clientes armazenados na lista negra, que possuem um percentual de *tokens* inválidos superior a definição dada por *Token_Limit* passam a ter suas requisições bloqueadas. Esse bloqueio se dá pelo tempo definido também junto a cada cliente, no caso este tempo inicial é determinado por *Ban_Time* e é definido em segundos, pois é o mesmo tempo definido para calcular o percentual de *tokens* inválidos. Uma vez transcorrido o tempo, ao cliente é dada uma nova chance de enviar requisições, caso o cliente mantenha seu comportamento enviando *tokens* inválidos, este tempo de bloqueio vai sendo dobrado. Entretanto, se o cliente deixar de enviar *tokens* inválidos (percentual

de *tokens* inválidos inferior a *Token_Limit*), este tempo de bloqueio é reiniciado, e o mesmo pode voltar a operar normalmente.

5.1 Considerações

As definições dos valores de operação dos parâmetros *Top_CPU*, *Low_CPU*, *Window*, *Ban_Time* e *Token_Limit*, devem ser configurados de acordo com o ambiente onde o MADRA for instalado. Diferentes aspectos podem ser levados em consideração, por exemplo, médias de uso passado do ambiente, capacidade computacional do parque de equipamentos, horários (noturno, diurno), consumo de energia, dentre outros.

No Capítulo 6 iremos apresentar o estudo de caso que serviu como base para essas definições bem como a aplicação prática do sistema MADRA sendo executado e atuando de forma a mitigar um ataque de sobrecarga do componente Keystone do sistema de gerenciamento de nuvem OpenStack.

6. ESTUDO DE CASO - OPENSTACK

Para avaliar a solução proposta, um estudo de caso utilizando o sistema de gerenciamento de nuvem de código aberto foi usado. O OpenStack [Ope14] é um software de código aberto considerado um sistema operacional para ambientes de nuvem, responsável por controlar grande quantidade de computadores, equipamentos de armazenagem e redes em um *datacenter*. Ele provê um painel de controle para o administrador do sistema gerenciar o ambiente, e um portal web para que os usuários consigam gerenciar suas próprias máquinas virtuais [Ope14]. Ele é mantido pela OpenStack Foundation, que é uma organização sem fins lucrativos, e possui em sua comunidade de desenvolvimento parceria de empresas como Dell, EMC2, HP, IBM, Oracle, VMware, dentre outras.

O OpenStack é composto por diferentes módulos, cada um com diferentes responsabilidades. A Figura 6.1 apresenta os componentes que fazem parte do OpenStack versão Grizzly: OpenStack Dashboard (Horizon), OpenStack Object Store (Swift), OpenStack Image Service (Glance), OpenStack Compute (Nova), OpenStack Block Storage (Cinder), OpenStack Networking (Quantum) e OpenStack Identity Service (Keystone). Na versão atual, *i.e.* IcedHouse, também foram incluídos os componentes OpenStack Telemetria (Ceilometer), OpenStack Orchestration (Heat) e OpenStack Database (Trove).

As funções de cada um destes componentes são as seguintes:

• **OpenStack Dashboard** (Horizon) é responsável pela interface gráfica, onde o usuário irá gerenciar suas máquinas virtuais, e onde o administrador vai gerenciar o OpenStack;

• **OpenStack Object Store** (Swift) é responsável pela criação do mecanismo de armazenamento de objetos. O Swift cria um sistema de arquivos distribuído utilizado pelo OpenStack;

• **OpenStack Block Storage** (Cinder) é utilizado para criação de dispositivos de armazenamentos que serão anexados à máquina virtual;

• **OpenStack Image Service** (Glance) é responsável pela gerência de imagens para utilização nas máquinas virtuais. As imagens armazenadas no Glance são utilizadas com o conceito de *template*, que é a descrição padrão de um sistema operacional e a partir do qual podem ser criadas as máquinas virtuais;

• **OpenStack Identity Servic**e (Keystone) é responsável pela gerência dos usuários do OpenStack. Atua como um sistema de autenticação do OpenStack e ainda pode ser integrado a serviços como LDAP. Ainda, é responsável por listar um catálogo de serviços que o usuário pode utilizar;

• **OpenStack Networking** (Quantum) é o serviço de gerenciamento de redes entre as máquinas virtuais do OpenStack. Através dele é possível utilizar VLAN ou *flat networks* para garantir o isolamento do tráfego das diferentes máquinas virtuais. Ainda, faz a gerência

de IPs estáticos ou dinâmicos (DHCP). Para garantir a segurança destas redes o OpenStack conta com uma extensão que implementa um mecanismo de detecção de intrusão, balanceamento de carga, *firewalls* e redes privadas virtuais (VPN);

• **OpenStack Compute** (Nova) oferece o modelo de serviço IaaS. Através deste serviço é possível que os usuários consigam criar e implantar suas infraestruturas. O Nova é integrado com os outros módulos do OpenStack para que sejam criadas as máquinas virtuais.

De acordo com o apresentado na Figura 6.1, as setas que ligam os diferentes componentes representam a direção da chamada executada para os componentes; as setas tracejadas representam as chamadas internas efetuadas entre os próprios componentes; já as setas com linhas contínuas representam as APIs expostas dos componentes para acesso externo do sistema do OpenStack. Essa comunicação acontece após cada componente implementar uma API REST, onde são expostas as operações aos outros módulos. A utilização de uma REST para comunicação entre os módulos do OpenStack permite que o sistema seja modular e possibilita que os diferentes módulos sejam executados em diferentes computadores. Analisando essas comunicações através das APIs podemos notar que o componente Keystone acaba por processar requisições de todos os outros componentes, tornando-se assim um ponto único que talvez possa apresentar alguma falha, e com isso uma falha nele possa comprometer todo o funcionamento do OpenStack.

Além dos mecanismos de comunicação, a figura também mostra os sistemas principais que compõem cada módulo do OpenStack.

6.1 Segurança no OpenStack

Para proteger os módulos do OpenStack durante sua instalação, o *firewall* do sistema operacional é configurado de modo a bloquear qualquer tráfego externo que tente acessar as REST APIs do OpenStack. Porém, em algumas situações é necessário que um sistema externo tenha acesso a essa API [LZB+13], por exemplo, quando um cliente do serviço de nuvem quer utilizar seu próprio sistema para gerenciar suas máquinas virtuais hospedadas na nuvem. Para que essa integração seja bem sucedida é necessário que as REST APIs dos componentes do OpenStack sejam expostas. No caso de gerenciamento de máquinas virtuais, a REST API do componente Nova deveria ser exposta, através de regras no *firewall* do sistema que permitam acesso externo.

De acordo com a *Common Vulnerabilities and Exposures* (CVE) [CVE13], em 2013 haviam 63 problemas de segurança relacionados ao OpenStack; já em 2014 esse número de vulnerabilidades aumentou para 113. Destas vulnerabilidades, 33 são relacionadas diretamente com problemas no componente do Keystone e 23 associadas ao componente Nova. Estes dois componentes do OpenStack são os principais módulos devido a suas

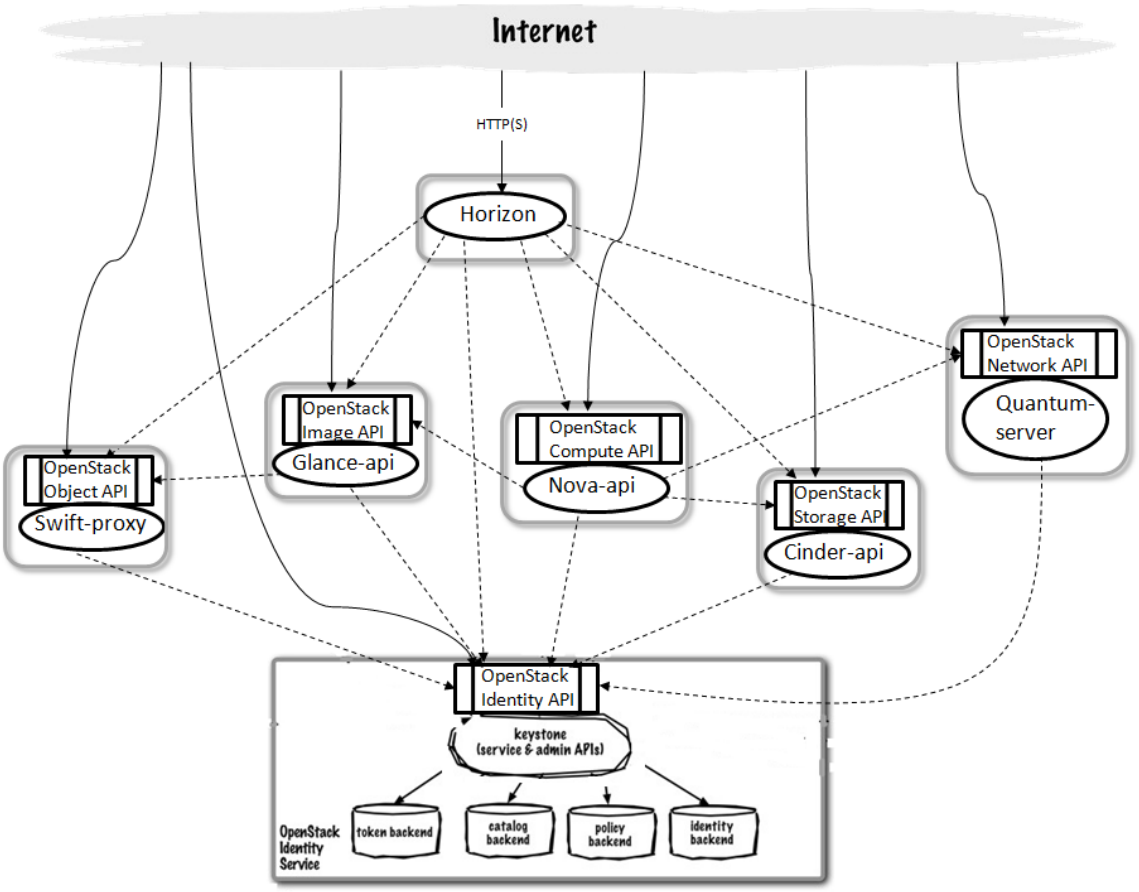

Figura 6.1 – Adaptação da arquitetura completa do OpenStack [Ope14]

atribuições no sistema. Além das vulnerabilidades apontadas pela CVE, a comunidade de desenvolvimento do OpenStack, através do *site* Launchpad [OSS13], também conta com uma lista de problemas relacionados a segurança do OpenStack. Dentre estes problemas, um deles é conhecido e relacionado à biblioteca de virtualização LibVirt [Lib15]. Outros dois problemas são relacionados ao componente Keystone. Dentre estes dois problemas encontrados no Keystone, um deles é relacionado a ataques de negação de serviço, pois o mesmo mantêm em memória as informações recebidas através das requisições dos clientes. Ao receber mensagens de tamanho considerável, levariam a exaurir os recursos disponíveis, fazendo com que o componente pare de responder. O outro problema é relacionado com o armazenamento inseguro de credenciais, as informações de usuário e senha do sistema ficam armazenadas em formato de texto dentro do arquivo de configuração do Keystone, assim qualquer pessoa que tenha acesso ao arquivo consegue ter as credenciais diretamente.

Podemos notar que o componente Keystone acaba por ser afetado pelas vulnerabilidades de negação de serviço (Seção 2.5) e usuário mal intencionado (Seção 2.6), conforme analisados no Capítulo 2. Já o problema relacionado com a biblioteca de virtualização LibVirt impacta diretamente o componente Nova, pois através desta falha diferentes máquinas virtuais compartilhando recursos conseguiriam interferir uma nas outras. Neste

caso o componente Nova é afetado pela vulnerabilidade de recursos compartilhados (Seção 2.9).

Sabendo das funcionalidades apresentadas pelo OpenStack e seus componentes, podemos fazer uma relação de como cada uma das nove principais ameaças de ambientes de nuvem podem afetar o OpenStack:

1. **Vazamento de dados** (Seção 2.1): Os componentes Swift e Cinder seriam os principais alvos desse tipo de ameaça, pois são eles que armazenam os dados de todas as máquinas virtuais criadas no ambiente de nuvem, e uma vez comprometidos estes componentes o dano seria grande. Além deles, o componente Quantum também pode ser alvo, uma vez que cria as redes por onde irão trafegar as informações dos clientes.

2. **Perda de dados** (Seção 2.2): Assim como a ameaça de vazamento de dados, os componentes do Swift e Cinder são os grandes alvos dessa ameaça.

3. **Sequestro de contas** (Seção 2.3): Por gerenciar todo sistema de identidade do OpenStack, o Keystone seria o principal alvo desse tipo de ameaça.

4. **Interfaces inseguras** (Seção 2.4): O componente Horizon acaba por ser o principal alvo do OpenStack, por ser através dele que todas as requisições são feitas para o ambiente. Porém, deve-se ter atenção especial a qualquer outro componente que tenha sua API exposta.

5. **Negação de serviços** (Seção 2.5): Todos os componentes que possuam uma REST API exposta ou algum outro meio de acesso está sujeito a um ataque de negação de serviço. O componente Quantum é fundamental para garantir o isolamento de tráfego entre as diferentes máquinas virtuais que rodam no ambiente de nuvem, pois caso seja comprometido, o ataque pode ser efetuado dentro do ambiente. Por outro lado, ao expor a REST do componente Keystone, este componente se transforma no principal alvo, visto que comprometendo ele, todos os demais módulos deixariam de conseguir validar os *tokens*.

6. **Usuário mal intencionado** (Seção 2.6): Um usuário mal intencionado pode comprometer completamente o sistema, por normalmente ter acesso ao ambiente onde se encontra a nuvem. No caso do OpenStack os principais componentes alvos de um usuário mal intencionado seriam o Keystone visando comprometer os dados de autenticação dos clientes da nuvem; o Nova para causar algum dano às máquinas virtuais já existentes; o Quantum com a finalidade de capturar dados em trânsito dos clientes; e também o Cinder tentando buscar as informações armazenadas em cada uma das máquinas virtuais.

7. **Abuso dos serviços** (Seção2.7): Dentre todos os componentes, o principal alvo deste tipo de ataque pode ser considerado o Quantum, por gerenciar as redes criadas para ligação das máquinas virtuais e em caso de uma falha, a rede da nuvem pode ser utilizada para comprometer outros serviços. O componente Ceilometer introduzido a partir da versão Havana do OpenStack pode ajudar a coibir esse tipo de ameça, visto que é

responsável por monitorar a utilização dos serviços de nuvem, e assim evitar um abuso dos mesmos.

8. **Investigação insuficiente** (Seção 2.8): Independe do sistema de nuvem, pois é uma vulnerabilidade relacionada aos seus usuários.

9. **Recursos compartilhados** (Seção 2.9): Pode afetar os componentes Glance, Nova, Keystone, Cinder, Swift e Quantum, pois estes manipulam informações de todos os clientes da nuvem.

Baseados nessa análise vemos que nem todas as vulnerabilidades estão ligadas com o OpenStack, como é o caso da investigação insuficiente, pois o cliente é o principal responsável por evitar esse tipo de ameaça. Por outro lado, todas as outras ameaças afetam diretamente os componentes do OpenStack. Dentre todos os componentes, o Keystone é um componente chave de todo o sistema, e também pelas ameaças de recursos compartilhados, usuário mal intencionado, sequestro de contas e negação de serviço merecendo assim atenção especial para garantir sua segurança.

6.2 Arquitetura com MADRA

A Figura 6.2 mostra uma simplificação da arquitetura do OpenStack com a aplicação do MADRA sendo utilizado para atuar na análise das requisições externas. Nela é mostrada a interconexão entre os componentes bem como a exposição da REST API do Keystone. Desse modo, pode-se ver que na execução usual do sistema, este componente poderá receber uma grande quantidade de requisições. Assim, para que cada operação seja executada nos outros módulos, o Keystone deve ser consultado para indicar se o *token* do usuário possui permissão para execução da operação.

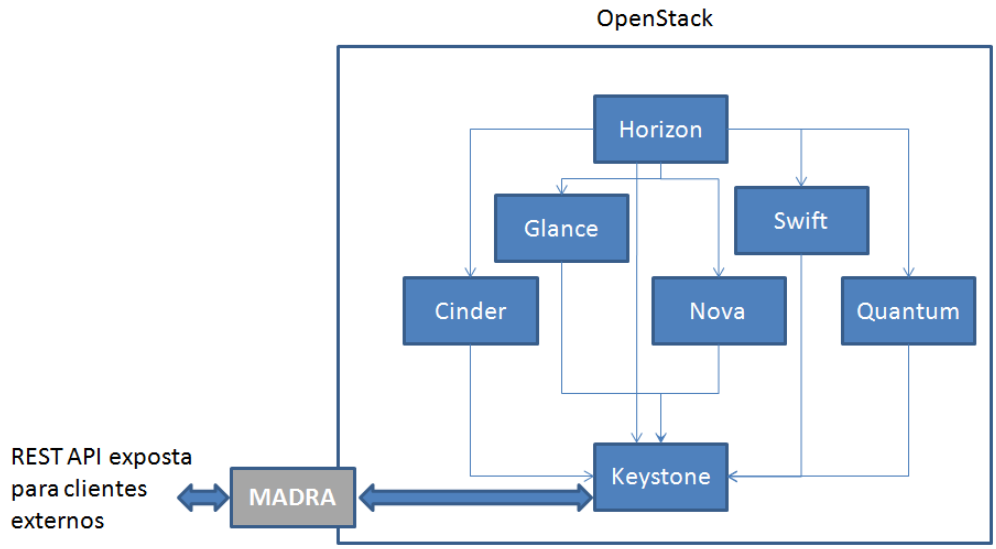

Figura 6.2 – Arquitetura simplificado do OpenStack utilizando MADRA

Por ser um componente que recebe requisições de todos os outros compoenetes, o Keystone acaba por se tornar um alvo atrativo para usuários maliciosos. Uma vez que este usuário tenha acesso a REST disponibilizada pelo Keystone, ele pode sobrecarregá-la enviando uma série de chamadas com *tokens* inválidos.

Para evitar que qualquer cliente tenha capacidade de sobrecarregar o componente Keystone, o OpenStack, durante sua instalação, configura o *firewall* do Linux para que não exponha a REST do Keystone. Desse modo, apenas usuários da rede interna do sistema tem acesso a esta REST. Caso o administrador do sistema precise expor esta REST para integrar com algum outro sistema, ele também pode configurar o *firewall* de modo a permitir acesso de sistemas que estejam em outras redes. Nesta análise será assumido que os usuários gerando tráfego malicioso encontram-se em outra rede ou mesmo na Internet, e a REST do Keystone foi exposta pelo administrador.

Com intuito de identificar o tempo de reposta das operações disponibilizadas pelo Keystone, executou-se uma sobrecarga de duas operações que são: geração de um *token* válido, onde era informado um usuário e senha válidos de modo que o sistema criasse um *token* válido e retorná-lo; e validação de um *token* tanto válido quanto inválido do maior tamanho aceito pela operação da REST. Para medir este tempo de resposta, durante o ataque ao componente, um cliente legítimo envia uma requisição para geração de um *token* válido. Foi percebido que o tempo de validação de um *token* inválido aumenta em uma velocidade maior que da geração de *tokens* muito similar ao comportamento da validação de um *token* válido. Entretanto, nesse caso o sistema pode simplesmente revogar o *token* que está sendo utilizado de maneira indevida. Desse modo, o presente trabalho se foca em analisar e mitigar o problema da validação de *tokens* inválidos.

Para executar os testes deste trabalho, toda arquitetura foi montada sob a tecnologia de máquinas virtuais, devido a sua grande importância no cenário do desenvolvimento e popularização de ambientes de nuvem.

O sistema de virtualização utilizado foi VMware workstation 10.0, por ser um dos mais conhecidos softwares de virtualização comercial. Este sistema foi executado em um computador Intel Core i5-4570 com clock de 3.20 GHz, 16 GB de memória RAM, barramento DDR3 1600, rodando Microsoft Windows 7 Professional 64 bits. A máquina virtual responsável por executar o ambiente do OpenStack versão Havana possui 4 GB de memória RAM, 2 processadores, rodando Linux Ubuntu Server 14.04. As máquinas virtuais clientes responsáveis por executar um *script* escrito em Python responsável por gerar o volume de requisições possui 2 GB de RAM, 1 processador rodando Linux Ubuntu Server 14.04.

A Figura 6.3 apresenta a estrutura do sistema do estudo de caso. Nela é mostrado o sistema de virtualização onde estão sendo executados os clientes maliciosos bem como o servidor com o OpenStack e o MADRA. Aqui foram criados 6 clientes maliciosos em diferentes máquinas virtuais. Em cada um deles é executado um *script* escrito em Python

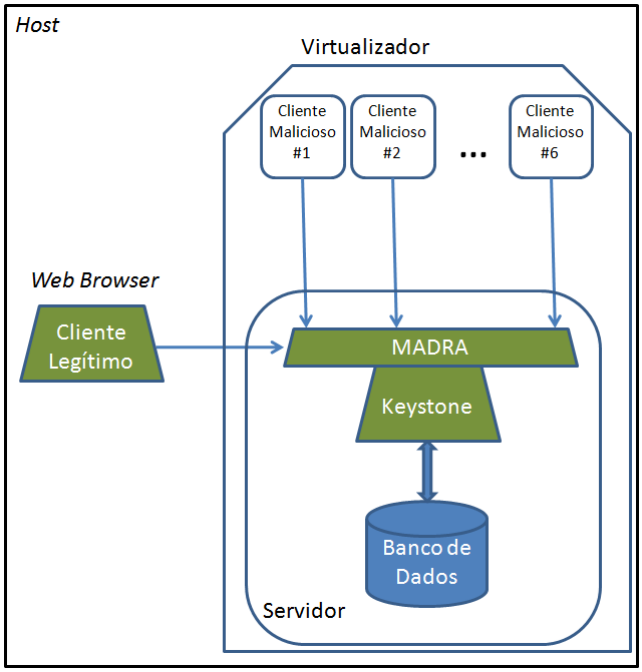

Figura 6.3 – Cenário do estudo de caso

responsável por gerar a carga de requisições por segundo que varia de 10 mil, 20 mil e 30 mil. Assim, durante o ataque inicialmente temos o servidor atendendo a 60 mil requisições por segundo, a seguir esse volume aumenta para 120 mil requisições por segundo e finalmente o ataque chega a 180 mil requisições por segundo.

Durante este ataque o *browser,* rodando no computador *host* que está executando o sistema de virtualização, é responsável por atuar como o cliente legítimo. Através dele são feitas as requisições de geração de *token* com usuário e senha válidos bem como operações com *token* válido.

Na máquina virtual do servidor está sendo executado o OpenStack, seu banco de dados, bem como o MADRA. Na instalação do OpenStack, foi alterada a configuração do *firewall* de modo a expor a REST API do componente Keystone para permitir acesso de clientes externos. O MADRA foi também devidamente configurado de modo a analisar as requisições feitas ao componente Keystone.

6.3 Configuração do ambiente

A definição dos valores de utilização de CPU se deram com base no estudo de caso apresentado. Conforme gráfico da análise da CPU da aplicação (Figura 6.5) durante um ataque, a mesma começa deteriorar o tempo de resposta quando o uso da CPU excede os 70%. Por outro lado, vemos que um uso de CPU de 50% estaria totalmente adequado para atender uma demanda de aproximadamente 50.000 requisições sem que haja perda

da qualidade do tempo de resposta. Desse modo, foi definido o valor de *Top_CPU* como 70% bem como *Low_CPU* em 50%.

A definição do valor da janela de tempo (*Window*) foi de 10 segundos e se deu em função da análise do tempo de resposta do sistema ocioso. No estudo de caso feito, o pior tempo de resposta obtido com o sistema sendo executado em condições normais foi de aproximadamente 200 milissegundos. Desse modo, *Window* foi definido com valor de 10 segundos que corresponde a 50 vezes do pior caso de execução do sistema em execução regular.

Para o valor que determina o tempo no qual o cliente fica impossibilitado de enviar novas requisições, no caso *Ban_Time*, foi definido o valor de 10 segundos. Desse modo, o cliente fica sem poder enviar nenhuma requisição pelo período de tempo igual à janela. Já o limite de *tokens* inválidos (*Token_Limit*) foi definido como 70%.

Durante a execução do sistema por usuário legítimos um *token* gerado corretamente pelo OpenStack possui tamanho de aproximadamente 2,5 Kb. No intuito de aumentar a carga gerada no OpenStack, os clientes maliciosos deste estudo de caso, geraram *tokens* do tamanho máximo suportado pela REST do Keystone, que foi de aproximadamente 16 Kb.

6.4 Análise de resultados

O gráfico da Figura 6.4 apresenta a execução do sistema OpenStack, no qual apenas a configuração da exposição da REST API do Keystone foi efetuada. Sua análise serviu de base para identificação do problema do aumento do tempo de resposta durante a sobrecarga do sistema diante do ataque de negação de serviço. Assim, o ambiente foi configurado de forma a gerar um determinado número de requisições de validação de *tokens* válidos e inválidos. Durante a execução dessa validação é feita uma requisição de geração de *token* e seu tempo é medido. Além disto, de forma a comparar o custo de validação com o de geração de *tokens*, é feita também a medição de tempo de geração de um *token* quando o sistema está sendo estressado com diversas requisições de geração de *tokens*.

A Figura 6.4 apresenta o gráfico com o tempo de resposta em milissegundos da operação de geração de um *token*, quando o sistema está atendendo, através de chamadas para a REST do Keystone, a: i) requisições de validação de *tokens* com *tokens* inválidos; ii) requisições de validação de *tokens* com *tokens* válidos; e, iii) requisições de geração de *tokens*. Assim, na situação i), o tempo médio de resposta, para uma chamada de geração de *token* de um cliente legítimo, é de cerca de 898 milissegundos (margem de erro de 10% [1]). Por outro lado, para a situação ii), o tempo de resposta fica em torno de 680

[1] Margem de erro calculada através do índice de confiança, utilizando o cálculo de desvio padrão e 10 amostras coletadas.

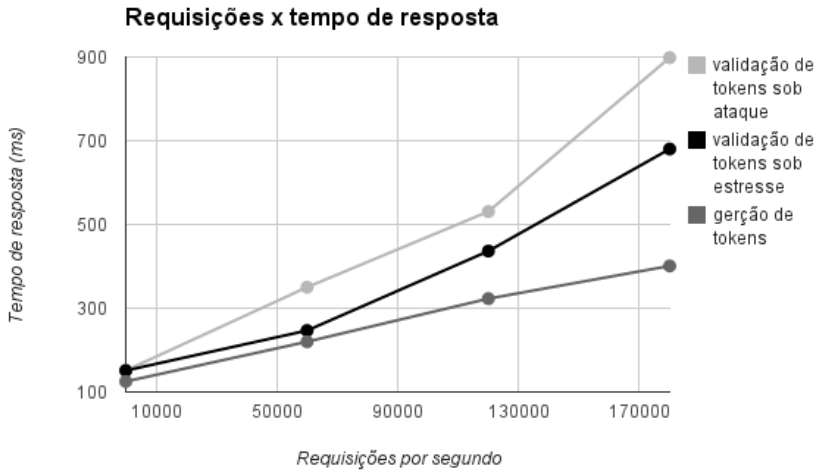

Figura 6.4 – Análise do aumento do tempo de resposta.

milissegundos (margem de erro de 12%). Podemos identificar que o tempo de resposta durante um ataque piora em cerca de 32%. Para comparação, o tempo de resposta para geração de um *token* na situação iii) é de 425 milissegundos (margem de erro de 16%). Por outro lado, este tempo durante a execução, onde o sistema está ocioso, foi de 151 milissegundos para gerar um *token* (margem de erro de 11%).

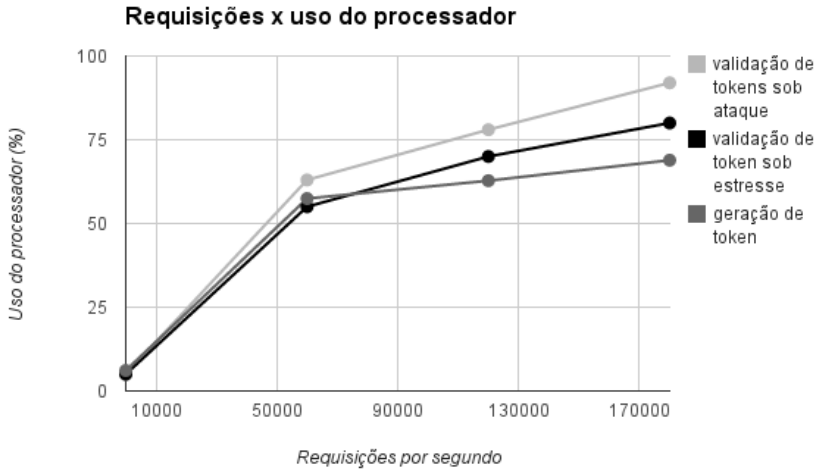

Figura 6.5 – Uso do processador durante um ataque

Além do tempo de resposta crescente, nota-se que o processador no qual a aplicação era executada esteve com a utilização próxima do seu limite. Na Figura 6.5 é mostrada a utilização do processador durante um ataque ou sob estresse. Na situação sob ataque, o uso do processador, com 180.000 requisições, é de cerca de 92%. Para a situação sob estresse, com 180.000 requisições, o uso do processador é de cerca de 80%. Estes números indicam um aumento de 15% no consumo do processador em uma situação de ataque,

comparado com uma situação de estresse. Novamente, para comparação, durante a geração de 180.000 *tokens* o consumo do processador é de cerca de 68%, mostrando que a validação é cerca de 17% mais custosa em termos de uso do processador em uma situação de estresse normal.

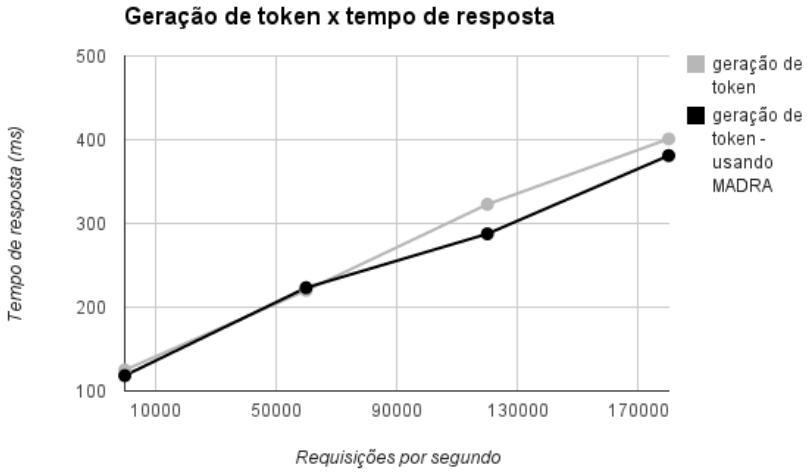

Figura 6.6 – Tempo de resposta para geração de *tokens*

Através dos dados de utilização de processador apresentados na Figura 6.5, foi possível definir os valores onde o ataque começa a degradar a performance do sistema. Na figura podemos ver que o sistema opera com uma carga de 50.000 requisições por segundo mantendo sua utilização de processador próxima a 50%. A partir de 60.000 requisições por segundo, a utilização do processador supera os 60% de uso, consequentemente, elevando o tempo de resposta para os clientes da aplicação. Ainda podemos identificar que a operação de geração de *token*, mesmo quando atingido o nível de 180.000 requisições por segundo, não superou o uso de 70% do processador.

A Figura 6.6 nos apresenta o tempo de resposta da operação de geração de um *token* válido durante uma sobrecarga dessa operação utilizando o MADRA. Nela podemos notar que o tempo de reposta variou de 425 milissegundos para 380 milissegundos, com margem de erro de 15% para a amostra coletada. Isso se dá pelo MADRA atuar apenas no tratamento de chamadas de operações que requerem a validação do *token*. Assim, podemos concluir que o MADRA não afeta a operação de geração de *tokens*.

Na Figura 6.7 é apresentada a diferença de tempo entre a execução do módulo Keystone do OpenStack utilizando a solução do MADRA e sem a mesma. Executando novamente a carga de 180.000 requisições por segundo de validação de *tokens*, durante uma situação de ataque ao Keystone, o tempo de resposta foi de 328 milissegundos (margem de erro de 10%), representando um tempo de resposta aproximadamente 36% mais rápido que o mesmo cenário sem a solução.

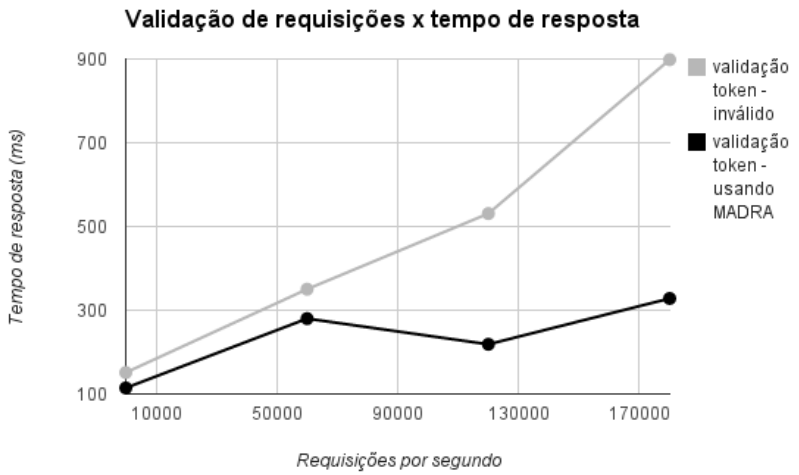

Figura 6.7 – Tempo de resposta do sistema em diferentes cenários

Durante os testes foram também medidos os tempos que o Keystone leva para efetuar a validação de um *token* válido e de um *token* inválido quando o sistema está ocioso. Para validar um *token* válido o Keystone leva 29 milissegundos, já para a validação de um *token* inválido este tempo foi de 144 milissegundos. Essa mesma análise efetuada com o MADRA sendo executado fez com que os valores passassem para 34 milissegundos e 156 milissegundos respectivamente.

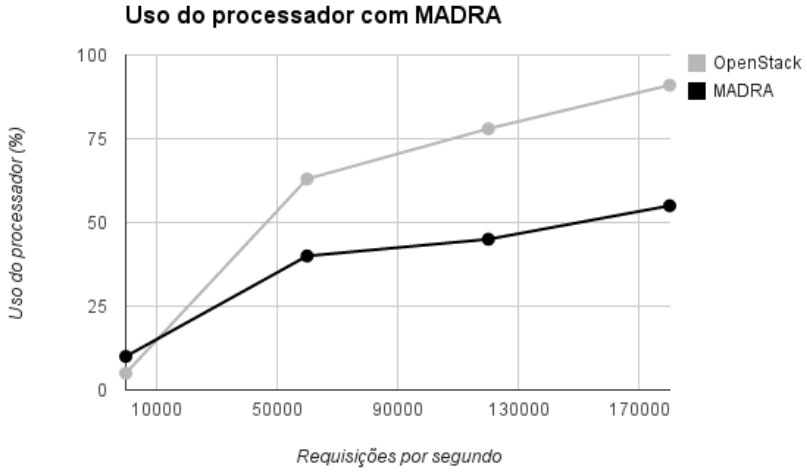

Figura 6.8 – Uso do processador no OpenStack orginal e com MADRA rodando

A Figura 6.7 mostra que o sistema inicia tendo tempos de resposta muito próximos até as 60.000 requisições por segundo, que é quando o processador atinge o nível de utilização que faz o MADRA transicionar do estado de monitoração para o estado de filtragem. A partir desse momento, descartando os pacotes que são gerados apenas com o intuito

de sobrecarregar o sistema, a aplicação mantem um tempo de resposta para os clientes abaixo de 350 milissegundos.

Notamos também que a utilização do processador diminui quando utilizando o MADRA no controle de clientes maliciosos. Na Figura 6.8, o sistema sob ataque e com o MADRA rodando, é mostrado que o mesmo consegue manter a utilização do processador em aproximadamente 55%, ao passo que sem a solução, diante de um ataque que gere 180.000 requisições por segundo, sua utilização fica próxima dos 90%.

7. CONCLUSÃO

A solução apresentada é específica para ser aplicada em sistemas que utilizem REST autenticáveis através de *tokens*. No caso de um ataque de negação de serviço utilizando algum tipo de falha da pilha TCP/IP ou mesmo uma inundação de pacotes, a proposta não seria viável. Nesse caso uma solução como a proposta por Tariq [TMA11] em seu trabalho poderia resolver o problema. No entanto, a proposta apresentada nesse trabalho poderia ser utilizada em conjunto, de modo a efetuar a defesa do sistema em nível de aplicação, conforme descrito na Seção 4.3.

Com base na Figura 6.7, que nos apresenta o tempo de resposta da aplicação com a solução proposta sendo utilizada, notamos que esse tempo fica muito próximo a utilização do sistema em condições normais de operação. Desse modo, mesmo diante de um ataque, o sistema teria um comportamento com tempo de resposta dentro do esperado por seus clientes. Além do tempo de resposta melhorado, também temos uma redução no uso do processador durante o ataque, conforme apresentado na Figura 6.8.

Uma importante contribuição do trabalho foi trazer a discussão sobre ataques de negação de serviço que são focados em determinadas aplicações, os quais não fazem uso de nenhuma ferramenta conhecida de ataque, e desse modo não são detectados por mecanismos de defesa tradicionalmente utilizados. Esse tipo de situação é possível graças a análise do código fonte da aplicação, onde qualquer usuário pode identificar as operações que mais consumam recursos, e dessa forma criar um *script* que execute a sobrecarga da mesma.

Na continuidade do presente trabalho, poderia ser explorado um mecanismo de configuração automático da janela de tempo de análise dos dados, definir essa informação através de um mecanismo para traçar um perfil do tempo de execução de cada chamada, similar ao apresentado por Barna [BSS+14] em seu trabalho. Além disso, também poderia ser explorado um mecanismo de análise da carga do processador, para também de maneira automática, definir a transição entre os estados. Outro aspecto que poderia ser explorado é relacionado a análise semântica das fraquezas, de modo a incorporar no sistema a análise de assinaturas de ataques conhecidos tais como os mencionados na Seção 4.2.

REFERÊNCIAS BIBLIOGRÁFICAS

[ADM05] Al-Duwairi, B.; Manimaran, G. “Victim-assisted mitigation technique for tcp-based reflector ddos attacks”. In: Proceedings of the 4th IFIP-TC6 International Conference on Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communication Systems, 2005, pp. 191–204.

[All13] Alliance, C. S. “The notoriuous nine cloud computing top threats in 2013”. Acessado em 12/04/2013, Capturado em: https://downloads.cloudsecurityalliance.org/initiatives/top_threats/The_Notorious_Nine_Cloud_Computing_Top_Threats_in_2013.pdf, 2013.

[Bak13] Baker, L. B.; Finkle, J. “Sony playstation suffers massive data breach”. Acessado em 22/06/2013, Capturado em: http://www.reuters.com/article/2011/04/26/us-sony-stoldendata-idUSTRE73P6WB20110426, 2013.

[Bei12a] Beitollahi, H.; Deconinck, G. “Analyzing well-known countermeasures against distributed denial of service attacks”, *Computer Communications*, vol. 35–11, 2012, pp. 1312 – 1332.

[Bei12b] Beitollahi, H.; Deconinck, G. “Tackling application-layer DDoS attacks”, *Procedia Computer Science*, vol. 10–0, 2012, pp. 432 – 441.

[BFZ07] Ballani, H.; Francis, P.; Zhang, X. “A study of prefix hijacking and interception in the internet”, *SIGCOMM Comput. Commun. Rev.*, vol. 37–4, Ago 2007, pp. 265–276.

[Bru02] Brustoloni, J. “Protecting electronic commerce from distributed denial-of-service attacks”. In: Proceedings of the 11th International Conference on World Wide Web, 2002, pp. 553–561.

[BSS+14] Barna, C.; Shtern, M.; Smit, M.; Tzerpos, V.; Litoiu, M. “Mitigating DoS attacks using performance model-driven adaptive algorithms”, *ACM Trans. Auton. Adapt. Syst.*, vol. 9–1, Mar 2014, pp. 3:1–3:26.

[CER13] CERT. “The CERT Insider Threat Center”. Acessado em 15/09/2013, Capturado em: http://www.cert.org/insider_threat/, 2013.

[Che10] Chen, X. “Distributed denial of service attack and defense”. In: Proceedings of the 2010 International Conference on Educational and Information Technology (ICEIT), 2010, pp. V3–318–V3–320.

[CI12] Cardellini, V.; Iannucci, S. “Designing a flexible and modular architecture for a private cloud: a case study”. In: Proceedings of the 6th international workshop on Virtualization Technologies in Distributed Computing Date, 2012, pp. 37–44.

[Clo14] CloudStack, A. “Apache cloudstack: Open source cloud computing”. Acessado em 18/06/2014, Capturado em: http://cloudstack.apache.org/, 2014.

[Cou13] Council, P. S. S. “PCI SSC data security standards overview”. Acessado em 24/05/2013, Capturado em: https://www.pcisecuritystandards.org/security_standards/, 2013.

[Cra09] Craig, J. S. “The human element: training, awareness, and human resources implications of health information security policy under the health insurance portability and accountability act (hipaa)”. In: Proceedings of the 2009 Information Security Curriculum Development Conference, 2009, pp. 95–99.

[Cre12] Cretella, G.; Di Martino, B. “Semantic web annotation and representation of cloud apis”. In: Proceedings of the 2012 Third International Conference on Emerging Intelligent Data and Web Technologies (EIDWT), 2012, pp. 31–37.

[CVE13] CVE. “Common vulnerabilities and exposures”. Acessado em 15/09/2013, Capturado em: http://cve.mitre.org/, 2013.

[DCG12] Duncan, A.; Creese, S.; Goldsmith, M. “Insider attacks in cloud computing”. In: Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2012, pp. 857–862.

[DDL+12] Damon, E.; Dale, J.; Laron, E.; Mache, J.; Land, N.; Weiss, R. “Hands-on denial of service lab exercises using slowloris and rudy”. In: Proceedings of the 2012 Information Security Curriculum Development Conference, 2012, pp. 21–29.

[Dig13] Dignan, L. “Sony’s data breach costs likely to scream higher”. Acessado em 22/06/2013, Capturado em: http://www.zdnet.com/blog/btl/sonys-data-breach-costs-likely-to-scream-higher/49161, 2013.

[DTM10] Dawoud, W.; Takouna, I.; Meinel, C. “Infrastructure as a service security: Challenges and solutions”. In: Proceedings of the 7th International Conference on Informatics and Systems (INFOS), 2010, pp. 1–8.

[Esp13] Espiner, T. “Wi-fi hack caused tj maxx security breach”. Acessado em 22/06/2013, Capturado em: http://www.zdnet.com/wi-fi-hack-caused-tk-maxx-security-breach-3039286991/, 2013.

[Euc14] Eucalyptus. “Eucalyptus open source private cloud softwaree”. Acessado em 18/06/2014, Capturado em: https://www.eucalyptus.com/, 2014.

[Fee14] Feedly. “Denial of service attack [neutralized] | building feedly”. Acessado em 18/06/2014, Capturado em: http://blog.feedly.com/2014/06/11/denial-of-service-attack/, 2014.

[Fie00] Fielding, R. T. “Fielding dissertation: Chapter 5: Representational state transfer (rest)”. Acessado em 18/06/2014, Capturado em: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000.

[Goo13] Goodin, D. “Zeus bot found using amazon’s EC2 as c&c server”. Acessado em 10/08/2013, Capturado em: http://www.theregister.co.uk/2009/12/09/amazon_ec2_bot_control_channel/, 2013.

[GSL13] Gracia, R.; Sanchez, M.; Lopez, P. “Cloud-as-a-Gift: Effectively exploiting personal cloud free accounts via REST APIs”. In: Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD), 2013, pp. 621–628.

[Hal07] Hall, J. A.; Liedtka, S. L. “The sarbanes-oxley act: implications for large-scale it outsourcing”, *Commun. ACM*, vol. 50–3, Mar 2007, pp. 95–100.

[HHL03] Huang, Y.-W.; Huang, S.-K.; Lin, T.-P. “Web application security assessment by fault injection and behavior monitoring”. In: Proceedings of the 12th International Conference on World Wide Web, 2003, pp. 148–159.

[Hon13] Honan, M. “Kill the password: Why a string of characters cant protect us anymore”. Acessado em 24/06/2013, Capturado em: http://www.wired.com/gadgetlab/2012/08/apple-amazon-mat-honan-hacking/, 2013.

[Kle14] Kleyman, B. “Understanding cloud apis, and why they matter”. Acessado em 15/10/2014, Capturado em: http://www.datacenterknowledge.com/archives/2012/10/16/understanding-cloud-integration-a-look-at-apis/, 2014.

[KMW01] Kargl, F.; Maier, J.; Weber, M. “Protecting web servers from distributed denial of service attacks”. In: Proceedings of the 10th International Conference on World Wide Web, 2001, pp. 514–524.

[KPR09] Kandukuri, B.; Paturi, V.; Rakshit, A. “Cloud security issues”. In: Proceedings of IEEE International Conference on Services Computing, 2009. SCC 09, 2009, pp. 517–520.

[KVM14] KVM. “Kernel based virtual machine”. Acessado em 18/06/2014, Capturado em: http://www.linux-kvm.org/page/Main_Page, 2014.

[Lem13] Lemos, R. “Cloud-based denial of service attacks looming, researchers say”. Acessado em 15/09/2013, Capturado em: http://www.darkreading.com/perimeter/cloud-based-denial-of-service-attacks-lo/226500300, 2013.

[Lib15] Libvirt. “The virtualization api”. Acessado em 02/01/2015, Capturado em: http://libvirt.org/, 2015.

[Lit07] Litoiu, M. “A performance analysis method for autonomic computing systems”, *ACM Trans. Auton. Adapt. Syst.*, vol. 2–1, Mar 2007.

[Liu10a] Liu, H. “A new form of dos attack in a cloud and its avoidance mechanism”. In: Proceedings of the 2010 ACM workshop on Cloud computing security workshop, 2010, pp. 65–76.

[Liu10b] Liu, S.; Kuhn, R. “Data loss prevention”, *IT Professional*, vol. 12–2, 2010, pp. 10–13.

[LZB+13] Lu, Q.; Zhu, L.; Bass, L.; Xu, X.; Li, Z.; Wada, H. “Cloud api issues: an empirical study and impact”. In: Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures, 2013, pp. 23–32.

[Mel13] Mell, P.; Grance, T. “The NIST definition of cloud computing”. Acessado em 12/09/2013, Capturado em: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2013.

[Men13] Meng, D. “Data security in cloud computing”. In: Proceedings of the 2013 8th International Conference on Computer Science Education (ICCSE), 2013, pp. 810–813.

[MGK13] McCallister, E.; Grance, T.; Karen, S. “Guide to protecting the confidentiality of personally identifiable information(PII)”. Acessado em 04/07/2013, Capturado em: http://csrc.nist.gov/publications/nistpubs/800-122/sp800-122.pdf, 2013.

[Mic14] Microsoft. “Virtualization for your modern datacenter and hybrid cloud”. Acessado em 18/06/2014, Capturado em: http://www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspx, 2014.

[Mil13] Miller, M. H. “Data theft: Top 5 most expensive data breaches”. Acessado em 22/06/2013, Capturado em: http://www.csmonitor.com/Business/2011/0504/Data-theft-Top-5-most-expensive-data-breaches/3.-TJX-256-million-or-more, 2013.

[Mir04] Mirkovic, J.; Reiher, P. “A taxonomy of DDoS attack and DDoS defense mechanisms”, *SIGCOMM Comput. Commun. Rev.*, vol. 34–2, Abr 2004, pp. 39–53.

[Mou13] Moustis, D.; Kotzanikolaou, P. “Evaluating security controls against http-based ddos attacks”. In: Proceedings of the 2013 Fourth International Conference on Information, Intelligence, Systems and Applications (IISA), 2013, pp. 1–6.

[ONe13] ONeil, M. “Cloud APIs - the Next Battleground for Denial-of-Service Attacks”. Acessado em 15/09/2013, Capturado em: https://blog.cloudsecurityalliance.org/2013/04/13/cloud-apis-the-next-battleground-for-denial-of-service-attacks/, 2013.

[Ope14] OpenStack. “Openstack open source cloud computing software”. Acessado em 22/06/2014, Capturado em: https://www.openstack.org/, 2014.

[OSS13] OSSG. “Openstack security group team”. Acessado em 15/10/2013, Capturado em: https://launchpad.net/~openstack-ossg, 2013.

[PMP10] Patel, V.; Mohandas, R.; Pais, A. R. “Attacks on web services and mitigation schemes”. In: Proceedings of the 2010 International Conference on Security and Cryptography (SECRYPT), 2010, pp. 1–6.

[PSL13] Perez, D.; Szefer, J.; Lee, R. B. “Characterizing hypervisor vulnerabilities in cloud computing servers”. In: Proceedings of the 2013 international workshop on Security in cloud computing, 2013, pp. 3–10.

[RKT05] Ross, R.; Katzke, S.; Toth, P. “The new fisma standards and guidelines changing the dynamic of information security for the federal government”. In: Proceedings of the Military Communications Conference. MILCOM 2005, 2005, pp. 864–870 Vol. 2.

[SCG+14] Schmerl, B.; Cámara, J.; Gennari, J.; Garlan, D.; Casanova, P.; Moreno, G. A.; Glazier, T. J.; Barnes, J. M. “Architecture-based self-protection: Composing and reasoning about denial-of-service mitigations”. In: Proceedings of the 2014 Symposium and Bootcamp on the Science of Security, 2014, pp. 2:1–2:12.

[Ste14] Stein, L. D.; Stewart, J. N. “WWW security faq: Securing against denial of service attacks”. Acessado em 18/06/2014, Capturado em: http://www.w3.org/Security/Faq/wwwsf6.html, 2014.

[Sub11] Subashini, S.; Kavitha, V. “A survey on security issues in service delivery models of cloud computing”, *Journal of Network and Computer Applications*, vol. 34–1, 2011, pp. 1 – 11.

[Sze13] Szefer, J.; Lee, R. “Bitdeposit: Deterring attacks and abuses of cloud computing services through economic measures”. In: Proceedings of the 2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2013, pp. 630–635.

[Tau09] Taurion, C. “Computação em nuvem: Transformando o mundo da tecnologia da informação”. Brasport, 2009, 180p.

[TG14] Thakare, S.; Gore, D. “Comparative study of cia and revised-cia algorithm”. In: Proceedings of the 2014 Fourth International Conference on Communication Systems and Network Technologies (CSNT), 2014, pp. 713–718.

[TMA11] Tariq, U.; Malik, Y.; Abdulrazak, B. “Collaborative peer to peer defense mechanism for DDoS attacks”, *Procedia Computer Science*, vol. 5–0, 2011, pp. 157 – 164.

[UA09] Udhayan, J.; Anitha, R. “Demystifying and rate limiting icmp hosted dos/ddos flooding attacks with attack productivity analysis”. In: Proceedings of the IEEE International Advance Computing Conference, 2009, pp. 558–564.

[VBL04] VonAhn, L.; Blum, M.; Langford, J. “Telling humans and computers apart automatically”, *Commun. ACM*, vol. 47–2, Fev 2004, pp. 56–60.

[VMW14] VMWare. “Vmware virtualization for desktop server, application, public hybrid clouds”. Acessado em 18/06/2014, Capturado em: http://www.vmware.com/, 2014.

[Wil07] Wilbanks, L. “The impact of personally identifiable information”, *IT Professional*, vol. 9–4, 2007, pp. 62–64.

[Xen14] Xen. “The xen project, the powerful open source industry standard for virtualization”. Acessado em 18/06/2014, Capturado em: http://www.xenproject.org/, 2014.

[Yan08] Yan, J.; El Ahmad, A. S. “A low-cost attack on a microsoft captcha”. In: Proceedings of the 15th ACM Conference on Computer and Communications Security, 2008, pp. 543–554.

[YZ08] Yuan, D.; Zhong, J. “A lab implementation of syn flood attack and defense”. In: Proceedings of the 9th ACM SIGITE Conference on Information Technology Education, 2008, pp. 57–58.

[Zet13] Zetter, K. “In surprise appeal, TJX hacker claims U.S. authorized his crimes”. Acessado em 22/06/2013, Capturado em: http://www.wired.com/2011/04/gonzalez-plea-withdrawal/, 2013.

[Zim80] Zimmermann, H. “Osi reference model–the iso model of architecture for open systems interconnection”, *IEEE Transactions on Communications*, vol. 28–4, Apr 1980, pp. 425–432.

[ZJR12] Zhang, Y.; Juels, A.; Reiter, M. K. “Cross-vm side channels and their use to extract private keys”. In: Proceedings of the 2012 ACM conference on Computer and communications security, 2012, pp. 305–316.

[Zor13] Zorz, Z. “Researchers detail attacks for compromising dropbox user accounts”. Acessado em 10/08/2013, Capturado em: http://www.net-security.org/secworld.php?id=15480, 2013.

[ZZX+10] Zhou, M.; Zhang, R.; Xie, W.; Qian, W.; Zhou, A. “Security and privacy in cloud computing: A survey”. In: Proceedings of Sixth International Conference on Semantics Knowledge and Grid (SKG), 2010, pp. 105–112.

APÊNDICE A – ARTIGO PUBLICADO NO 9TH ICITST

Neste apêndice é apresentado o artigo "*Mitigating DoS to authenticated cloud REST APIs*" publicado na conferência ICITST-2014.

- Título: *Mitigating DoS to authenticated cloud REST APIs*
- Conferência: *9th International Conference for Internet Technology and Secured Transactions*
- URL: http://icitst.org/
- Data: 08-10 de Dezembro 2014
- Local: Londres, Inglaterra
- Submetido: 15 de Setembro 2014
- Índice: H5-Median: 20

Mitigating DoS to authenticated cloud REST APIs

Régio A. Michelin, Avelino F. Zorzo, Cesar A. De Rose
Computer Science School
Pontifical Catholic University of Rio Grande do Sul
Porto Alegre - Brazil
regio.michelin@acad.pucrs.br, avelino.zorzo@pucrs.br, cesar.derose@pucrs.br

Abstract—**Systems available on the Internet are day-by-day targets of Denial of Service (DoS) attacks. These attacks can leave a system with high response time or even make it unresponsive. A DoS attack can be executed at the network level, just by exploiting communication protocols weakness, or at application level, by exploiting implementation issues. Based on this scenario, this article presents a mechanism for mitigating DoS attacks aimed at exploiting REST applications using authentication tokens. This mitigation is based on the client behaviour, where it can be classified as possible malicious client. Our results show a response time decrease of 36% during an attack scenario applied to a cloud management system.**

Keywords-component; DoS; security; REST; cloud

I. INTRODUCTION

Nowadays, more and more systems become available on the Internet facilitating their exposure to several different types of attack. These attacks are intended to steal some information, deploy malicious code and even to make a system slow to respond, or worst, to become completely offline. This last kind of attack is called Denial of Service (DoS) attack, and its main goal is to bring a whole system offline, or at least make it very slow [1]. In order for the attack to achieve its goal, it consumes all computer resources like network bandwidth, CPU cycles or storage space. Once the system is compromised, legitimate clients are not able to have theirs requests responded.

When a malicious user is able to consume all computer resources from its target, and make the computer system unavailable for legitimate users, the attacker, in some cases, uses this DoS attack to perform extortion on their target. Recently the Feedly web site [2], which is a RSS feed aggregator, was victim of a powerful DoS attack that consumed its server's bandwidth and legitimate users were not able to access it. During the DoS, attackers contacted the web site owners asking for ransom to stop the attack.

The DoS usage increased in last years due to the increase of availability of services on the Internet, which was facilitated by the grow of cloud computing. A system on the cloud consists of physical or virtual machines that can be rented by developers that pay only for what they use. So, from the developers perspective, they do not need to worry with cloud infrastructure because its management is delegated to the cloud company (this business model is called Infrastructure as a Service - IaaS) [3]. Developers hove to be only concerned on writing the application service.

Many cloud companies grow using this cloud model, which became popular due to the advances of virtualization technology. Virtualization allows several virtual machines to share the same hardware. These virtual machines are, usually, managed by a software called hypervisor [4], *e.g*. VMWare [5], Hyper-V [6], XEN [7] and KVM [8].

However, a hypervisor is not enough to create a cloud, it is also necessary to include a system responsible to orchestrate the usage of several different hypervisors, physical computer allocation, storages, etc. Also, this system must provide a self-service interface to allow cloud customers to manage their own virtual machines [9]. Examples of Cloud Management Systems (CMS) responsible to provide this kind of feature are, for example, OpenStack [10], CloudStack [11] or Eucalyptus [12].

On the application level, a CMS provides several ways to allow cloud customers to interconnect their own systems and to manage virtual machines and services that will run on the cloud [13]. One way for doing that is through REST (Representational State Transfer) calls [14]. REST is an abstraction architecture that allows distributed systems to communicate over networks. However, once REST is available to customers, the cloud company must consider how it will be used to avoid that an attacker compromises the whole cloud system, which will impact several different cloud clients.

One strategy to avoid damage to a CMS is to use an authentication mechanism; hence only authenticated users will be allowed to perform operations using REST [15]. This authentication can be performed through user name and password, and upon a correct pair of user name and password, the CMS generates a token that will be used to allow the user to access REST operations. This kind of authentication is provided by different CMSs, but due to the way authentication is provided by the CMS, it is possible to explore that for DoS attacks.

Therefore, this work proposes a mechanism to mitigate DoS that attack REST calls in CMSs. This will be achieved by analysing client requests performed through REST calls, and based on the client information and behaviour, *i.e*. whether it is a legitimate client or not, to block REST calls. This mechanism is based on the client IP and a timed control queue. As a case study, we analysed the Keystone [16] component of OpenStack, which is the module responsible for identity management. Although we have applied our solution to a CMS, we believe that our solution can be applied to any application on the Internet that uses REST APIs.

This paper is organized as follows. Section II presents related work. Section III describes how the problem was observed in REST that relies on authentication tokens. Section IV proposes a solution to mitigate the DoS attacks. Section V shows our solution applied to OpenStack Keystone module. Section VI presents our conclusions and future work.

II. Related Work

Lu [13] work presents an empirical study related to cloud APIs. In his work, Lu analyses the Amazon Elastic Compute Cloud (EC2) APIs, and shows a quantitative classification related to its API. The majority of cases that cause the API failure are related to the call being unresponsive. There are also a significant portion of cases in which they brought the system to provide slow response time.

Kargl [17] studied the first DoS as well as its change to the DDoS (Distributed DoS), performed through several machines infected by daemons that allow an attacker to remotely control the machine. To defend a system against these attacks, he proposes a Linux kernel change that includes a mechanism to route different packages through round robin and last connection. This change basically creates a load balancing in order to properly distribute client requests among servers. Kargl solution works in a network level, and once the package is received, it is validated and if the client is in a suspect queue, the package is dropped.

The research performed by Beitollahi [18] presents a DoS defense mechanism operating at application level. In his work, Beitollahi creates a mechanism that attributes different points to each received connection, based on connection history and statistics. However, this solution is applied only if the client is a human, because when it detects a suspect behaviour it sends a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) to the client. However, since in our scenario the client will be other system interacting through REST, Beitollahi's solution cannot be applied.

A collaborative defense system against DoS is proposed by Tariq [19]. This defense system is deployed in several nodes over a network, so when a node detects any malicious traffic, it sends a sign to other nodes and then the malicious traffic is filtered, avoiding it to reach the target system. This control and analysis is performed by collecting packages in a time frame window, and the collected data is compared with known DoS behaviours. This research works at network level, so for application level attacks, which create a valid connection, it will not work properly allowing the malicious traffic to hit its target.

III. DoS in REST API

A. DoS Taxonomy

DoS attacks consist basically in consuming all system available resources. The attack target goals are bandwidth, when this attack is performed at network level, and CPU and storage, when the attack is performed at application level. In the latter case, the attack usually is more complex due to the need to create many valid requests to its target, making this attack much more expensive than a regular network level DoS attack. Thus the attack can vary based on its characteristics.

The different characteristics allow the DoS attack to be classified in different ways, *i.e.* it can be executed in different levels from network to application level. Mirkovic [20] presents a taxonomy to classify several different types of DoS attacks, as well as defense mechanisms. Actually, Mirkovic classifies Distributed DoS (DDoS), *i.e.* attacks that are performed by several different clients aiming a single target system. Figure 1 presents a simplified version of Mirkovic DDoS attack classification.

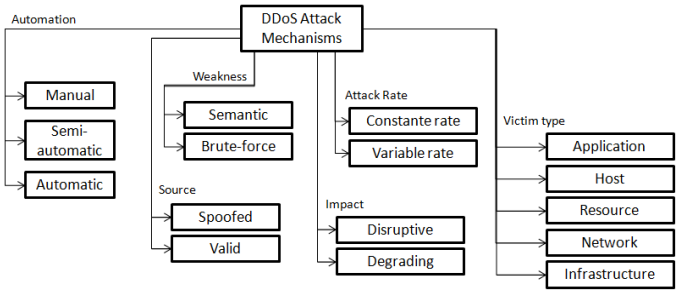

Figure 1. Taxonomy of DDoS attacks adapted from Mirkovic [20] work

As shown in Figure 1, the source is used to evaluate if the attack is performed from a valid IP address, normally when the attack targets a specific application, or spoofed IP address, applied to create more noise during attack to avoid its detection. The exploited weakness can be semantic when the attack targets a specific protocol implementation bug or even a feature. During the attack, the number of packages sent can be used to classify it to a constant or random rate. Depending on the attack characteristics, it will let the target system slow or, in the worst case, completely offline, not allowing legitimate clients to have their valid requests answered.

B. Authentication on REST

Nowadays, as mentioned before, many requests performed over Internet are based on the architectural style known as REST [14]. REST calls are very important part to consider when designing a system over the Internet or cloud, because through this type of calls, different operations are exposed and many different applications can be integrated through them. Once REST calls are used, services will be available through the network to be accessed by users or systems. At this point, anyone with network access is able to perform a call to a REST operation, *i.e.* any malicious user (attacker) can use this exposure to try to compromise the system, for example, to perform a DoS.

The exposed REST services can be divided into services that do not require any authentication, *i.e.* the user does not need to provide a user name and password or token to execute an operation; and services that require an authentication mechanism, which usually is performed by using user name and password.

Figure 2 shows the steps performed during a typical REST operation that uses an authentication mechanism. First, a client accesses the system address sending the user name and password (1). This is performed, usually, through a secure

connection. The system validates this information and, if the user name and password are correct, a token is generated and stored in a database (2). After that, that token is returned to the user (3). Therefore, the user does not need to send the user name and password again every time a REST call is performed. So every time the user wants to perform an operation, the desired operation with the generated token is sent to the service (4). Once the token is received by the REST service, it must verify whether the user is allowed to execute the operation or not. This validation is performed consulting a database (5) in order to identify the user's permission level. If the token exists and the user has enough privileges, the operation is executed.

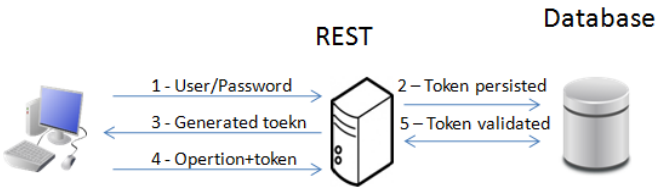

Figure 2. Regular REST flow considering authentication

C. *Performing the DoS attack*

REST services allow a client to execute several different operations that provide information about the system, for example, list of virtual machines, list of system tenant users, etc. If the client is not authenticated before executing these operations, the system can be compromised. Even though the authentication prevents a malicious user to access vital information about the system, token validation action can be a very attractive target for malicious users, because they can, continuously, send invalid tokens to try to overload that system. This might happen since the system will have to validate each invalid token that is submitted.

Basically, the service overload problem happens because upon receiving each request, the application has to check the database (or something similar to a database, *i.e*. a storage that contains valid tokens) in order to identify if the received token is valid and what are the operations the owner of that token can execute. This query may be time consuming depending on the type of access to the database. Traditional detection and traffic block mechanisms for DoS are not applied to this scenario, because all the incoming traffic (on the network level) is valid, *e.g*. the XML (eXtensible Markup Language) or JSON (JavaScript Object Notation) contents sent through REST calls are valid, only the token information is invalid. Usually, most DoS defense mechanisms work at network level, which consider only the information inside each network packet. Even some works that detect the malicious behaviour at application level, normally consider that on the other side of the connection there is a human user, not a system [18] (see Section II).

Figure 3 presents the scenario where several malicious users are sending malicious traffic in order to consume the CPU target system, and then leading the legitimate client responses being delayed or even denied.

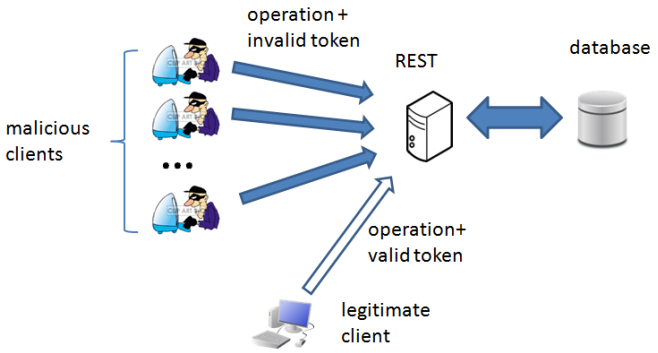

Figure 3. Attack scenario to a sytem with REST calls

IV. PROPOSED SOLUTION

As mentioned in the problem described in Section III, the service overload leads to resource depletion when the DoS attack is executed. As mentioned in Section II, there are several different DoS attacks based on different attack aspects (see Figure 1). Based on that taxonomy, this work focus on attacks that are automatic (automation), based on brute force (weakness), constant (attack rate), where the attackers have valid IPs (source), and their target is an application (victim type). The impact of the attacks can start as degrading, but can be disruptive. The proposed defense mechanism is reactive based on anomaly detection, running on the target computer, which takes an action based on the attacker agent identification.

Based on the presented problem (see Section III) and the above classification, we propose a defense mechanism at application level in order to minimize the unnecessary validation of tokens, avoiding the system to query the database. This implementation assumes that a legitimate client will not flood the system with invalid tokens.

Our mechanism works in two modes: monitoring and filtering. During monitoring, our mechanism verifies whether the system is being stressed or not, for example, when the CPU is overloaded, *i.e*. more than 70% of usage (this can be set with a different percentage depending on the type of system in which our mechanism is applied to). If the CPU is not overloaded, it takes the requests from each client, and if it detects that the requests contain invalid tokens, it marks this client as a probable attacker and includes this client in a gray list. When the system is overloaded, then our mechanism moves from monitoring mode to filtering mode, in which the gray list becomes a black list and clients that are in this list have their REST calls dropped. Therefore, any request performed by a client that is in the black list will be discarded as soon as it is received. Our mechanism gets back to monitoring mode again when the system is not overloaded, for example, CPU usage is 30%. Notice that there is a difference between how we consider whether the system is overloaded or not. This is performed to avoid our mechanism to change modes too frequently. Imagine the situation in which our mechanism changes from monitoring mode to filtering mode and right after it starts to drop REST packets, the CPU usage of the system, for example, becomes 69%. If we considered that the system is not overloaded anymore and we move our

mechanism to monitoring mode again, we stop dropping REST packets, but in the next instant of time, the CPU usage could become 70% again, so our system would move again to filtering mode. This process could keep changing while the attack takes place, basically keeping the CPU usage at a peak that was not necessary if we dropped packets from the attackers. This would not make the system disruptive, but it would keep the system working in a degraded mode.

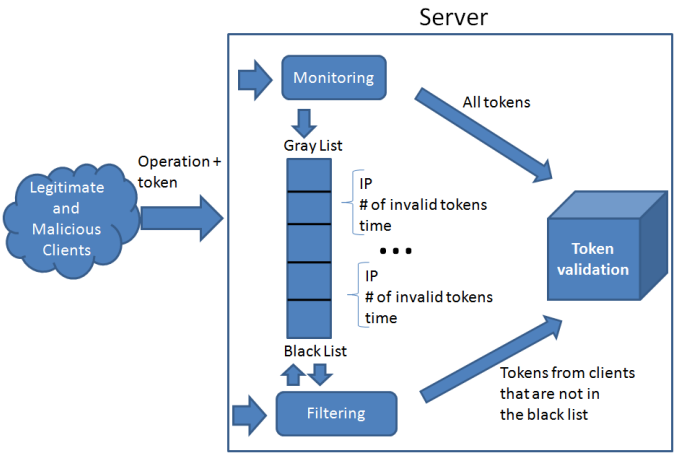

Figure 4. Client control solution achicteture

Figure 4 shows our solution architecture, in which legitimate and malicious clients send REST operations and tokens. Notice that when our mechanism is in filtering mode, some invalid tokens will be verified if the clients are not in the black list. When this happens, during filtering mode, our mechanism includes those clients in the black list and their REST calls are also dropped. Requests from clients that are in the black list will be dropped for a period of time (window). After that window, our mechanism allows one request to be verified to check if the client now has a valid token. This may happen when a legitimate client sent invalid tokens (expired for example) and was included in the gray/black list, but after some time got a valid token. If we did not do that, legitimate clients could be blocked forever. If the client is malicious and keeps sending invalid tokens, then the time they are blocked (window) is increased.

V. Case Study: Keystone Module of Openstack

In order to evaluate our proposed solution, we applied our mechanism in an open source CMS, *i.e*. OpenStack, which is composed by the following components: Cinder (responsible for controlling block storages for the virtual machines), Glance (responsible for managing operating system images), Swift (controls object storages), Neutron (responsible for network management), Horizon (provides the graphical user interface), Nova (orchestrate all OpenStack components), and Keystone (responsible for identity management). These different modules communicate through REST calls. For example, when a user accesses the cloud via the Horizon module, a token is generated by the Keystone module. Every time a client wants to execute a new operation, for example, to create a new virtual machine or to list the available disks, a REST call is made to the Horizon module that sends this call to the corresponding module, *i.e*. Nova or Cinder. These modules will send a REST call to Keystone to verify whether the token is valid or not.

As mentioned above, OpenStack modules communicate among them through REST calls (In the Figure 5, arrows represent these calls).

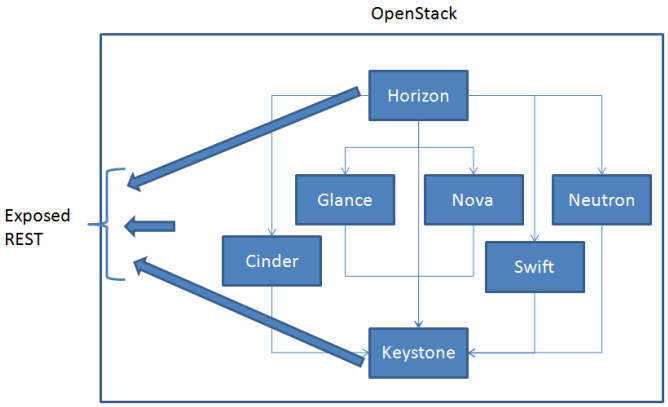

Figure 5. Simplified OpenStack archicteture

Figure 5 shows a simplified OpenStack version. As can be seen in the figure, the Keystone module is central to the whole functioning of OpenStack, since it is responsible for the identity management. Therefore, Keystone is contacted by all modules to verify whether an operation can be executed or not. This is done because Keystone is responsible for storing all valid tokens and for checking if an arriving token is valid or not. Hence, Keystone is a good target for malicious users that want to overload the whole infrastructure provided by OpenStack. Once this user can get access to Keystone REST, he can send multiple requests containing invalid tokens, and this will force the component to consult the database, overloading the system.

In order to avoid Keystone REST exposure, during OpenStack installation, it is possible to configure the Linux firewall to block any call to Keystone REST. Hence, only users with access to the OpenStack management network are allowed to access the REST API. Despite the firewall protection, in some situations, like when a module has to be integrated with third party software, this REST must be exposed. Therefore, this default installation must be changed in order to allow access to the Keystone REST from different networks. In this work we assume the situation in which REST has to be exposed.

Our experiment focuses on two Keystone REST operations to be stressed, *i.e*. simulating an attack. After the attack is performed, we retrieve the requests response time to verify what the impact on the Keystone module is. The operations that are called are: token generation, where we assume that a real system user is calling the REST sending a valid user and password; and token validation through tenant list operation, so this operation requires a token to list all system tenants. Figure 6 shows the time difference when both RESTs where overloaded. The experiment executes three different scenarios: i) clients start calling tenant list REST operation sending

invalid tokens; ii) the same operation but now in a stress scenario where all clients are sending valid tokens; and, iii) stress situation with clients sending valid users and passwords but calling the token generation operation. In the three execution scenarios, our system started in an idle state, and the number of clients sending requests increased until 180000 requests. As can be seen in Figure 6, response time degrades significantly faster for invalid token validation than for token generation or even for valid token validation.

A. Results Analysis

Since all cloud system were built on top of virtualization concept (and using different hypervisors), we chose to run our experiment on a virtualized infrastructure. The virtualization system used was VMware workstation 10.0.1 running on an Intel Core i5-4570@3.20GHz platform, with 16GB of RAM bus DDR3 1600 and Microsoft Windows 7 Professional 64 bits Operating System. The virtual machine where OpenStack Grizzly was running uses 4GB of RAM and 2 processors using Linux Ubuntu Server 14.04. The virtual machines for the client have 2GB of RAM and 1 processor running on Linux Ubuntu Server 14.04. The client virtual machines are used for starting the attack scripts.

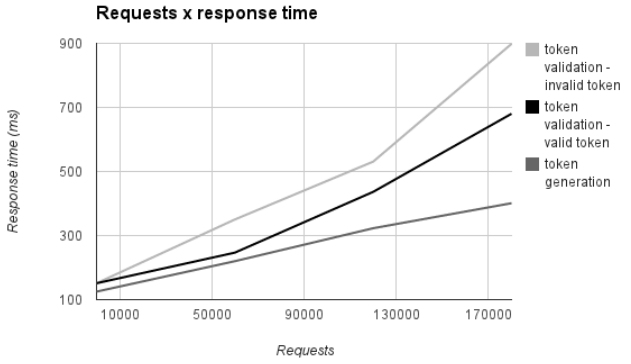

Figure 6. Response time increases when component is stressed

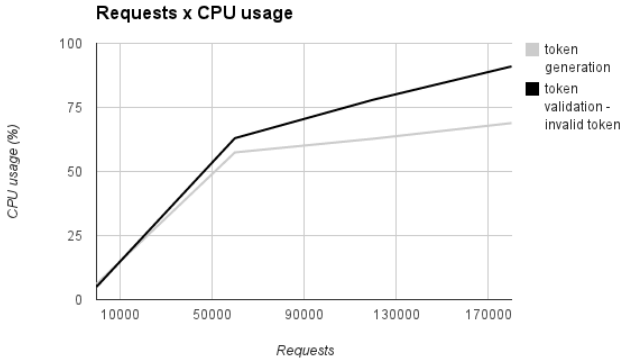

Figure 7. CPU usage during attack

Figure 6 shows the response time, in milliseconds, for both REST operations (token validation and generation). The system handling 180000 invalid tokens requests for the Keystone REST tenant list operation, would take 898 milliseconds (error 10%) to respond to a valid client. The same system validating valid tokens for 180000 valid requests, would take 425 milliseconds (error 16%). On the other hand, while the system was idle, this token generation time was 151 milliseconds (error 11%). Besides this response time growth, the processor usage also increased, reaching around 90%. Figure 7 shows this use and also that token validation is the operation that demands more processor usage.

Once the problem was identified in the Keystone REST operation, our solution was deployed in order to analyse the received tokens. Running the system again with the same load of 180000 and performing the tenant list operation with an invalid token, the response time drops to 328 milliseconds (error 10%). This time is approximately 36% faster than the same scenario without our solution. Figure 9 shows this time difference.

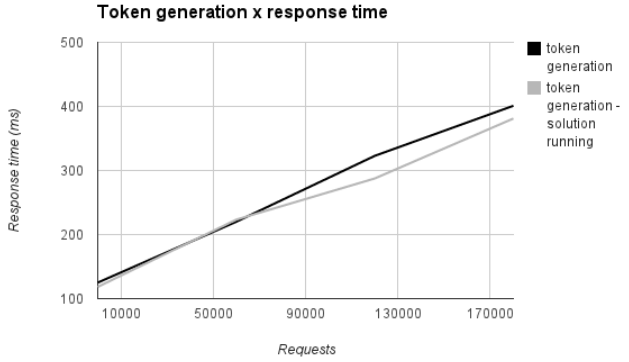

Figure 8. Response time during token generation

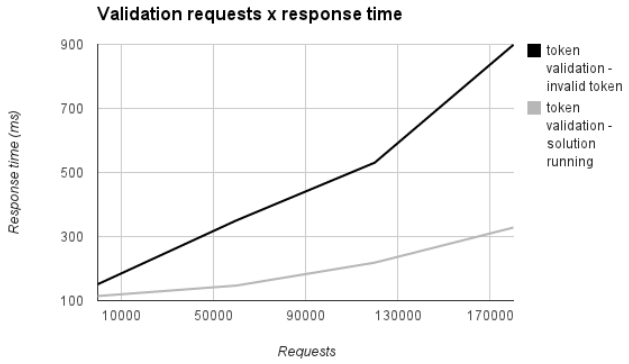

Figure 9. Response time with our solution running

Figure 8 shows the token generation during a system stress scenario. We noticed that this response time varied from 401 milliseconds to 381 milliseconds considering an error of 15% to our samples. This behaviour happens since the proposed solution works only for handling the token validation operation. We also noticed the processor usage decreased when our solution is running. Figure 10 shows the processor usage on the system when our solution was applied. Our solution was responsible for keeping its usage around 40%, against 90% usage when no solution was applied.

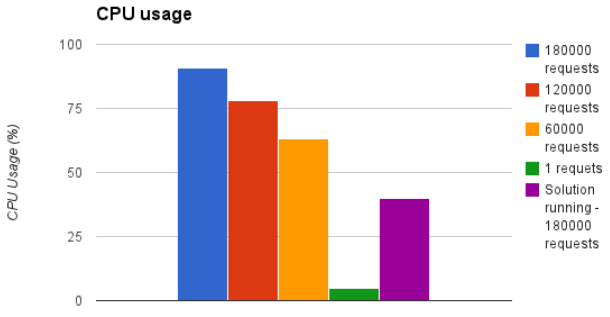

Figure 10. Processor usage with our solution running

VI. CONCLUSION

In this paper we presented a solution to avoid DoS attacks made on application level. These DoS attacks would be performed exploiting the authentication mechanism of REST calls, since every time a call is made, it is necessary to verify whether a token, generated at the beginning of the process, is valid or not. This might cause a system to overload if a huge amount of invalid tokens is sent during normal operation. Naturally, as our solution works only for DoS attacks on application level, it is important to include also some kind of protection to avoid DoS on network level. For example, if an attack exploits a TCP/IP failure or even a network package flood, then a solution like the one proposed by Tariq [19] would successfully protect the system. Therefore, if these, and other, solutions are used together, the system would be more resilient against DoS attacks.

It is important to notice that when our solution is running, there was an improvement on the response time when the system is under attack (see Figure 9). The response time was improved in 36% when our solution is running, so even during a DoS attack, the system still responds in an acceptable time.

As future works we intend to analyze the attacks that happen during a time window. This time window will be used to verify the amount of valid and invalid requests that are being sent from the same IP address. This will be useful to avoid dropping packets from legitimate clients that are behind a NAT solution, *i.e*. if some attackers are executing behind a NAT, to hide invalid packets with valid packets, our extended solution would allow the CMS to reconfigure, dynamically, the percentage of packets from that IP that might be filtered or not.

ACKNOWLEDGMENT

The authors would like to thank Mauro Storch the discussions on cloud computing and to HP R&D Brazil for providing a scholarship through Brazilian law 8.248/91.

REFERENCES

[1] L. D. Stein and J. N. Stewart, "WWW security faq: Securing against denial of service attacks.", 2003, http://www.w3.org/Security/Faq/wwwsf6.html (Access Date: 04 Jun, 2014).

[2] Feedly, "Denial of service attack [neutralized] — building feedly.", 2014, http://blog.feedly.com/2014/06/11/denial-ofservice-attack/ (Access Date: 05 Aug, 2014)

[3] W. Dawoud, I. Takouna, and C. Meinel, “Infrastructure as a service security: Challenges and solutions,” in 7th International Conference on Informatics and Systems (INFOS), 2010, pp. 1–8.

[4] D. Perez-Botero, J. Szefer, and R. B. Lee, “Characterizing hypervisor vulnerabilities in cloud computing servers,” in Proceedings of the 2013 international workshop on Security in cloud computing, 2013, pp. 3–10.

[5] I. VMWare, "Vmware virtualization for desktop server, application, public hybrid clouds.", 2014, http://www.vmware.com (Access Date: 03 Jul, 2014)

[6] I. Microsoft, "Virtualization for your modern datacenter and hybrid cloud.", 2014, http://www.microsoft.com/enus/server-cloud/solutions/virtualization.aspx (Access Date: 03 Jul, 2014)

[7] Xen, "The xen project, the powerful open source industry standard for virtualization.", 2014, http://www.xenproject.org/ (Access Date: 03 Jul, 2014)

[8] KVM, "Kernel based virtual machine.", 2014, http://www.linux-kvm.org/page/Main Page (Access Date: 03 Jul, 2014)

[9] P. M. e Timothy Grance, "The NIST definition of cloud computing.", 2002, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf (Access Date: 23 Jul, 2014)

[10] OpenStack, "Openstack open source cloud computing software.", 2014, https://www.openstack.org/ (Access Date: 03 Sep, 2014)

[11] A. CloudStack, "Apache cloudstack: Open source cloud computing.", 2014, http://cloudstack.apache.org/ (Access Date: 23 Jul, 2014)

[12] Eucalyptus, "Eucalyptus open source private cloud software.", 2014, https://www.eucalyptus.com/ (Access Date: 23 Jul, 2014)

[13] Q. Lu, L. Zhu, L. Bass, X. Xu, Z. Li, and H. Wada, "Cloud API issues: an empirical study and impact," in Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures, 2013, pp. 23–32.

[14] R. T. Fielding, "Fielding dissertation: Chapter 5: Representational state transfer (rest).", 2000, http://www.ics.uci.edu/ fielding/pubs/dissertation/restarchstyle.htm (Access Date: 03 Jun, 2014)

[15] R. Gracia-Tinedo, M. Sanchez Artigas, and P. Garcia Lopez, “Cloudas-a-Gift: Effectively exploiting personal cloud free accounts via REST APIs,” in Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on, June 2013, pp. 621–628.

[16] OpenStack, "Welcome to keystone, the openstack identity service!", 2014, http://docs.openstack.org/developer/keystone/ (Access Date: 03 Jun, 2014)

[17] F. Kargl, J. Maier, and M. Weber, “Protecting web servers from distributed denial of service attacks,” in Proceedings of the 10th International Conference on World Wide Web, 2001, pp. 514–524.

[18] H. Beitollahi and G. Deconinck, "Tackling application-layer DDoS attacks" Procedia Computer Science, vol. 10, no. 0, pp. 432 – 441, 2012, ANT 2012 and MobiWIS 2012.

[19] U. Tariq, Y. Malik, B. Abdulrazak, and M. Hong, “Collaborative peer to peer defense mechanism for DDoS attacks,” Procedia Computer Science, vol. 5, pp. 157 – 164, 2011.

[20] J. Mirkovic and P. Reiher, “A taxonomy of DDoS attack and DDoS defense mechanisms,” SIGCOMM Computing Communications, vol. 34, no. 2, pp. 39–53, Apr. 2004.

APÊNDICE B – OUTROS ARTIGOS

Artigo submetido como coautor:

- Título: *Multi-channel Secure Interconnection Design for Hybrid Clouds*
- Autores: Mauro Storch, Cesar A. F. De Rose, Avelino F. Zorzo, Regio Michelin
- Conferência: *The Eleventh International Conference on Networking and Services* - ICNS 2015
- URL: http://www.iaria.org/conferences2015/ICNS15.html
- Data: 24-29 de Maio 2015
- Local: Roma, Itália
- Submetido: 30 de Dezembro 2014
- Índice: H5-Median: 20