Uma proposta de arquitetura de segurança para a detecção e reação a ameaças em redes SDN

Maxli Barroso Campos[1]
Joberto Martins[1]

Resumo: As Redes Definidas por Software (*Software Defined Networking* – SDN) desacoplam o controle do encaminhamento de dados, oferecendo alta programabilidade e uma visão global da rede. A adoção dessa abordagem é crescente em redes empresariais, centros de dados e outras infraestruturas críticas de rede. No entanto, constitui-se um desafio não só prover segurança nessas redes de nova geração como também permitir que um ataque à rede seja suscetível a um procedimento de tratamento de incidentes e perícia forense. Desta forma, esse artigo propõe uma arquitetura de rede explorando os novos paradigmas das redes SDN com mecanismos implementados de detecção e reação a ameaças de segurança, capaz de fornecer recursos para realização de análises de intrusão e de ataques. De maneira geral, a arquitetura de segurança de rede proposta implementa mecanismos de proteção usando técnicas de *hardening* e detecção e reação a ameaças de segurança usando sistemas de detecção de intrusão por meio de *Intrusion Detecction System* - IDS, explorando as novas abordagens empregadas pela tecnologia das redes SDN e fornecendo recursos para realização de análise de intrusão e de ataques.

Palavras-chave: **Ameaças de segurança, OpenFlow, Redes Definidas por Software**

Abstract: *The Software Defined Networking (SDN) decouples control and routing of data, offering high programmability and a global view of the network. The adoption of this approach is growing in enterprise networks, data centers and other critical network infrastructures. However, it is still a challenge not only provide security in these next generation networks as well as to allow a network attack to be susceptible of an incident and forensic treatment procedure. In this context, this paper proposes a network architecture exploring the new SDN paradigm by creating a detection and protection mechanism to deal with security threats. In general, the network security architecture proposed implements a protection mechanism using hardening and detection techniques that reacts to security threats using an Intrusion Detection System - IDS, exploring new empowered by the SDN networks and providing resources to perform intrusion analysis and attacks.*

Keywords: *Security, Software Defined Networking (SDN), OpenFlow, Security Threat*

1 Introdução e motivação

As redes atuais de computadores não atendem requisitos mais recentes que envolvem a operação de protocolos diversificados, crescimento das tabelas de roteamento, suporte à mobilidade dos usuários, implementação de novos mecanismos de segurança e ainda apresentam novas demandas que precisam ser pensadas. No âmbito da pesquisa, essas exigências são ainda maiores pois requerem experimentações de novos protocolos e funcionalidades da rede em condições reais, idealmente em ambientes distribuídos [1]. Segundo [2] a ausência de flexibilidade no controle do funcionamento interno dos equipamentos assim como o alto custo da infraestrutura vigente são barreiras para a evolução das arquiteturas e para a inovação necessária na oferta de novos serviços e aplicações de rede. Para tentar contornar esse problema, a comunidade de redes de computadores tem investido em várias iniciativas que levem à implantação de uma infraestrutura de redes com

[1] Programa de Pós-Graduação em Sistemas e Computação, UNIFACS, Salvador, BA - Brasil
{joberto.martins@unifacs.br,maxli.campos@unifacs.br}

http://dx.doi.org/10.5335/rbca.v9i1.6595

maiores recursos, de forma que novas tecnologias e serviços possam ser inseridas de forma gradual na rede. Uma das iniciativas nesse sentido foi a estratégia de SDN, tendo como base uma de suas possibilidades de implantação a definição da interface e do protocolo *OpenFlow* [3].

Com o *OpenFlow*, os elementos de encaminhamento oferecem uma interface de programação simples que lhes permite estender o acesso e controle das tabelas de encaminhamento/roteamento utilizadas pelo hardware para determinar o próximo passo de cada pacote recebido. As redes SDN empregam tipicamente um controlador para instalar, sob demanda, regras de encaminhamento de pacotes por fluxo nos nós da rede [4]. Desta forma, faz-se uma separação das funções de controle do plano de dados, que executa o encaminhamento dos pacotes (informações). O controlador processa as mensagens de controle e, assim, reduz os requisitos de memória e de processamento dos nós comutadores. As redes SDN constituem um novo paradigma computacional de redes que propõe o desacoplamento entre o plano de dados (implementado em hardware especializado para suportar o desempenho requerido pelas redes atuais) e o plano de controle (executado em um ou mais controladores, os quais são responsáveis pela programação das ações de encaminhamento/roteamento realizadas pelo *hardware*). Contudo, esse novo paradigma apresenta algumas limitações quanto à segurança da rede, pois um componente com comportamento malicioso pode comprometer o funcionamento de toda a rede.

Logo, o desafio de prover segurança no contexto de redes SDN tornou-se um aspecto fundamental até mesmo para se estabelecer a confiança na tecnologia que é inovadora e disruptiva. Apesar das vantagens advindas da implementação das redes SDN, algumas das vulnerabilidades das redes tradicionais persistem neste novo ambiente, seja por empregarem os mesmos tipos de equipamentos de rede e servidores, seja pela natureza tipicamente centralizada do plano de controle [5], ou pelo fato de que grande parte das ferramentas e técnicas utilizadas para segurança da informação como antivírus e *firewall* não são suficientes para assegurar a segurança. Isso acaba por originar novos problemas exclusivos deste domínio. Neste sentido, a proposta deste artigo consiste em apresentar uma arquitetura de segurança de rede, visando tratar diferentes vulnerabilidades em redes SDN com suporte *OpenFlow*. A arquitetura de segurança de rede proposta implementa mecanismos de proteção usando técnicas de *hardening* e detecção e reação a ameaças de segurança usando tecnologia de detecção de intrusão por meio de IDS, explorando as novas abordagens empregadas pela tecnologia das redes SDN e fornecendo recursos para realização de análise de intrusão e de ataques.

2 Trabalhos relacionados

Shim et al. propõem um arcabouço de segurança para o controlador NOX *OpenFlow* (sistema FRESCO) [8] que provê uma linguagem de programação e módulos de *software* para desenvolver aplicações de segurança. Esses módulos permitem a integração de serviços de segurança sobre uma nova arquitetura de controladores. O uso do *OpenFlow* na área de segurança de rede relacionado à análise e prevenção de intrusão já foi abordado e apresentou bons resultados em pesquisas anteriores, como, por exemplo, nas pesquisas de Ballard et al. (2010), Mattos (2013), Porras (2012), Xing (2013), Mehdi et al. (2011) e Nagahama (2012). Xing et al. (2013) apresentam o SnortFlow, uma proposta de sistema de prevenção de intrusão (*Intrusion Prevention System* - IPS) em um ambiente de nuvem baseado na solução com XEN, utilizando *switches OpenFlow* para auxiliar na captura de tráfego. Embora atividades de contramedida tenham sido apresentadas para a reconfiguração da rede, estas não foram avaliadas. Os autores do trabalho elaboraram um protótipo no qual o agente SnortFlow foi instalado nos domínios Dom 0 e Dom U. A avaliação do SnortFlow foi focada no desempenho, visto que os autores não detalharam as características do tráfego que estava sendo detectado pelo Snort e nem a análise que estava sendo feito sobre ele.

Em relação aos trabalhos relacionados, observa-se também que uma parte dos trabalhos contempla aspectos gerais de proteção em redes SDN sem uma demonstração aplicada nos equipamentos de rede e utilizando ferramentas de emulação como Mininet para demonstração de resultados. Desta forma a proposta deste trabalho visa extender os trabalhos relacionados por meio de uma proposta de uma arquitetura de rede *OpenFlow*, com tráfego de rede e equipamentos reais, dando subsídios ao pesquisador para apresentação, demonstração e validação da pesquisa e que permita realizer: a captura do tráfego de rede, encaminhamento do tráfego para análise em um IDS e realizar ações que permitam a reconfiguração automática da tabela de fluxos do s*witch OpenFlow* quando identificado alguma ação maliciosa na rede.

3 SDN/OpenFlow e segurança

Um dos protocolos amplamente usados na implantação das redes SDN é o *OpenFlow*, um protocolo aberto e padronizado para comunicação entre controladores e equipamentos de comutação, tais como *switches,* roteadores, pontos de acesso sem fio e o isolamento de recursos entre usuários [14]. Esse protocolo define formas para que *switches* que o suportem sejam programados por um *software* controlador externo, recebendo comandos para cada tipo de fluxo de pacotes observados e armazenando essas regras em suas tabelas internas. De uma forma geral, todos os equipamentos *OpenFlow*, utilizam uma tabela interna para encaminhar os pacotes. O *OpenFlow* define um protocolo-padrão para determinar as ações de encaminhamento de pacotes em dispositivos de rede, como, por exemplo, comutadores, roteadores e pontos de acesso sem fio. As regras e ações instaladas no hardware de rede são responsabilidade de um elemento externo, denominado controlador, que pode ser implementado em um servidor comum. A principal abstração utilizada na especificação *OpenFlow* é o conceito de fluxo, que é constituído pela combinação de campos do cabeçalho do pacote a ser processado pelo dispositivo.

O controlador da rede é o software responsável por tomar decisões e adicionar e remover as entradas na tabela de fluxos, de acordo com o objetivo de uma camada de abstração da infraestrutura física, facilitando a criação de aplicações e serviços que gerenciem as entradas de fluxos na rede. Esse modelo assemelha-se a outros sistemas de software que proveem abstração do *hardware* e funcionalidade reutilizável. Dessa forma, o controlador *OpenFlow* atua como um sistema operacional para gerenciamento e controle das redes, e oferece uma plataforma com base na reutilização de componentes e na definição de níveis de abstração (comandos da Interface de Programação de Aplicação – *Application Programming Interface* - API). Desta forma, novas aplicações de rede podem ser desenvolvidas rapidamente [17]. A programabilidade do controlador permite a evolução em paralelo das tecnologias nos planos de dados e as inovações na lógica das aplicações de controle.

Embora a SDN forneça muitos benefícios em comparação às redes tradicionais, algumas das vulnerabilidades das redes tradicionais persistem na SDN, originando novos problemas exclusivos deste domínio. Em particular, as mesmas características da SDN que são desejáveis (gestão e configuração centralizada, por exemplo) se tornam alvos de ataques, pois comprometer o controlador ou obter controle sobre o mesmo pode ter um impacto significativo na rede de controle. O atacante poderia, por exemplo, tentar algumas formas de ataque de consumo de recursos no controlador para inundá-lo com solicitações e forçá-lo a responder lentamente a eventos do tipo *Packet_In* e provocando uma demora no envio das mensagens *Packet_Out*. Além disso, tendo em vista que muitos controladores SDN executam alguma forma de sistema operacional Linux, executando assim um sistema operacional de propósito geral, as mesmas vulnerabilidades daquele sistema irão se tornar vulnerabilidades para o controlador, tornando necessária algumas ações no sentido de se aplicar as melhores práticas de segurança da informação nos componentes da arquitetura de rede *OpenFlow* por meio de técnicas de *hardening*.

O fato de que hoje grande parte das ferramentas e técnicas utilizadas para segurança da informação com a finalidade de combater ataques, tais como *firewall*, sistemas de criptografia, *hash*, assinatura digital, antivírus, dentro outros, não são suficientes para assegurar a segurança de redes e sistemas torna crítico a implementação de mecanismos para lidar com os incidentes de segurança de forma automatizada nesta nova arquitetura de rede, levando em consideração os novos paradigmas das redes *OpenFlow*. Segundo o Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (CERT.br), só em 2015 foram quase 722.205 incidentes reportados. Desta forma, implementar IDSs e sistemas de prevenção de intrusão (*Intrusion Prevention System* – IPS) acaba sendo essencial e necessário nos dias atuais para garantir a aplicação e acompanhamento das políticas de segurança da rede. Um IPS é um NIDS que, além de capturar, analisar e reportar a ocorrência de um tráfego malicioso na rede, tenta bloquear ou mesmo minimizar os impactos causados por um usuário maliciosos.

4 Arquitetura de segurança de rede SDN para detecção e reação a ameaças

Esta seção descreve os aspectos relacionados ao desenvolvimento de um ambiente de rede baseado na plataforma *OpenFlow*, com mecanismos implementados de detecção e reação a ameaças de segurança voltados para uma rede SDN. A arquitetura apresentada na Figura 01 tem por objetivo a apresentação e identificação dos equipamentos e componentes necessários para validação da proposta de pesquisa e obtenção de dados em rede *OpenFlow*. A arquitetura proposta foi implementada se utilizando de servidores, estações de trabalho e um

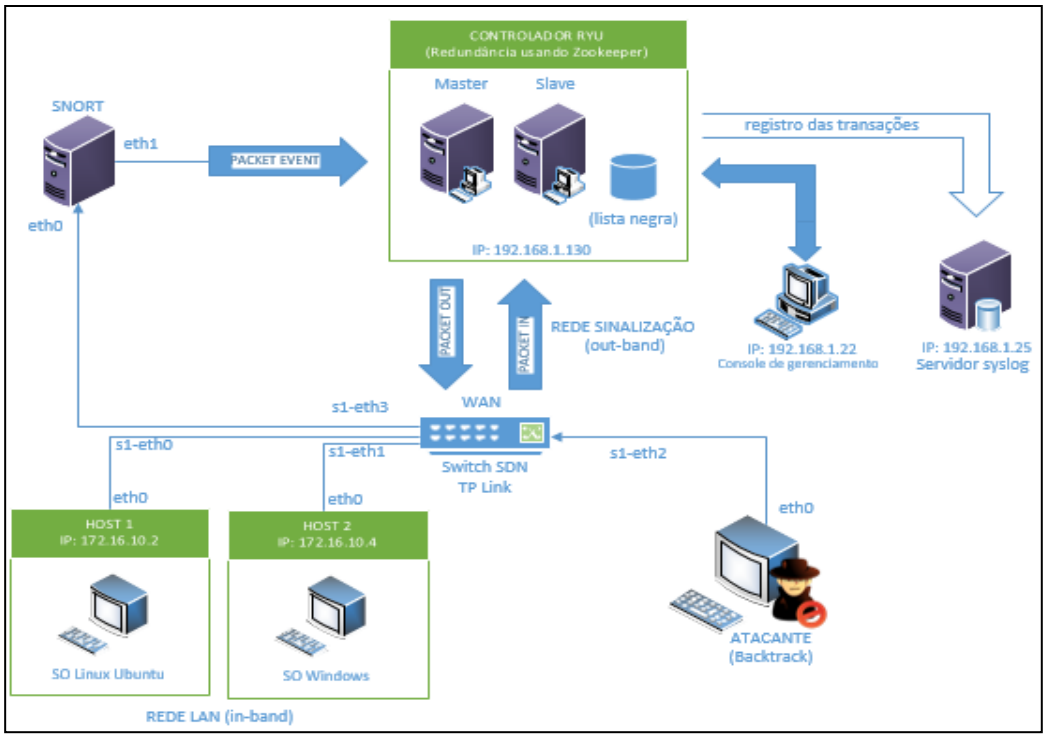
switch OpenFlow, visando se aproximar o mais perto possível de um ambiente que apresente os melhores resultados por meio de tráfego de rede real e não apenas simulado.

Para realização das simulações foi escolhido um equipamento real, mas especificamente o hardware TP-Link WR1043ND, que é um roteador sem fio homologado pela OpenWRT.org [19] para funcionar como um s*witch OpenFlow* de baixo custo. Na Tabela 1 estão descritas as especificações do equipamento:

Tabela 1. Especificações do roteador sem fio TP-Link

HARDWARE	DESCRIÇÃO
CPU	Atheros AR9132@400MHz
Memória RAM	32 MB
Interfaces de rede	4 interfaces LAN e WAN, além de interface wireless

Figura 01: Arquitetura de Segurança da Rede OpenFlow



Para o suporte ao protocolo *OpenFlow* 1.3 o equipamento teve seu *firmware* alterado e recebeu uma versão de sistema operacional OpenWrt[2] compilada a partir da versão "trunk" do OpenWRT (*Barrier Breaker*) [20]. O fato de ser implementado em código aberto permite que seja possível modificar, melhorar ou substituir algumas de suas funcionalidades, de acordo com as necessidades dos usuários. Uma vez implementado, o modelo é usado para executar um conjunto de provas de funcionamento e medidas de rendimento com a ideia de dispor de um ambiente operacional de experimentação (*testbed)*.

4.1 Modelo da solução implementada no plano de controle

SDN é uma nova arquitetura de rede que simplifica o gerenciamento da rede por meio da abstração do plano de controle e do plano de encaminhamento de dados. Como plataforma para codificação de aplicações no plano de controle este trabalho emprega o Ryu, que é um dos controladores disponíveis no mercado, desenvolvido pelo Centro de Inovação em software da empresa japonesa NTT sob a licença Apache 2.0 [Ryu

[2] OpenWrt é uma distribuição GNU/Linux, de código aberto, sob licença GPL, voltados para sistemas embarcados, tradicionalmente usada em roteadores sem fio [16].

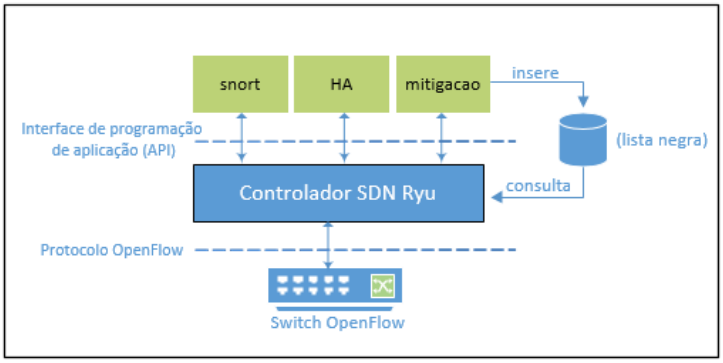
2014]. O projeto Ryu é um arcabouço baseado em componentes para programação de redes definidas por software, que emprega a linguagem de programação Python e permite programar aplicações utilizando vários protocolos, entre os quais se destacam no contexto deste trabalho os protocolos *OpenFlow* 1.0, 1.2, 1.3 e 1.4.

Existe um grande número de controladores SDN que empregam o protocolo *OpenFlow* para a configuração do plano de dados. Logo, antes de se chegar a esta definição e tendo em vista os resultados definidos no trabalho e dos recursos disponíveis do projeto, foi necessária uma fase prévia de investigação e levantamento de qual controlador seria mais adequado para solução. Serviram de apoio à decisão o curso online sobre SDN disponibilizado no Coursera [21], artigos na internet e os documentos disponíveis na página oficial do projeto [18].

Os principais aspectos fundamentais para definição do controlador Ryu como solução adotada neste artigo foram: suporte ao protocolo *OpenFlow* 1.3, haja vista que a versão do protocolo *OpenFlow* 1.0 está praticamente obsoleta e conta com campos de cabeçalho mais restritos para identificação de assinaturas de ataque, a linguagem de programação Python pela facilidade de programação e assimilação da linguagem, curva de aprendizagem, já que o fato do esforço em assimilação dos detalhes do controlador demanda um curto espaço de tempo, o aspecto de segurança, pelo fato do controlador Ryu apresentar uma aplicação (*app*) que já está disponível no seu arcabouço e que possibilita o suporte e integração nativa com a ferramenta Snort e boa documentação, suporte e comunidade de usuários. A despeito das soluções SDN serem recentes, o fato do projeto Ryu contar com ampla documentação, de projeto e arcabouço e disponibilidade de fóruns no site do projeto também foi um fator determinante da escolha deste controlador.

A plataforma que dá suporte ao controlador Ryu consiste em um executável chamado "/bin/ryu-manager" que é responsável pelo carregamento das diferentes aplicações que são chamadas e todas herdam da classe "ryu.base.app_manager.RyuApp". Quando executado, ele passa a escutar em um endereço de rede específico (Ex: 0.0.0.0) e em uma porta específica (6633 por padrão), onde qualquer *switch* (*hardware*, OpenVSwtich ou OVS) pode se conectar. A lógica implementada pelo programador (como os que foram desenvolvidos neste trabalho) é uma aplicação que trata esse processo. Embora seja possível executar vários aplicativos ao mesmo tempo, apenas uma instância por aplicação é suportada no Ryu. A linguagem de programação utilizada é o Python com suporte a suas várias versões, sendo que a versão 2.7 foi a empregada no contexto deste trabalho. Por sua vez, estas aplicações se comunicam uma como a outra por meio de geração de eventos, por meio de objetos da classe "ryu.controller.event.EventBase". Cada aplicação tem sua própria fila FIFO de eventos. Ao iniciar a aplicação, o Ryu cria automaticamente um segmento de processamento em forma de um ciclo dedicado a atender aos eventos na fila e por ser uma fila única deve ser dado atenção especial para que as funções que tratam os eventos não sejam não bloqueantes[3]. No contexto deste trabalho, como apresentado na Figura 02, foram implementadas no topo do controlador Ryu mais três aplicações.

Figura 02: Blocos de construção do código do Ryu



A primeira delas é o Snort (*switch_snort.py*) que é o código principal que dá suporte a um Switch L2 e responsável por possibilitar que o tráfego do *switch OpenFlow* redirecione, em modo promíscuo, todo o tráfego de uma de suas portas, a configurar, para a ferramenta do Snort. A segunda aplicação chamada HA (*ha.py*) faz referência a alta disponibilidade (*high availability* - HA) e tem por objetivo o emprego de controladores

[3] Implica que o fluxo de execução continua após ter chamado a rotina de envio, sem se importar se a mensagem chegou ao *buffer* da fonte ou do destino.

redundantes, visando dar alta disponibilidade ao plano de controle da rede *OpenFlow* por meio de uma ferramenta denominada Zookeper. E a terceira aplicação, denominada mitigação (*mitigacao.py*) tem por finalidade tratar as mensagens *Packet_event* enviadas pela ferramenta Snort ao controlador Ryu quando é reportado algum evento associado a uma assinatura de ataque na rede *OpenFlow*. Estas três aplicações visam fornecer no plano de controle os recursos necessários para se implementar os mecanismos de proteção, detecção e reação que serão apresentados em detalhes nos próximos subcapítulos.

5 Avaliação da arquitetura e validação

A validação do ambiente foi realizada por meio de testes conduzidos na arquitetura proposta e direcionados aos clientes da rede SDN e ao próprio s*witch OpenFlow*, ressaltando que neste ambiente, uma parte substancial do tráfego capturado terá origem ilícita ou maliciosa, ou seja, estará comprometida por códigos maliciosos. Logo, o objetivo deste capítulo consiste em fazer uma descrição sucinta das estratégias adotadas e implementadas na arquitetura, apresentar a validação da própria arquitetura e a metodologia usada para validação dos mecanismos de detecção e reação por meio das seguintes fases:

a. FASE 1: validação dos mecanismos de detecção;
b. FASE 2: validação dos mecanismos de reação.

5.1 Descritivo da arquitetura de rede segura proposta

Toda a estrutura do projeto foi implementada utilizando *hardware* dedicado para garantir o melhor desempenho na avaliação e validação da arquitetura de segurança proposta. Como apresentado na Figura 01, o ambiente possui três redes distintas: a rede SDN 172.16.10.0/24 em que estão conectados os nós da rede *OpenFlow*; a rede dos servidores em alta disponibilidade que estão rodando a aplicação do controlador Ryu da rede OpenFlow no IP 192.1681.130/24; e a rede de gerência 192.168.2.0/24 para gerência do controlador e hospedagem do servidor *syslog* que registra todas as ações do administrador da rede. O controlador Ryu (IP 192.168.1.130), contando com alta disponibilidade, foi programado para dar suporte a um *switch* L2 com recursos de integração com a ferramenta do Snort e mecanismos de reação empregando a flexibilidade do controlador de ser programável. A máquina do Snort tem a interface *eth0* configurada em modo de espelhamento na porta 3 do *switch OpenFlow*, e *eth1* configurada no IP 192.168.1.120 com o objetivo de enviar pacotes do tipo *Packet_event* ao controlador Ryu quando for identificado um tráfego malicioso na rede.

Para os testes foi empregado uma máquina real, como sendo um dos clientes da rede, com a distribuição Linux Kali (IP 172.16.10.251) conectado a porta 4 do *switch OpenFlow* para simular alguns ataques. Para demonstrar os mecanismos de detecção e reação a ameaças de segurança está sendo empregado na arquitetura proposta um switch comercial de prateleira (*commercial off-the-shelf* - COTS) que foi modificado [19] com o objetivo de explorar a capacidade de equipamentos existente no mercado e que podem ser usados para prototipação, com desempenho compatível e baixo custo. Para a geração de tráfego foram utilizadas as ferramentas t50[4] e Nmap[5] para Linux e para implementação do mecanismo de detecção de intrusão foi empregado a ferramenta Snort. Para viabilizar o armazenamento dos dados gerados pela ferramenta Snort foi utilizado o gerenciador de banco de dados MySQL. E para aumentar a eficiência na gravação dos registros dos possíveis ataques, foi utilizado a ferramenta *Barnyard* que é um pós-processador dos registros gerados pelo Snort no formato *UNIFIED*, permitindo que o Snort foque somente na análise dos pacotes gerados pelo tráfego na rede, reduzindo a carga do sistema e evitando a perda destes pacotes.

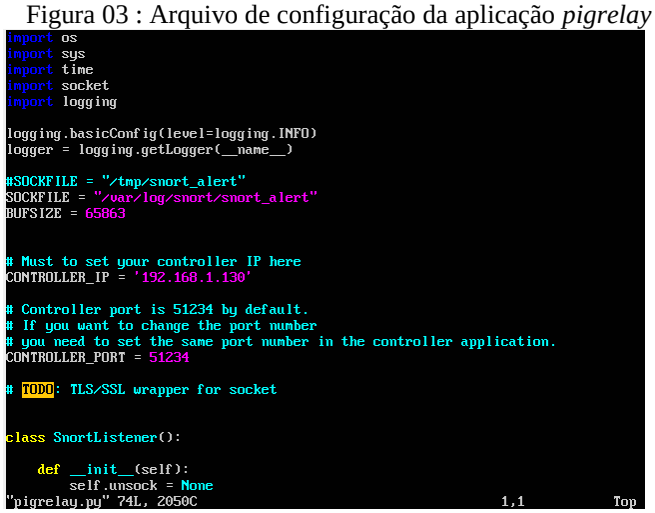
Os mecanismos de reação foram implementados por meio de codificação no arquivo *mitigacao.py* no controlador e que trata os eventos *Packet_event* gerados pela ferramenta Snort e que são enviados para o controlador Ryu por meio de uma aplicação desenvolvida em python chamada *pigrelay*, disponível em [23]. Esta

[4] A ferramenta T50 é um *packet injector* (injeção de pacote) livre, criado pelo brasileiro Nelson Brito, capaz de fazer ataques DoS e DDoS usando o conceito de teste de estresse.

[5] Ferramenta de código aberto para exploração de rede e auditoria de segurança, desenhada para escanear rapidamente amplas redes.

aplicação em execução na máquina do Snort foi configurada com os parâmetros apresentados na Figura 03, para se adequar a arquitetura de rede proposta, com a finalidade de possibilitar que as mensagens de alerta gerados sejam encaminhadas para o controlador por meio de soquetes de rede por meio de um *Unix Domain Socket.*

Figura 03 : Arquivo de configuração da aplicação *pigrelay*

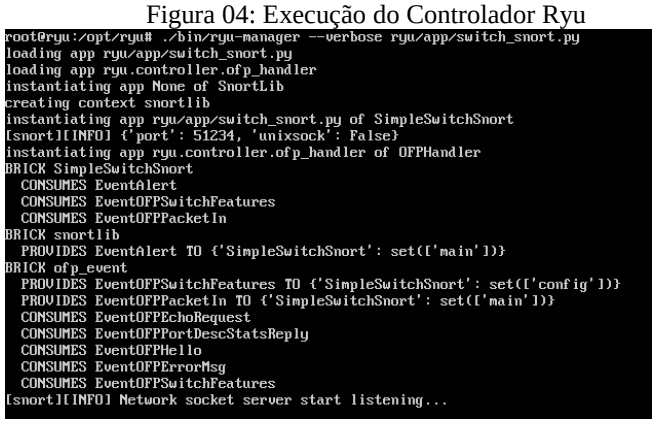


Em linhas gerais a ferramenta *pigrelay* ao ser executada envia para o controlador no IP 192.168.1.130 e porta 51234 os eventos de segurança que estão sendo gerados pela ferramenta do Snort e armazenados em /var/log/snort/snort_alert. E como ferramenta de análise para os alertas gerados pelo Snort está disponível na arquitetura proposta a ferramenta *Basic Analysis and Security Engine* (BASE) que foi instalada na mesma máquina do Snort.

5.2 Validação da arquitetura

O resultado do processo de validação da arquitetura envolve a criação de um identificador de caminho de dados (*datapath identifier)* da rede *OpenFlow* no *switch*. Este processo envolve a correta execução do serviço *openflow* do *switch* TP-Link e o estabelecimento de conexão com o plano de controle da rede que está rodando em alta disponibilidade nas máquinas *master/slave* do controlador da rede SDN. No contexto do plano de controle é empregado o controlador Ryu em execução no equipamento servidor no IP 192.168.1.130/24. A Figura 04 apresenta o comando *ryu-manager* sendo executado em modo *verbose* e executando o arquivo *switch_snort.py,* implementado em *python,* que também executa as outras aplicações (*ha.py* e *mitigacao.py*) que foram desenvolvidas no contexto deste projeto e apresentadas em seções anteriores deste trabalho.

Figura 04: Execução do Controlador Ryu



Como demonstrado na Figura 04, ao final da execução do comando, o controlador Ryu fica aguardando a conexão do *switch OpenFlow* autorizado. Neste momento ao se executar o serviço *openflow* no *switch* TP-Link é estabelecido uma conexão com o controlador Ryu e é criado o plano de dados, conforme apresentado na Figura 05. O serviço *Openflow* configura o Switch para funcionar em modo *out-of-band*, com suporte ao *OpenFlow* 1.3,

e cria um *datapath* ou plano de dados com o *id* 000000000111 que irá possibilitar a comunicação entre os nós da rede *OpenFlow*. Após estabelecida a conexão, o plano de controle dá suporte a um switch L2 com integração a ferramenta Snort por meio da biblioteca *snort.lib* do Linux e coloca a porta 3 do *switch* em modo de espelhamento, viabilizando a análise de todo o tráfego da rede *OpenFlow* pela ferramenta Snort em busca de assinaturas de ataques. A partir da criação do plano de dados é possível identificar o pleno funcionamento da rede SDN por meio de um simples comando *ping* entre os nós da rede *OpenFlow*.

Figura 05: Criação do plano de dados da rede *OpenFlow*

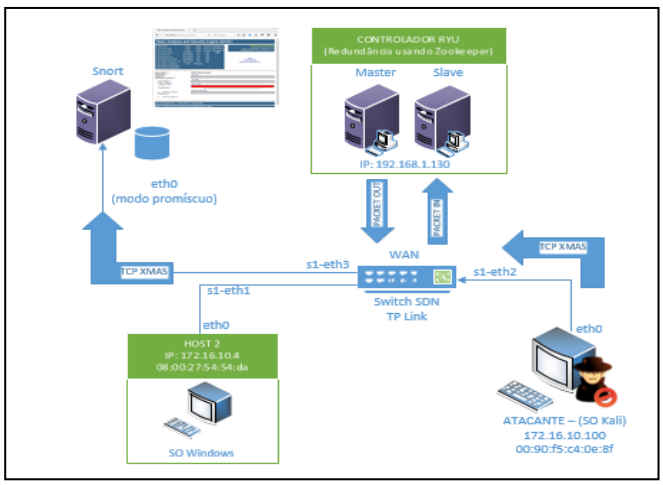
6 Simulações e resultados obtidos

O objetivo deste estudo de caso é mostrar como os recursos de detecção e reação apresentados neste trabalho podem ser empregados em ambiente de *testbed,* como fonte de pesquisa, para coletar, analisar e estudar ataques usando o paradigma das redes *OpenFlow* e sua flexibilidade por meio da programabilidade do controlador da rede.

6.1 Fase 1: Validação dos mecanismos de detecção

A realização dos testes de validação dos mecanismos de detecção, podem ser acompanhados por meio da Figura 06 e consiste na execução da ferramenta Snort e o uso da ferramenta nmap na máquina ATACANTE para geração de tráfego malicioso usando técnica de escaneamento de rede (*port scan*) direcionado a máquina 172.16.10.4..

Figura 06: Ataque de escaneamento (*port scan*)



Ao final da execução do comando de execução apresentado abaixo a ferramenta Snort inicia no modo de detecção de intrusão de rede e passa a escutar todo o tráfego da rede, de forma promíscua, que está entrando pela interface eth0 da própria máquina do Snort.

```
#sudo snort –i eth0 –A console –l /var/log/snort –c /etc/snort/snort.conf –g snort –u snort
```

O parâmetro -A console refere-se ao modo de alerta que envia alertas para a tela do computador, a opção -l /var/log/snort informa o caminho em que os alertas serão armazenados, a opção -c /etc/snort/snort.conf define o caminho do arquivo de configuração da ferramenta Snort que contém as regras configuradas que serão utilizadas para comparação com os dados capturados e as opções -g snort e -u snort informam que o comando será executado com as permissões do usuário e grupo snort respectivamente.

Para validação do funcionamento do Snort foi criada uma regra local no arquivo de configuração local.rules que permite identificar tráfego suspeito na rede que possuam todas as flags do cabeçalho TCP habilitados (flags: SRAFPU):

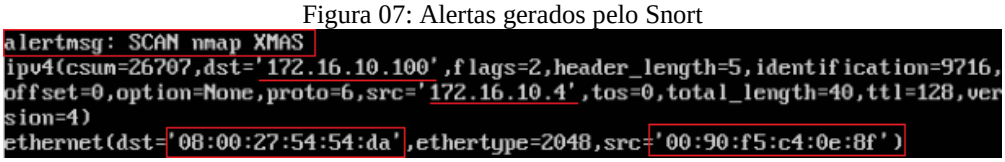
alert tcp any any -> $HOME_NET any (msg: "XMAS Scan"; flags: SRAFPU; GID:1; sid:9000003; rev:001; classtype:attempted-recon;

Neste ponto é possível iniciar a simulação de ataque para validação da regra criada, por meio da execução da ferramenta nmap na máquina do ATACANTE, direcionado ao nó com o sistema operacional Windows na máquina com o IP 172.16.10.4, da seguinte forma:

#nmap -sX 172.16.10.4

A opção -sX habilita um escaneamento, onde o nmap envia pacotes combinando os parâmetros FIN, PUSH e URG do pacote TCP, se aproveitando de que muitos *firewalls* ignoram a detecção desta possibilidade. Como a interface *eth0* da máquina Snort recebe todo o tráfego que está passando pela interface *eth3* do Switch *OpenFlow* (modo de espelhamento) o pacote TCP XMAS é identificado pelo Snort como tráfego malicioso, armazenado em banco de dados MySQL e pode ser analisado por meio da ferramenta BASE pelo administrador de rede para revisão das políticas de segurança aplicadas na rede *OpenFlow*. E como o Snort está sendo executado em modo console, o tráfego gerado pelo comando nmap é identificado como malicioso e apresentado de acordo com a Figura 07 na saída da tela do computador:

Figura 07: Alertas gerados pelo Snort



É possível observar que foi gerado uma mensagem de alerta pelo Snort, identificando um ataque do tipo "SCAN nmap XMAS", que por meio da ferramenta nmap gerou ataques do tipo escaneamento de rede da máquina do ATACANTE no endereço IP 172.16.10.100 e endereço físico 08:00:27:54:54:da com destino a máquina Windows no IP 172.16.10.4 e endereço físico 00:90:f5:c4:0e:8f.

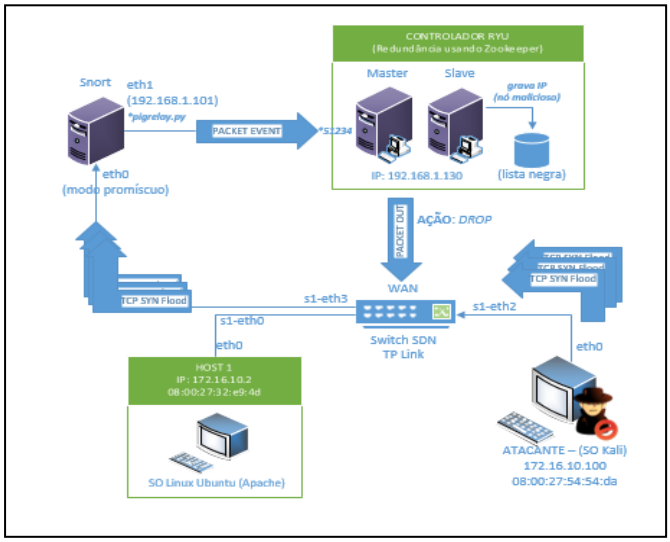
6.2 Fase 2: Validação dos mecanismos de reação

A realização dos testes de validação dos mecanismos de reação, podem ser acompanhados por meio da Figura 08 e consiste na execução da ferramenta Snort e o uso da ferramenta t50 na máquina do ATACANTE para geração de tráfego malicioso por meio de teste de stress direcionado ao servidor de aplicação de páginas executando o serviço Apache na máquina 172.16.10.2.

Ao final da execução do comando abaixo a ferramenta Snort inicia no modo de detecção de intrusão de rede, idêntico a execução do Snort da FASE 1, com a diferença de que agora a opção -A usa o parâmetro unixsock que irá permitir redirecionar os alertas detectados pelo Snort para aplicação pigrelay, por meio de um soquete de rede, que permitirá que as mensagens *Packet_event* sejam encaminhadas para o controlador da rede na porta 51234.

#sudo snort –i eth0 –A unixsock –l /var/log/snort –c /etc/snort/snort.conf –g snort –u snort

Figura 08: Ataque de stress de rede (*DoS*)



Para validação do funcionamento do Snort foi criada uma regra local no arquivo de configuração local.rules que permite identificar tráfego suspeito na rede *OpenFlow* do tipo *DoS flood denial of service*:

Alert tcp any any -> [172.16.10.100] any (flags: S; msg: DoS flood denial of service"; treshold: type both, track by_src, count 100, seconds 1 sid:1000000010; rev:1;)

Neste ponto é possível iniciar a simulação de ataque para validação da regra criada, por meio da execução da ferramenta t50 na máquina do ATACANTE, direcionado ao nó que executa o serviço de aplicação Apache na máquina Linux com o IP 172.16.10.2. A ferramenta t50 é um injetor de pacotes utilizado na execução de testes de stress em uma rede, permitindo simular ataque DoS e DDoS e que permite disparar simultaneamente pacotes de diferentes protocolos. Para fins de simulação, foi usado o seguinte comando na máquina do atacante:

#t50 172.16.10.2 –flood –S –turbo –dport 80

Neste comando os seguintes parâmetros são definidos: "—flood" substitui o threshold, "-S" inicia a conexão através do TCP SYN Flag, "—turbo" aumenta a potencialidade do ataque e "—dport" define a porta a ser atacada. No momento em que se dá o início do ataque é possível identificar por meio da ferramenta de monitoramento e gerenciamento de rede ntop (Figura 09), em execução no *switch OpenFlow*, que no pico do ataque o consumo de CPU no plano de dados (*datapath*) chegou a ordem de 92%.

Figura 09: Saída gerada pela máquina do Snort

```
Mem: 25848K used, 3452K free, 0K shrd, 2076K buff, 12392K cached
CPU:  46% usr  29% sys   0% nic   0% idle   0% io   0% irq  24% sirq
Load average: 1.62 1.08 0.58 2/41 24739
  PID  PPID USER     STAT   VSZ %VSZ %CPU COMMAND
22639     1 root     R     1396   5%  92% ofdatapath ptcp:6634 -i eth0.1 eth0.2
22647     1 root     S     1136   4%   0% ofprotocol tcp:127.0.0.1:6634 tcp:192
15936     2 root     SW       0   0%   0% [kworker/u:2]
```

Seguindo esta tendência e não sendo adotada nenhuma medida de mitigação para conter o ataque DoS, a rede estaria comprometida e seria impossível o *switch OpenFlow* tratar novos tipos de tráfegos na rede. É neste

sentido que foram implementados alguns recursos na arquitetura proposta e que serão apresentadas nos próximos parágrafos para tornar efetiva a implementação dos mecanismos de reação.

Assim que a máquina do Snort começa a identificar o tráfego malicioso na rede, a aplicação *pigrelay.py* inicia o envio dos alertas gerados no formato de *Packet_event* para o controlador por meio de uma *daemon* no servidor Linux. Como apresentado na Fig 10 o IDS Snort se mostrou eficaz na detecção da simulação de ataque gerados pela ferramenta T50, detectando a intrusão e gerando alertas para posterior análise.

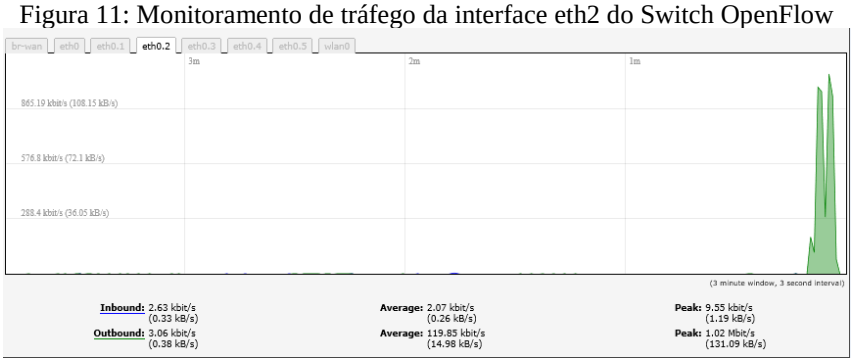
Figura 10: Alertas *Packet_event* recebidos pelo controlador Ryu

```
[snort][INFO] Connected with 192.168.1.101
EVENT ofp_event->SimpleSwitchSnort EventOFPPacketIn
EVENT snortlib->SimpleSwitchSnort EventAlert
alertmsg: DoS flood denial of service
ipv4(csum=43936,dst='172.16.10.2',flags=2,header_length=5,identification=22308,o
ffset=0,option=None,proto=6,src='140.69.54.115',tos=64,total_length=40,ttl=255,v
ersion=4)
ethernet(dst='08:00:27:32:e9:4d',ethertype=2048,src='08:00:27:54:54:da')
```

A imagem gerada na Figura 10 foi capturada na saída do comando de execução em modo verbose do controlador Ryu e indica que antes de enviar os pacotes do tipo *Packet_event* para o controlador, a máquina do Snort por meio de sua interface eth1 no IP 192.168.1.101 estabelece um procedimento de conexão cliente-servidor com o controlador Ryu na porta 51234. Ainda na Figura 10 é possível comprovar que o controlador Ryu recebeu uma mensagem de alerta por meio da biblioteca snortlib do Linux, que foi identificado como um ataque do tipo "*denial of service*", tendo como origem a ferramenta t50 que gerou ataques do tipo SYN Flood com endereços de redes aleatórios e endereço físico 08:00:27:54:54:da com destino ao IP 172.16.10.2 e endereço físico 08:00:27:32:e9:4d. A cada simulação de ataque detectada pelo Snort, uma mensagem de alerta era encaminhada ao controlador da rede, que por meio da aplicação mitigacao.py extraía as informações do alerta e buscava o fluxo que coincidisse com o alerta recebido.

No caso deste trabalho ao encontrar o fluxo a ação consiste em enviar uma mensagem *OpenFlow* com ação DROP ao *switch* para bloquear todo o tráfego que tenha por origem o host responsável pela ação maliciosa. Outra ação programada na aplicação *mitigacao.py* é inserir em uma blacklist o IP do host responsável pela ação maliciosa para garantir que não seja possível novas conexões do nó na rede *OpenFlow* até que o administrador pudesse garantir que o host esteja em conformidade com as políticas de segurança. Isto é possível, pois toda nova conexão (*Packet_in*) enviada ao controlador só é tratada depois da leitura do arquivo de *blacklist*. Desta forma, após qualquer tipo de ação maliciosa detectada por um determinado nó da rede e estando este nó na blacklist torna-se praticamente impossível um novo acesso a rede, já que a política da rede segue o padrão *deny-by-default*. A validação dos mecanismos de reação pode se comprovar por meio da Figura 11 que apresenta o monitoramento do tráfego de fluxos na rede na interface *eth2* do *switch Openflow* que está conectado ao nó do servidor Apache no momento do ataque DoS. No nomento da execução da ferramenta T50 o tráfego na interface eth2 alcançou um volume de tráfego de entrada de 900Kbps e este tráfego foi restabelecido após terem sido tomadas as medidas de mitigação apresentadas neste trabalho.

Figura 11: Monitoramento de tráfego da interface eth2 do Switch OpenFlow



Pode-se inferir então, que o tempo decorrido desde o momento da identificação do tráfego malicioso pelo Snort, o envio da mensagem *Packet_event* para o *switch* e a efetiva remoção (DROP) do fluxo levou em média de 6 a 7 seg, intervalo este em que se percebe o início e o término do pico de tráfego apresentado na

interface *eth2* do *switch OpenFlow* na qual esta conectado a máquina Linux que sofreu o ataque DoS, demonstrado por meio da Figura 11.

7 Conclusão

O desafio da implantação de uma estratégia de segurança para redes baseadas no paradigma SDN com *OpenFlow* é o foco da arquitetura de segurança proposta. A arquitetura juntamente com sua abordagem de segurança inova principalmente na detecção de intrusão por meio da utilização de um IDS. Esta técnica, agora associada aos mecanismos de controle de rede da arquitetura *SDN/OpenFlow*, permite de forma inovadora, não somente a detecção de potenciais ataques como também uma "reação" às ameaças de forma controlada. Em efeito, os testes de validação relatados ilustraram uma das abordagens de reação com a inclusão em lista negra e sinalização para o administrador. De maneira geral, a potencialidade e diversidade da reação é, agora no contexto *SDN/OpenFlow*, programável e, além disso, pode ser dinamicamente ajustada segundo critérios e políticas de gerência da rede e/ou sistema.

A integração do controlador Ryu com o Snort, com funcionalidades apenas de um IDS, permitiu a concepção de um ambiente de rede com mecanismos de proteção com comportamento idêntico a de um IPS, disponibilizando recursos não só de detecção, mas também de prevenção de intrusão. Outro aspecto importante, no contexto da solução, foi poder manipular informações do cabeçalho TCP/IP de L3 em uma rede que é essencialmente de L2, permitindo detectar ações maliciosas, de forma centralizada no controlador, e poder tomar uma ação mais perto da origem possível.

A proposta, em termos do seu potencial de utilização e resultados apresentados, se alinha ao conjunto de novas soluções e mecanismos que dão suporte para uma implantação mais segura do paradigma *SDN/OpenFlow*.

Referências

[1] SHERWOOD, R. e al. Carving research slices out of your production networks with openflow. SIGCOMM Comput. Commun. Rev. , 40:129–130, 2010.

[2] ANWER M. BILAL; MOTIWALA M.; TARIQ, M. B.; FEAMSTER, N. Switchblade: a platform for rapid deployment of network protocols on programmable hardware. Anais da ACM SIGCOMM 2010 Conference;40(4):183-194, 2010.

[3] MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: Enabling innovation in campus networks SIGCOMM Comput. Commun. Rev. , 38(2):69–74, 2008.

[4] MATTOS, D.; FERNANDES, N.; DA COSTA, V.; CARDOSO, L.; CAMPISTA, M.; COSTA, L.; DUARTE, O. OMNI: OpenFlow MaNagement Infrastructure. In: Network of the Future (NOF), 2011 International Conference on the pages 52 –56, 2011.

[5] BOBBA, R.; BORRES, D.; HILBURN, R.; SANDERS, J.; HADLEY, M.; SMITH, R. Redes definidas por software (SDN). Disponível em: <osetoreletrico.com.br/web/a-revista/edicoes/1786-redes-definidas-por-software-sdn.html>. Acesso em: Out. 2016.

[6] MATTOS, D. M. F.; FERRAZ, L. H. G.; DUARTE, O. C. M. B. Um mecanismo para isolamento seguro de redes virtuais usando a abordagem hıbrida Xen e OpenFlow. Em XIII SBSeg'13, paginas 128–141, 2013.

[7] PORRAS, P.; SHIN, S.; YEGNESWARAN, V.; FONG, M.; TYSON, M.; GU, G. A security enforcement kernel for openflow networks. Proceedings of the First Workshop on Hot Topics in Software Defined Networks ACM, 2012.

[8] SHIN, S.; PORRAS, P.; YEGNESWARAN, V.; FONG, M.; GU, G.; TYSON, M. FRESCO: Modular composable security services for software-defined networks. In: Proceedings of Network and Distributed Security Symposium, 2013.

[9] BALLARD, J. R.; RAE, I.; AKELLA, A. Extensible and scalable network monitoring using opensafe. Proc. INM/WREN, 2010.

[10] XING, T.; HUANG, D.; XU, L.; CHUNG, C. J.; KHATKAR, P. Snort-flow: A openflow-based intrusion prevention system in cloud environment. In Research and Educational Experiment Workshop (GREE), 2013 Second GENI, pages 89–92. IEEE, 2013.

[11] MEHDI, S. A.; KHALID, J.; KHAYAM, S. A. Revisiting traffic anomaly detection using software defined networking. In: Recent advances in intrusion detection. Springer; p. 161 e 80, 2011.

[12] NAGAHAMA, F. Y.; FARIAS, F.; AGUIAR, E.; GASPARY, L.; GRANVILLE, L.; CERQUEIRA, E.; ABELÉM, A. IPSFLOW uma proposta de sistema de prevenção de intrusão baseado no framework openflow. Anais do III WPEIF-SBRC'12, páginas 42–47, 2012.

[13] XING, T., HUANG, D.; XU, L.; CHUNG, C. J.; KHATKAR, P. Snort-flow: A openflow-based intrusion prevention system in cloud environment. In Research and Educational Experiment Workshop (GREE), 2013 Second GENI, pages 89–92. IEEE, 2013.

[14] MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: Enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev., 38(2):69–74, 2008.

[15] ONSUMMIG. Open Networking Summit 2012. Disponível em: <http://opennetsummit.org>. Acesso em: Nov. 2016.

[16] OPENFLOW. OpenFlow Specification. Disponível em: <http://OpenFlowSwitch.org>. Acessado em: Jul. 2012.

[17] GUDE, N.; KOPONEN, T.; PETTIT, J.; PFAFF, B.; CASADO, M. MCKEOWN, N; SHENKER, S. NOX: towards na operating system for networks. SIGCOMM Computer Communication Review, 38:105110, 2008,

[18] RYU, P.T. (2014). Ryu SDN framework using openflow 1.3. Disponível em: <http://osrg.github.io/ryu-book/en/Ryubook.pdf>. Acesso em: 30 Out. 2016.

[19] OPENWRT. OpenWrt - Wireless Freedom. Setembro 2012. Disponível em: <https://openwrt.org/>. Acesso em: 30 Out. 2016.

[20] CPQD. OpenFlow 1.3 Software Switch. Disponível em: <https://github.com/CPqD/ofsoftswitch13>. Acesso em: 30 Nov. 2016.

[21] COURSERA. Curso de SDN. Disponível em: <https://www.coursera.org/course/sdn>. Acesso em: 12 Jan. 2016.

[22] MALLERY, J. Hardening Linux. Editora McGraw-Hill Osborne Media, 2004.

[23] PIGRELAY. Disponível em: <https://github.com/John-Lin/pigrelay/blob/master/pigrelay.py>. Acesso em: 30 Out 2016.