Dispositivos conectados a serviço web em nuvem: complementariedade entre infraestrutura e seletividade para proteção contra DDoS

João H. Corrêa[1], **Epaminondas Aguiar**[2], **Pablo Brunetti**[1], **Vivek Nigan**[3]
Iguatemi E. Fonseca[3], **Rodrigo Laiola Guimarães**[1], **Magnos Martinello**[1]
Moisés R. N. Ribeiro[2], **Rodolfo S. Villaça**[1]

[1]Programa de Pós-Graduação em Informática (PPGI)
Universidade Federal do Espírito Santo (UFES)

[2]Programa de Pós-Graduação em Engenharia Elétrica (PPGEE)
Universidade Federal do Espírito Santo (UFES)

[3]Programa de Pós-Graduação em Informática (PPGI)
Universidade Federal da Paraíba (UFPB)

{jhenrique,epa480,vivek.nigan,rlaiola}@gmail.com, pablobrunetti@hotmail.com,
iguatemi@ci.ufpb.br, {magnos,rodolfo.villaca}@inf.ufes.br,
moises@ele.ufes.br

Abstract. *The denial of service (DoS) or its distributed form (DDoS) attacks are among the most important business impacts nowadays, affecting service availability severely. Application Layer Denial of Service attacks exploit vulnerabilities in protocols and the main challenge in mitigating such attacks is due to the fact that attacker requests have the same status as legitimate clients. This paper proposes to use, automatically, the instantiation of more servers to defend attacks with high rate, and the SeVen to mitigate attacks with low rate. Experiments in the OpenStack cloud environment show that the infrastructure can be used as a form of defense against these types of attacks.*

Resumo. *Os ataques de negação de serviço (DoS), ou sua forma distribuída (DDoS), estão entre os que mais impactam nos negócios, afetando a disponibilidade do serviço. Ataques de negação de serviço na camada de aplicação exploram características existentes nos protocolos desta camada e um grande desafio na mitigação deste tipo de ataque é o fato que as requisições de atacantes tem igual validade a de clientes legítimos. Este trabalho propõe utilizar, de forma automática, a instanciação de mais servidores para suportar ataques com alta demanda, e a ferramenta SeVen para mitigar ataques com baixa taxa. Experimentos realizados no ambiente de nuvem OpenStack mostram que a infraestrutura pode ser utilizada como forma de defesa contra esses tipos de ataques.*

1. Introdução

Ataques de negação de serviço são problemas reais nas redes de computadores, seja a sua forma realizada por um dispositivo/atacante (DoS - *Denial of Service*), ou sua forma distribuída, realizada por vários dispositivos/atacantes (DDoS - *Distributed Denial of Service*). De acordo com o relatório da Verisign de 2018[1], esses tipos de ataques são os que

[1]https://www.verisign.com/en_US/security-services/ddos-protection/what-is-a-ddos-attack/index.xhtml

mais impactam nos negócios atualmente, principalmente em serviços financeiros, serviços de tecnologia da informação e de telecomunicações. A intenção do ataque é simplesmente fazer com que um serviço, como uma página *web*, por exemplo, pare de responder a clientes que solicitam atendimento. Ou seja, é uma tentativa de um usuário não legítimo de degradar ou negar recursos a um usuário legítimo [Shea and Liu 2012].

De acordo com os relatórios da Akamai de 2016[2] e do terceiro trimestre de 2017[3], dispositivos de IoT (*Internet of Things*) estão sendo utilizados para integrar *botnets*, com intuito de realizar ataques de negação de serviço do tipo DDoS. Isso se deve, principalmente, à simplicidade de projeto dos dispositivos de IoT, que visam exercer uma determinada função, mas não se preocupam com questões de segurança. Um exemplo dessas *botnets* é a Mirai[4] [Antonakakis et al. 2017].

Atualmente, os ataques, que antes utilizavam protocolos das camadas de Rede e/ou Transporte, migraram para a camada de Aplicação, que visam indisponibilizar apenas um serviço dentro do servidor. Além disso, os ataques de negação de serviço na camada de Aplicação (ADDoS) tem como característica principal a geração de um tráfego semelhante ao de clientes honestos [Zargar et al. 2013]. Dessa forma, as ferramentas tradicionais de análise do tráfego, não detectam esses tipos de ataques [Dantas et al. 2014].

Dentre os ataques ADDoS, o tipo *high-rate* utiliza características dos protocolos da camada de aplicação, para realizar uma inundação de requisições, como por exemplo, o ataque *Get-Flooding*, que utiliza o método *GET* para realizar um pedido a uma página da internet. O ataque consiste em realizar uma grande quantidade de requisições, e como se assemelha a pedidos de clientes legítimos, o servidor *web* aloca recursos para atender a todos esses pedidos. Esse tipo de ataque, de acordo com relatório do ano de 2014, divulgado pela NSFOCUS[5], foi o quarto mais praticado no mundo. De acordo com Akamai, no quarto trimestre do ano de 2017, dentre os ataques da camada de aplicação, o *Get-Flooding* é o mais realizado. Além disso, de acordo com [Herzberg et al. 2016], a *botnet* Mirai utilizou o ataque HTTP *floods*.

Diante disso, o objetivo deste trabalho é apresentar uma contra-medida para aplicativos em nuvem contra ataques de negação de serviço na camada de aplicação, principalmente do tipo *low-rate* e *high-rate*, tais como o *Slowloris* e *Get-Flooding*, oriundos de dispositivos conectados, entre eles os utilizados em IoT. Diante da dificuldade de se identificar cliente legítimo e um tráfego malicioso nesse tipo de ataque, essa defesa consiste em não realizar descarte de qualquer tipo de tráfego, pois há possibilidade de clientes legítimos serem afetados. Dessa forma, a estratégia é oferecer mais infraestrutura para que a demanda seja atendida. Ou seja, quando há a necessidade de mais recursos para atendimento, a nuvem instancia, de forma automática, novos servidores *web*. Esse processo, chamado de *auto scale*, confere ao sistema um nível de escalabilidade. Assim, a demanda é atendida sem que haja algum tipo de indisponibilidade (seja por falta de

[2]https://www.akamai.com/us/en/our-thinking/state-of-the-internet-report/global-state-of-the-internet-security-ddos-attack-reports.jsp

[3]https://www.akamai.com/de/de/multimedia/documents/state-of-the-internet/q3-2017-state-of-the-internet-security-report.pdf

[4]https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/

[5]http://www.prnewswire.com/news-releases/2014-mid-year-ddos-threat-report-documents-high-volume-high-rate-attacks-on-the-rise-276438821.html

atendimento ou a utilização de técnicas de descarte). Após o ataque, a própria nuvem se encarrega de diminuir a quantidade de servidores, realizando assim uma economia de recursos.

Essa abordagem também se torna efetiva com um alto volume de clientes legítimos, quando é gerado uma negação de serviço apenas pela grande quantidade de solicitações de clientes. Por exemplo, em grande promoções de *sites* de venda como a *blackfriday*; em época de inscrição para cursos superiores em universidades públicas; ou então em final de prazo para a entrega da declaração do imposto de renda. Dessa forma, a estratégia de utilizar a nuvem para provisionar novos servidores, de forma automática, consegue eliminar a indisponibilidade gerada pelo mau dimensionamento da infraestrutura por parte dos administradores de rede.

Para os ataques do tipo *low-rate*, como o *Slowloris*, há uma defesa, chamada *SeVen*, proposta por [Dantas et al. 2014], que consegue oferecer uma disponibilidade de 95% para clientes legítimos. No entanto, quando se utiliza o *SeVen* contra ataques do tipo *high-rate*, a defesa não consegue ser eficaz, provendo disponibilidade a apenas 13,6% dos clientes, como discutido na Seção 4.2.1. Por outro lado, a simples utilização do processo de *auto scale* não é suficiente para mitigar ataques do tipo *low-rate*, pois esses ataques não elevam métricas que poderiam acionar o processo de escalabilidade. Assim, a presente proposta visa utilizar a defesa *SeVen* combinada com o processo de *auto scale*, oferecida pela plataforma OpenStack, para criar uma defesa capaz de suportar ataques combinados do tipo *low-rate* e *high-rate*, oriundas de dispositivos conectados. Dessa forma, a eficiência da defesa contra ataques de DDoS está na complementariedade da infraestrutura e da seletividade.

O restante do artigo está estruturado da seguinte forma: a Seção 2 apresenta alguns trabalhos relacionados ao presente artigo; na Seção 3 será discutido as ferramentas que habilitam a utilização da infraestrutura como defesa a ataques de negação de serviço; na Seção 4 serão apresentados os cenários elaborados para verificar a proposta, os resultados obtidos e a discussão desses resultados; por fim, a Seção 5 conclui o trabalho, inclusive com indicações de trabalhos futuros.

2. Trabalhos Relacionados

O rápido desenvolvimento da computação em nuvem envolve questões importantes relacionadas à segurança [Balobaid et al. 2016, Lai et al. 2016]. Em [Balobaid et al. 2016] há o estudo do impacto de diversos tipos de ataques de DoS e DDoS na nuvem OpenStack e os procedimentos que são utilizados para afetar a disponibilidade de serviços. Além disso, diversas técnicas de mitigação são discutidas. Este trabalho utiliza a plataforma OpenStack, porém utiliza diferentes ferramentas para simulação de ataques e defesa sem o uso do *auto scale*.

Em [Lai et al. 2016] é proposto um SPDS (Sistema de Decisão de Política com Segurança) na nuvem, usando OpenStack, para criar múltiplos vIDS para detectar ataques DDoS e combinar com múltiplas vFirewall para filtrar os pacotes de ataque. O SPDS proposto faz uso da tecnologia SDN (Redes Definidas por Software) para distribuir o fluxo para vários vIDS, fazendo análise dos pacotes de ataque ou direcionando o tráfego em outro lugar do processamento. Diferentemente do artigo proposto, em nosso trabalho não é utilizada a tecnologia SDN para distribuição de fluxos e a estratégia para defender

é utilizar o poder de escalar servidores e não rejeitar requisições.

Existe também a possibilidade de se utilizar as propriedades das SDN e NFV (Virtualização de Funções de Rede) para propor soluções de gerenciamento e segurança de redes. É proposto em [Mauricio et al. 2017] uma arquitetura de NFV que oferece proteção de forma eficiente contra ataques. São adicionados três novos componentes na arquitetura NFV do ETSI, sendo: um módulo Controlador de Segurança e duas funções virtuais de rede específicas, IDS e Firewall, encadeadas em um fluxo de SFC (*Service Function Chaining*). O IDS contém uma interface que verifica o tráfego na rede e, ao detectar uma ameaça, envia uma mensagem para o Controlador que cria regras, de forma automática e dinâmica, para o Firewall bloquear. Esse artigo se assemelha ao artigo citado anteriormente, o [Lai et al. 2016]. Dessa forma, difere também com a proposta do presente artigo.

As tecnologias NFV e SDN possibilitam que as operadoras de telecomunicações atribuam funções virtualizadas de rede sob demanda, garantindo a garantia de qualidade de serviço (QoS) e gerenciamento de custos no ambiente de computação através da provisão e orquestração de recursos físicos e virtuais. Nesse foco, [Tang et al. 2015] apresenta uma nova solução de *SLA-aware* e *Resource-efficient Self-learning Approach (SRSA)* para decisões de política de *auto scale*. Com a escalabilidade de recursos é possível ajudar as operadoras de telecomunicações suportar o número máximo de usuários com Garantia do Nível de Serviço, mantendo o consumo de recursos em um nível baixo. O foco deste artigo é *auto scale* para disponibilizar o serviço com alta qualidade em telecomunicações, enquanto que o artigo proposto trabalha com o *auto scale* como auxílio de defesa contra ataques de DDoS.

3. Ferramentas e plataformas habilitadoras para prover contra medidas de ataques de negação de serviço em nuvem

3.1. SeVen

A defesa *SeVen* foi introduzida por [Dantas et al. 2014], que visa propor uma defesa seletiva para mitigar ataques de negação de serviço na camada de aplicação. *SeVen* funciona como um *proxy* e monitora o uso de um servidor *web* sob sua proteção. Quando o servidor *web* não está sobrecarregado, ou seja, o número de pedidos que está sendo processado é menor do que a sua capacidade máxima, *SeVen* permite que qualquer requisição (mesmo de atacantes) possa ser processado. No entanto, quando o servidor *web* está sobrecarregado, isto é, já não consegue processar um novo pedido, *SeVen* decide (usando alguma função de probabilidade) se a aplicação *web* deve processar a nova requisição. Há dois resultados possíveis: (i) *SeVen* decide que não deve processar este novo pedido, assim simplesmente retorna ao cliente uma mensagem que o servidor não está disponível; (ii) *SeVen* decide que deve processar este novo pedido. Em seguida ele também deve decidir qual o pedido que está sendo processado no momento deve ser descartado. Esta decisão é aleatória.

Intuitivamente as estratégias seletivas funcionam porque sempre que um aplicativo está sobrecarregado, é muito provável que ele esteja sofrendo um ataque. Em vez de bloquear todos os novos pedidos (de ambos, os atacantes e clientes legítimos), como feito sem a defesa, *SeVen* abre a possibilidade de nova solicitação de clientes legítimos serem

processados, melhorando assim a disponibilidade do aplicativo. [Dantas et al. 2014] realizou testes com ataques de negação de serviço na camada de aplicação do tipo *low-rate* que também exploram vulnerabilidades do protocolo HTTP, mas que tem uma taxa de requisições extremamente baixa. Esses ataques são conhecidos como *Slowloris* e Ataque *POST*.

Embora *SeVen* funcione bem para mitigar ataques *low-rate*, ele sofre ao tentar mitigar ataques do tipo *high-rate*. Isso ocorre porque *SeVen* tem memória e poder de CPU limitada e não pode lidar com a esmagadora maioria das requisições que chegam quando um aplicativo está sofrendo ataques de DDoS com altas taxas, como no ataque *Get-Flooding*. Quando o ataque é do tipo *low-rate*, como um ataque *Slowloris*, *SeVen* pode garantir serviço a 95% dos clientes legítimos. No entanto, quando um ataque é do tipo *high-rate*, como *Get-Flooding*, *SeVen* só pode garantir serviço a uma média de 13,6% dos clientes legítimos. Esses experimentos estão descritos na Seção 4.2.1.

A metodologia utilizada no *SeVen* também foi utilizada para mitigar ataques de negação de serviço contra aplicações VoIP [Dantas et al. 2016] [Lemos et al. 2016], sendo adaptada a estratégia para esse tipo de aplicação.

3.2. NFV

A virtualização de Funções de Redes (*Network Functions Virtualization* - NFV) consiste primordialmente em substituir funções de rede específicas (implementadas em *network appliances*), tais como, roteadores, *switches*, *firewall* etc, com forte acoplamento de recursos proprietários, por software e automação em hardware de propósito geral.

NFV tem como objetivo tornar as redes mais simples e flexíveis, minimizando a dependência de restrições de hardware com a virtualização de funções específicas da rede. Assim, pode-se citar benefícios esperados com a adoção da tecnologia NFV como a redução de CAPEX e OPEX, a diminuição do tempo para um novo produto chegar ao mercado, a interoperabilidade, melhor escalabilidade e flexibilidade para a instanciação de novos serviços e, por fim, possibilitar um incentivo à inovação e o fortalecimento de plataformas abertas [Herrera and Botero 2016].

A ETSI (*European Telecommunications Standards Institute*) definiu uma arquitetura de referência para a implementação de NFV visando padronizar a forma de como ocorrerá a virtualização de funções de rede virtuais, levando em consideração os recursos disponíveis para tal, bem como a forma como estas serão instanciadas, suas conectividades, dados atribuídos, dependências, controle e outros atributos [ETSI 2013]. Nesta arquitetura destaca-se três blocos: VNF, NFVI e MANO.

O bloco de VNFs (Funções de rede virtualizadas - VNF) representa as implementações via software de funções de rede. Como exemplo de funções de rede, podemos citar: *firewall*, balanceador de carga, IDS, IPS, serviços de NAT, etc. Já o bloco NFVI (Infraestrutura NFV) representa todos os recursos computacionais disponíveis que servição como ambiente para a instanciação, gerenciamento e execução de VNFs.

A contribuição principal da arquitetura é representada no bloco Gerenciamento e Orquestração de NFV (MANO). Este bloco controla a interação da VNF com os recursos físicos sob sua autoridade, realizando gerenciamento de recursos, operações como visibilidade da infraestrutura, coleta de informações para a gerência de falhas e desempe-

nho, além de ser responsável pelo gerenciamento do ciclo de vida das VNFs, incluindo operações como instanciação, atualização e finalização.

3.3. OpenStack

OpenStack[6] é uma plataforma de computação em nuvem pública e privada, de código aberto, que permite controlar grandes conjuntos de recursos de computação, armazenamento e redes pertencentes a um *datacenter*, tudo gerenciado através de uma interface dedicada que permite aos administradores controlar e provisionar recursos para múltiplos usuários.

Um dos principais objetivos da plataforma é construir um serviço de computação em nuvem que pudesse ser instalado em hardware padrão ou "customizado". Sobre esta camada de hardware atuam serviços compartilhados do OpenStack e os componentes responsáveis por controlar os grupos de computadores, armazenamento e recursos de rede. No topo desta estrutura estão a camada de aplicações e administração de acesso, que é representada por uma interface *web*. Dessa forma, de acordo com a arquitetura padronizada apresentada na Seção 3.2, o OpenStack se enquadra no bloco do gerenciamento da infraestrutura virtualizada (VIM).

Um dos motivos para que o OpenStack seja uma plataforma difundida e utilizada é que foi construída de forma modularizada, ou seja, cada função específica dentro do projeto é dividida em um módulo separado. Dessa forma, o desenvolvimento de componentes podem variar, podendo ter grau de maturidade diferente para diversos módulos.

Dentre os módulos do OpenStack, alguns são considerados componentes chaves para a implementação de um ambiente de nuvem, por serem responsáveis pela orquestração de recursos, processamento, armazenamento e rede. Neste trabalho, foram utilizados os seguintes módulos: Nova (provisionamento de instâncias de computação), Keystone (gerência de acesso à nuvem), Glance (gerência de imagens), Neutron (conectividade entre as redes virtuais), Horizon (painel de controle da nuvem via página *web*) e Ceilometer (coleta de dados para monitoramento) e o Tacker (orquestração e gerenciamento a tecnologia de redes virtualizadas).

De acordo com a OpenStack Fundation[7], há algumas funcionalidades do OpenStack que o insere como candidato para habilitar arquiteturas NFV: (i) capacidade de provisionar nuvens públicas e privadas; (ii) interfaces padronizadas entre elementos NFV e a infraestrutura; (iii) contempla a maioria de *plugins* e *drivers* de rede comerciais e de código aberto; (iv) em constante atualização de funcionalidades NFV; (v) capacidade de gerenciar recursos de computação, rede e armazenamento disponíveis em infraestrutura de rede; (vi) serviço de rede maduro e flexível, provendo infraestrutura de rede virtualizada para as funções e; por fim vii) suporta inúmeros tipos de hipervisores.

3.4. Tacker

O módulo Tacker tem como principal objetivo implementar um gerenciador genérico de VNFs e um orquestrador NFV para instalar e operar serviços de rede e VNFs em uma plataforma habilitadora de NFV, como o OpenStack. É baseado na arquitetura ETSI

[6]https://www.openstack.org/

[7]https://www.openstack.org/assets/telecoms-and-nfv/OpenStack-Foundation-NFV-Report.pdf

e provê uma pilha funcional para orquestrar serviços de rede fim-a-fim usando VNFs [Chen et al. 2017].

O Tacker como gerenciador de VNFs é responsável pela instanciação, atualiação e remoção de VNFs, além de prover serviços de monitoramento. Como Orquestrador NFV, tem a responsabilidade de orquestrar e gerenciar de forma ampla os serviços de rede fim-a-fim, tendo uma visão global desses serviços na infraestrutura. O Tacker trabalha com o conceito de catálogos de VNFs que funciona como um repositório de diferentes tipos de VNFs que estarão prontas para serem instanciadas pelo gerenciador e que são armazenadas em seu no banco de dados.

Para a descrição de VNFs, o Tacker utiliza uma linguagem padronizada pela *Organization for the Advancement of Structure Information Standards* (OASIS) conhecida como TOSCA [Hung et al. 2017]. Esta linguagem é conhecida por ser uma linguagem de descrição de uso de recursos para o contexto em computação em nuvem, mas também sendo utilizada no modelo NFV [Garay et al. 2016]. Ela define, por meio de *scripts*, a criação de VNFs contendo as configurações básicas das máquinas virtuais, suas conexões de rede, podendo adicionar políticas de monitoramento e funcionalidades.

O modelo NFV traz inúmeras melhorias como a redução de custos para operadoras e agilidade no processo de para a entrega de soluções. O Tacker se encarrega dos desafios de entregar essas soluções em funcionalidades, tais como, gerencia do ciclo de vida, encadeamento e escalabilidade de VNFs, etc.

A escalabilidade de VNFs consiste em adicionar à VNF a capacidade de aumentar ou diminuir o número de recursos em uma aplicação com base em suas necessidades. Existem dois tipos de escalabilidade, horizontal e vertical. Na horizontal ocorre o incremento ou decremento do número de componentes da função de rede virtualizada, neste caso, em vez da função de rede estiver em uma máquina virtual de alto desempenho, ela estaria dividida em múltiplas maquinas virtuais de menor porte. Já na vertical, ocorre o incremento ou decremento dos recursos da função de rede virtualizada, ou seja, é modificada os seus recursos tais como, processador, memória e disco.

Dessa forma, pode-se utilizar a infraestrutura como defesa, utilizando a tecnologia NFV, proporcionando a funcionalidade de *auto scale*, para suportar ataques do tipo *high-rate*, juntamente com a defesa *SeVen*, responsável por mitigar ataques do tipo *low-rate*. Assim, a proposta do presente artigo visa criar uma contra medida para aplicativos, executados em nuvem, contra ataques combinados de negação de serviço *low-rate* e *high-rate*, realizados por dispositivos conectados.

4. Experimentos

4.1. Cenários de Avaliação

Para a elaboração dos cenários foi criada uma nuvem OpenStack, em sua versão estável Queens[8], em um servidor Dell PowerEdge T430, contando com 32 GB de memória RAM, HD de 2 TB e processador Intel Xeon E5-2609, com o sistema operacional Ubuntu Server 16.04.3 LTS Xenial Xerus. Para verificar a proposta deste trabalho, foi criado dois cenários: o primeiro visa verificar a defesa *SeVen* contra ataques de negação de serviço na

[8]https://releases.openstack.org/queens/index.html

camada de aplicação do tipo *high-rate*; o segundo cenário tem como intenção verificar a funcionalidade de *auto scale* como ferramenta para mitigar ataques do tipo *Get-Flooding*.

Em todos os cenários foram utilizados duas métricas para verificar a eficácia das propostas: a primeira é a disponibilidade dos clientes legítimos, ou seja, a quantidade de clientes que solicitaram a página ao servidor *web* e obtiveram sucesso. Essa métrica visa verificar o quanto de clientes legítimos foram impactados com o ataque; a segunda métrica é o tempo de serviço (TTS - *Time to Service*), que é o tempo, em segundos, em que o cliente leva para solicitar uma página *web*, o servidor receber a requisição, processar e a resposta retornar ao cliente. As métricas de disponibilidade e tempo de serviço são coletados por meio da ferramenta Siege[9], que oferece um relatório completo dos clientes legítimos.

Foram simulados, por meio da ferramenta Siege, total de 200 clientes legítimos simultâneos, durante 45 minutos, que é o tempo total de cada repetição dos experimentos. Cada cliente solicitava a página *web* de forma aleatória e repetitiva. Ou seja, quando um cliente solicita uma página e recebe a resposta, ele espera um tempo aleatório (que varia entre 0 e 4 segundos) para enviar outra solicitação.

4.1.1. Cenário utilizando *SeVen*

Neste cenário, foi criado uma máquina virtual dentro da nuvem OpenStack, contendo um servidor *web* e a defesa *SeVen* protegendo-o. Essa VM contém 1 vCPU, 2 GB de memória e 10 GB de HD, com o sistema operacional Ubuntu Server 16.04.3 LTS Xenial Xerus. Além disso, foi criado duas máquinas virtuais, cada uma com 1 vCPU, 1 GB de memória e 10 GB de HD, também com o sistema operacional Ubuntu Server 16.04.3 LTS Xenial Xerus. A primeira VM é utilizada para realizar um ataque de negação de serviço, simulando usuários maliciosos. O ataque é realizado através da ferramenta GoldenEye[10], que realiza um ataque do tipo *high-rate*, o *Get-Flooding*. Dessa forma, o tráfego criado simula tráfego oriundo de dispositivos conectados, assemelhando a requisições quando há ataque de equipamentos de IoT. A segunda máquina virtual é utilizada para simular clientes legítimos realizando requisições ao servidor *web*. Os clientes são simulados por meio da ferramenta Siege.

4.1.2. Cenário utilizando *auto scale*

Para verificar a proposta deste trabalho, a de utilizar a infraestrutura como defesa, foi criado o cenário, como apresentado na Figura 1, em que foi instanciado uma VNF contendo a funcionalidade de *auto scale*, inicialmente com uma máquina virtual, com 1 vCPU, 1 GB de memória e 10 GB de HD, tendo o sistema operacional Ubuntu Server 16.04.3 LTS Xenial Xerus, o serviço *web* Apache2 e a defesa *SeVen* [Dantas et al. 2014]. A proposta de se utilizar o *SeVen* é a possibilidade de se criar uma defesa contra *low-rate* e *high-rate*, dessa forma, quando há um ataque *slowloris*[11], o *SeVen* consegue gerar disponibilidade sem a necessidade de escalar outro servidor *web*. E quando há um ataque *Get-Flooding*,

[9]https://www.joedog.org/siege-home/

[10]https://wroot.org/projects/goldeneye/

[11]https://web.archive.org/web/20150426090206/http://ha.ckers.org/slowloris

a infraestrutura entra em ação, oferecendo a disponibilidade para clientes legítimos.

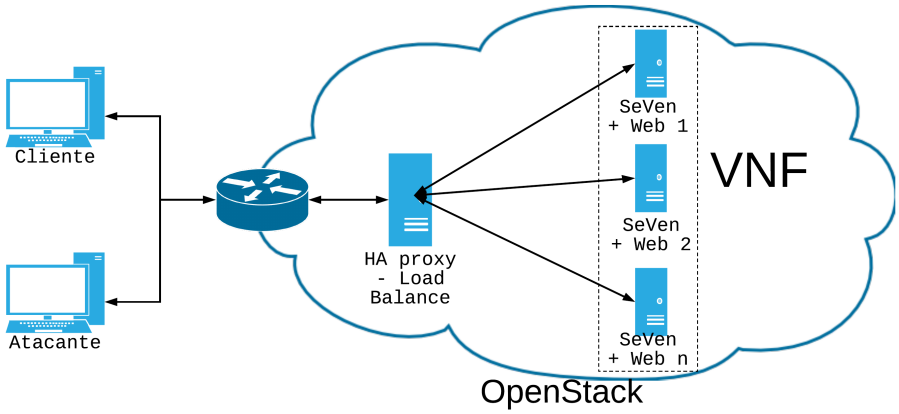

Figura 1. Cenário de experimento com *auto scale*

Da mesma forma que no cenário descrito na Seção 4.1.1, foram criadas duas máquinas virtuais, com 1 vCPU, 1 GB de memória e 10 GB de HD, com o sistema operacional Ubuntu Server 16.04.3 LTS Xenial Xerus. Uma VM para realizar o tráfego malicioso com a ferramenta GoldenEye, simulando tráfego de dispositivos de IoT, por exemplo. E a segunda simulando o tráfego de clientes legítimo, utilizando a ferramenta Siege. Por fim, foi criado também uma terceira máquina virtual, situada a frente do servidor *web*, com as mesmas configurações das VMs anteriores, realizando a função de balanceador de carga entre as instâncias do serviço *web*. Para essa função foi utilizado o HA Proxy[12], com a opção de balanceador *Round Robin*, realizando o encaminhamento das requisições na camada de aplicação.

A medida que clientes requisitam a página *web* e atacantes injetam tráfego malicioso, a nuvem OpenStack, continuamente, verifica a a taxa de utilização do processador da máquina virtual que contém o serviço *web*. A medida que a taxa de CPU ultrapassa o limiar de 70%, a nuvem OpenStack instancia outra VM, com as mesmas configurações e serviço. Dessa forma, a VM HA *Proxy* começa a balancear a carga, enviando as requisições para essa nova instância.

Para verificar o real uso da CPU em cada instância foi utilizado o comando `sar`[13], que é usado para verificar o uso da utilização de CPU em intervalos de 1 segundo. Assim, pode-se verificar o momento exato e a taxa de CPU que estava sendo utilizada quando ocorrer a instanciação de outro servidor *web*.

4.2. Resultados

4.2.1. Resultados da defesa *SeVen* em ataques do tipo *High-Rate*

A defesa *SeVen* foi proposta para ataques de negação de serviço na camada de aplicação, do tipo *low-rate*, ou seja, que tem pouca taxa de envios de requisições, por exemplo o ataque *Slowloris*. De acordo com Dantas [Dantas 2015], sem a defesa *SeVen*, a disponibilidade era de 0% e o tempo de serviço infinito, enquanto que com a estratégia, foi

[12]http://www.haproxy.org/
[13]http://sebastien.godard.pagesperso-orange.fr/

Tabela 1. Resultados dos experimentos utilizando *SeVen* com ataques do tipo *Low-Rate* e *High-Rate*

	Disponibilidade (%)	Tempo de Serviço (s)
SeVen em ataques Low-Rate	95	0,06
SeVen em ataques High-Rate	13,6	8,4

verificado uma disponibilidade de 95% dos clientes em um servidor *web* sofrendo ataque *Slowloris*, e um tempo de serviço de 0,06 segundo. Esses valores estão apresentados na linha correspondente da Tabela 1.

A partir desses resultados, foram realizados experimentos para verificar a eficiência da defesa *SeVen* em ataques do tipo *high-rate*, por exemplo, o *Get-Flooding*. Ou seja, no cenário construído para o presente trabalho, foi inserido apenas a defesa *SeVen* e realizado o ataque. Foram coletados os dados acerca da disponibilidade e de tempo de serviço.

A Tabela 1, na última linha, mostra o resultado, em média, da disponibilidade e do tempo de serviço, obtidos nesses experimentos. A defesa *SeVen* só conseguiu prover disponibilidade para 13,6% dos clientes legítimos, durante um ataque de negação de serviço, do tipo *high-rate*. Além disso, *SeVen* tem memória limitada e poder de processamento também limitado, e não consegue gerenciar a maioria das requisições destinadas ao servidor *web*, pois, como foi dito anteriormente, os ataques *high-rate* tem como característica enviar várias requisições.

Dentre os 13,6% dos clientes atendidos, o tempo de serviço, em média, foi de 8,4 segundos, de acordo com a Tabela 1. Ou seja, o processo do cliente realizar uma requisição, o servidor *web* receber, responder e a resposta chegar ao cliente, demorou, em média, pouco mais de 8 segundos. Um tempo relativamente alto para um cliente esperar por uma solicitação.

Dessa forma, não é recomendado utilizar a estratégia *SeVen* em ataques *high-rate*, pois, não se consegue prover disponibilidade razoável para os clientes legítimos. Por outro lado, quando se trata de ataques *low-rate*, *SeVen* é extremamente eficiente, conseguindo que quase a totalidade dos clientes sejam atendidos.

4.2.2. Resultados dos experimentos realizando a estratégia de *auto scale*

Na Figura 2 tem-se o gráfico da taxa de utilização da CPU, descrita em porcentagem, no eixo y, enquanto que no eixo x está o tempo, em segundos, do experimento. No gráfico estão representados a taxa média de utilização da CPU de cada servidor *web*. Pode-se perceber que há uma variação na taxa de utilização justamente quando há a alocação de um novo servidor *web*. Ou seja, a máquina 'Seven-Web1', que inicia o experimento, começa com média de 50% da CPU. Quando há um aumento na quantidade do tráfego malicioso, a taxa sobe para quase 90%. O OpenStack, verificando esse aumento, providencia uma outra instância, chamada 'Seven-Web2'. Ao momento que essa máquina entra em atividade, a média de porcentagem dos dois servidores *web* voltam para a casa de 55%.

Novamente, o atacante aumenta a quantidade de tráfego, fazendo com que a taxa de utilização da CPU ultrapasse o limiar de 70%. Mais uma vez a nuvem verifica essa alta

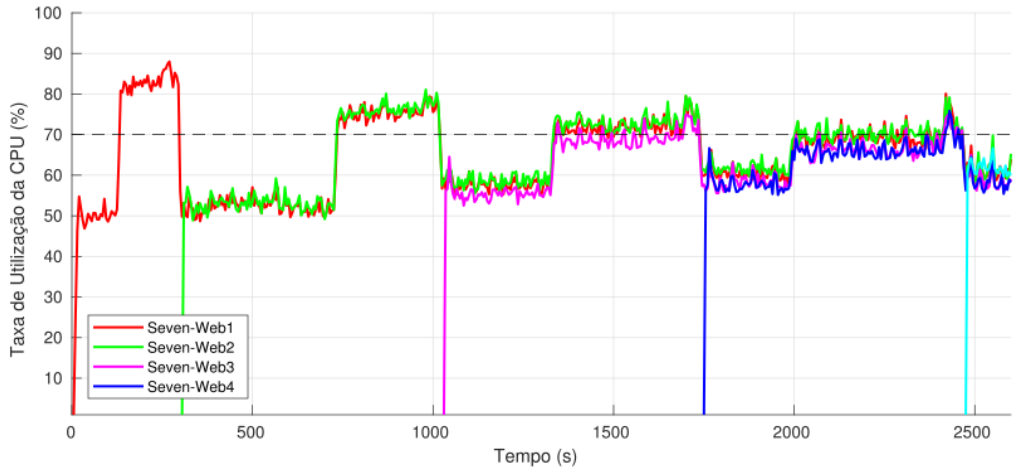

Figura 2. Taxa de utilização da CPU durante os experimentos

utilização e realiza a instanciação de um novo servidor. Durante todo o experimento esse comportamento se repete, mas em nenhum momento verifica-se uma taxa de utilização próximo a 100%, o que levaria, neste caso, a uma indisponibilidade de clientes legítimos.

A Figura 3 apresenta a média da disponibilidade, em porcentagem, dos clientes legítimos que realizaram a solicitação da página ao servidor *web* e foram atendidos. São apresentados os três cenários: sem nenhuma defesa, sendo a barra com a cor verde; apenas a defesa *SeVen*, representado pela barra de cor azul, sendo o mesmo resultado apresentado na Seção 4.2.1; e por fim, a barra vermelha representando os resultados obtidos com utilizando a proposta deste trabalho.

No cenário sem a defesa, apenas 17,5% dos clientes legítimos, em média, conseguiram solicitar e receber a página *web*. Esse resultado demonstra que o ataque de negação de serviço, o *Get-Flooding*, está sendo efetivo, ou seja, consegue deixar mais de 80% dos clientes sem acessar a página requerida. Dessa forma, verificando que o ataque está sendo efetivo, pode-se realizar os experimentos nos outros cenários. No cenário com a defesa *SeVen*, a disponibilidade foi de 13,6%, um pouco abaixo do que a disponibilidade verificada no cenário sem a defesa. Essa piora nos resultados deve-se ao processamento a mais inserido com a estratégia *SeVen*, pois a defesa não consegue lidar com a quantidade de requisições que estão chegando ao servidor *web*.

Por fim, ainda na Figura 3, no cenário com a proposta deste trabalho, utilizando a infraestrutura disponibilizada para conseguir aguentar a enxurrada de requisições, conseguiu prover uma disponibilidade de 100%. Ou seja, dos clientes que solicitaram uma página *web*, todos conseguiram ser atendidos, em um cenário em que está ocorrendo um ataque de negação de serviço. Esse resultado mostra que, para o cenário proposto, a estratégia de aumentar a quantidade de atendimento, de acordo com a demanda e de forma automática, se fez efetiva para o ataque *Get-Flooding*, conseguindo, suportar o ataque, e não simplesmente mitiga-lo.

Na Figura 4 é apresentado o tempo de serviço (TTS - *Time to Service*), em segundos, nos três cenários: verde para o cenário sem a defesa; azul com a defesa *SeVen*; e vermelho para o cenário com a utilização do *auto scale*. No cenário sem qualquer tipo

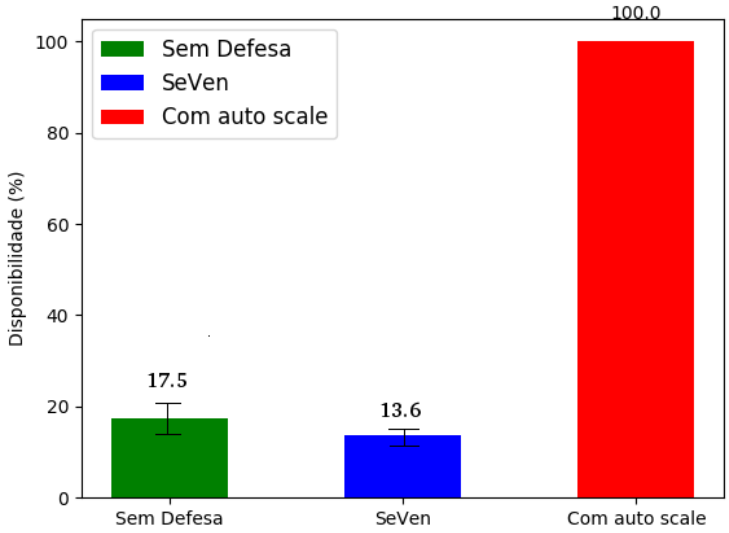

Figura 3. Porcentagem de clientes legítimos que foram atendidos nos cenários.

de defesa, foi constatado que, dos clientes que conseguiram a página *web*, esperaram, em média, 14,5 segundos para receber a resposta do servidor. Mais uma vez, esse resultado demonstra a efetividade do ataque neste cenário, pois 14,5 segundos é um tempo alto para clientes esperarem por uma página *web*. Nos experimentos em que foi utilizado a defesa *SeVen*, foi verificado um tempo de serviço de 8,4 segundos, em média. Ou seja, por mais que a disponibilidade neste cenário seja pior do que o cenário sem a defesa, como verificado na Figura 3, o tempo de serviço diminuiu pela metade, indicando que a estratégia *SeVen* consegue diminuir o tempo que o cliente espera pela resposta do servidor *web*.

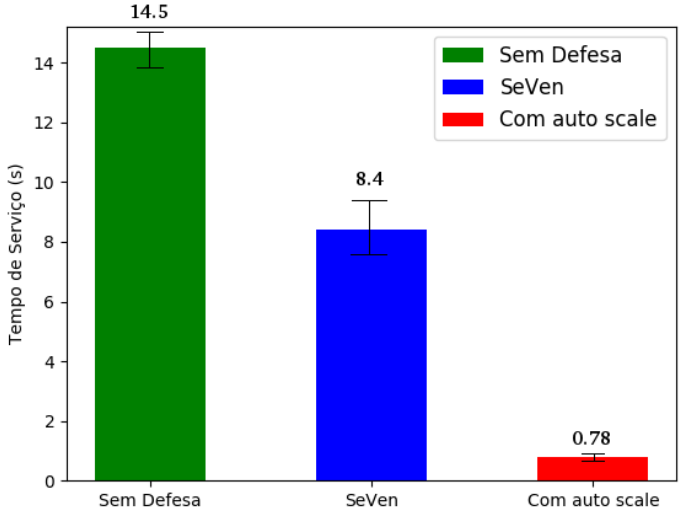

Figura 4. Tempo de serviço, em segundos, nos três cenários.

Por fim, na Figura 4, o tempo de serviço no último cenário, utilizando a estratégia da infraestrutura como defesa, obteve um tempo de serviço de 0,78 segundo, mostrando que utilizar várias instâncias de servidores, sendo alocados automaticamente de acordo com a necessidade, não influencia no tempo em que o cliente leva para solicitar a página, o servidor receber, processar e enviar novamente ao cliente.

Diante disso, verifica-se que, para o cenário apresentado, a estratégia de utilizar a infraestrutura para proporcionar mais opção de atendimento de forma dinâmica é eficaz. Ou seja, a infraestrutura se adapta a demanda das requisições. Há uma clara deficiência nessa estratégia, pois há um limite imposta pela própria infraestrutura física. Dessa forma, se ocorrer um ataque com um grande volume de tráfego, e a infraestrutura da nuvem não for grande o suficiente, continuará a acontecer a indisponibilidade, sendo que, neste caso, técnicas tradicionais de detecção e mitigação podem ser adicionadas para lidar com esses tipos de ataques.

5. Conclusão e Trabalhos Futuros

Ataques de negação de serviço, principalmente contra a camada de aplicação, tem como característica tráfego de ataque se assemelhar ao tráfego de clientes legítimos, devido a isso, são difíceis de serem detectados e consequentemente de serem mitigados. Ferramentas que realizam a defesa contra esses tipos de ataques utilizam estratégia de descartar tráfego, podendo assim atingir clientes legítimos.

O presente trabalho apresentou uma estratégia contra ataques de negação de serviço na camada de aplicação, particularmente o *Get-Flooding*, utilizando infraestrutura para gerar disponibilidade a clientes legítimos. Ou seja, a medida que uma determinada métrica, por exemplo taxa de CPU, alcançar um limiar a infraestrutura instancia outro servidor *web*, de forma automática, providenciando assim mais recursos para que clientes sejam atendidos. Para verificar essa estratégia foram realizados experimentos no ambiente de nuvem OpenStack, verificando a disponibilidade e o tempo de serviço nos cenários sem defesa e com a proposta deste trabalho. Verificou-se que a estratégia é eficaz nesse cenário, tendo vantagem sobre outras propostas de mitigação pois não corre risco de realizar descarte de clientes legítimos.

Como trabalhos futuros pretende-se utilizar outras métricas para ser o gatilho da escalabilidade, inclusive da própria aplicação por exemplo. Dessa forma, o *auto scale* pode ser acionado em um limiar mais próximo e real da aplicação. Por fim, propõe-se combinar a estratégia do *auto scale* com ferramentas tradicionais, inclusive utilizando o conceito de Encadeamento de funções de rede (SFC - *Service Function Chaining*), como verificado nos trabalhos relacionados.

6. Agradecimentos

Este trabalho recebeu financiamento do projeto GT-NosFVeraTO, por meio da RNP sob o contrato de no. RNP/FEST 002917, e do projeto FUTEBOL sob o contrato de no. MCTIC/RNP/FEST 688941. Além disso, os autores gostariam de agradecer o financiamento parcial proveniente de recursos de bolsas CNPq, CAPES e FAPES.

Referências

Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., et al. (2017). Understanding the mirai botnet. In *USENIX Security Symposium*.

Balobaid, A., Alawad, W., and Aljasim, H. (2016). A study on the impacts of dos and ddos attacks on cloud and mitigation techniques. In *2016 International Conference on Computing, Analytics and Security Trends (CAST)*, pages 416–421.

Chen, J., Chen, Y., Tsai, S.-C., and Lin, Y.-B. (2017). Implementing nfv system with openstack. In *Dependable and Secure Computing, 2017 IEEE Conference on*, pages 188–194. IEEE.

Dantas, Y. G. (2015). Estratégias para tratamento de ataques de negação de serviço na camada de aplicação em redes ip. Master Thesis.

Dantas, Y. G., Lemos, M. O. O., Fonseca, I., and Nigam, V. (2016). Formal specification and verification of a selective defense for TDoS attacks. In *11th WRLA*.

Dantas, Y. G., Nigam, V., and Fonseca, I. (2014). A selective defense for application layer ddos attacks. In *ISI-EISIC*. http://www.nigam.info/docs/ddos.pdf.

ETSI, N. (2013). Etsi gs nfv 002 v1. 1.1 network functions virtualization (nfv). *Architectural Framework. sl: ETSI*.

Garay, J., Matias, J., Unzilla, J., and Jacob, E. (2016). Service description in the nfv revolution: Trends, challenges and a way forward. *IEEE Communications Magazine*, 54(3):68–74.

Herrera, J. G. and Botero, J. F. (2016). Resource allocation in nfv: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532.

Herzberg, B., Bekerman, D., and Zeifman, I. (2016). *Breaking Down Mirai: An IoT DDoS Botnet Analysis*. `https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html` – Acessado: 16-03-2018.

Hung, Y.-M., Chien, S.-C., and Chunghwa, Y.-Y. H. (2017). Orchestration of nfv virtual applications based on tosca data models. In *Network Operations and Management Symposium (APNOMS), 2017 19th Asia-Pacific*, pages 219–222. IEEE.

Lai, S. F., Su, H. K., Hsiao, W. H., and Chen, K. J. (2016). Design and implementation of cloud security defense system with software defined networking technologies. In *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 292–297.

Lemos, M. O. O., Dantas, Y. G., Fonseca, I., Nigam, V., and Sampaio, G. (2016). A selective defense for mitigating coordinated call attacks. In *34th Brazilian Symposium on Computer Networks and Distributed Systems (SBRC)*.

Mauricio, L. A., Alvarenga, I. D., Rubinstein, M. G., and Duarte, O. C. M. (2017). Uma arquitetura de virtualização de funções de rede para proteção automática e eficiente contra ataques.

Shea, R. and Liu, J. (2012). Understanding the impact of denial of service attacks on virtual machines. In *Quality of Service (IWQoS), 2012 IEEE 20th International Workshop on*, pages 1–9.

Tang, P., Li, F., Zhou, W., Hu, W., and Yang, L. (2015). Efficient auto-scaling approach in the telco cloud using self-learning algorithm. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6.

Zargar, S. T., Joshi, J., and Tipper, D. (2013). A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys and Tutorials*, 15(4):2046–2069.