UNIVERSIDADE SALVADOR – UNIFACS
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO
MESTRADO EM SISTEMAS E COMPUTAÇÃO

MAXLI BARROSO CAMPOS

UM AMBIENTE PARA DETECÇÃO E PREVENÇÃO FLEXÍVEL DE ATAQUES EM REDES OPENFLOW/SDN

Salvador
2017

MAXLI BARROSO CAMPOS

UM AMBIENTE PARA DETECÇÃO E PREVENÇÃO FLEXÍVEL DE ATAQUES EM REDES OPENFLOW/SDN

Dissertação apresentada ao Programa de Pós-Graduação em Sistemas e Computação da UNIFACS Universidade Salvador, Laureate International Universities, como requisito parcial para obtenção do título de Mestre.

Orientador: Prof. Dr. Joberto S. B. Martins.

Salvador
2017

FICHA CATALOGRÁFICA
(Elaborada pelo Sistema de Bibliotecas da UNIFACS Universidade Salvador, Laureate International Universities)

Campos, Maxli Barroso

Um ambiente flexível para detecção e prevenção flexível de ataques em redes OpenFlow/SDN / Maxli Barroso Campos. - 2017.

139 f. : il.

Dissertação (Mestrado) – UNIFACS Universidade Salvador – UNIFACS, Laureate International Universities. Mestrado em Sistemas e Computação, 2017.

Orientador: Prof. Dr. Joberto S. B. Martins.

1. Rede de computadores - Segurança. 2. OpenFlow. 3. Redes Definidas por Software. I. Martins, Joberto B., orient. II. Título.

MAXLI BARROSO CAMPOS

UM AMBIENTE PARA DETECÇÃO E PREVENÇÃO FLEXÍVEL DE ATAQUES EM REDES OPENFLOW/SDN

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre em Sistemas e Computação, UNIFACS Universidade Salvador, Laureate International Universities, pela seguinte banca examinadora:

Joberto S. B. Martins - Orientador _____
Doutor em Ciência da Computação, Université Pierre et Marie Curie, Paris - França
UNIFACS Universidade Salvador, Laureate International Universities

Leobino Nascimento Sampaio _____
Doutor em Ciência da Computação, Universidade Federal de Pernambuco (UFPE)
Universidade Federal da Bahia (UFBA)

Paulo Nazareno Maia Sampaio_____
Doutor em Informática e Telecomunicações, Universidade Paul Sabatier, Paria - França
UNIFACS Universidade Salvador, Laureate International Universities

Salvador, 3 de abril de 2017.

Dedico este trabalho aos meus pais: Jorge e Nadir, que com muita dificuldade ultrapassaram todos obstáculos que a vida apresentou a eles e que nem por isso deixaram de criar e dar condições para me apoiarem na minha formação e que me permitiu galgar mais este degrau na minha vida profissional e acadêmica.

Maxli B. Campos.

AGRADECIMENTOS

A Deus pela vida.

Aos professores do Programa de Pós-graduação em Sistemas e Computação da UNIFACS que sempre buscaram passar suas experiências ao corpo discente do curso com profissionalismo e dedicação.

Ao meu professor orientador, Prof Dr Joberto B. Martins, pela orientação e amizade com que sempre conduziu nossos encontros na elaboração desta dissertação e principalmente pelo estímulo à promoção do trabalho por meio de publicação de artigos em eventos acadêmicos. Ao receber minhas primeiras respostas de aprovação de publicação de artigos em eventos, até mesmo internacionais, pude me convencer que o caminho definido por mim como estratégia de pesquisa desde o primeiro semestre do mestrado estava certo e percebi que boa parte deste sucesso se devia a orientação precisa, meticulosa e oportuna deste baiano que nunca se deixou intimidar pela ansiedade de seu orientado carioca.

À secretária do Programa de Pós-graduação em Sistemas e Computação da UNIFACS, Sra. Josiane, que em todos os momentos do curso que precisei de seu apoio, foi atenciosa e sempre buscou me atender oportunamente. Após longo caminho traçado nestes dois anos de mestrado, como não se lembrar da sua receptividade e atenção no processo de seleção do curso daquele carioca que estava com menos de um mês em Salvador, sequer conhecia a Universidade, e que ainda tinha dúvidas se iria fazer o curso ou não.

Finalmente, agradeço à minha amada esposa Luciene pela companhia, dedicação a família, amor e paciência, e aos meus filhos Maxli e Davi que sempre buscaram entender que os momentos em que tive ausente foi por um motivo maior, principalmente durante o período em que me dediquei com afinco ao mestrado e a produção desta dissertação.

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 infraestruturas críticas como as redes elétricas inteligentes. 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, esta dissertação de mestrado propõe a implementação de um Ambiente para Detecção e Prevenção Flexível de Ataques em redes *OpenFlow/SDN,* que contempla a construção de uma rede real *OpenFlow/SDN* em ambiente de experimentação *(Testbed)* e mecanismos implementados de detecção e reação a ameaças de segurança, capaz de fornecer recursos para realização de análises de intrusão e de ataques. Estes mecanismos foram implementados com o objetivo de prover um monitoramento e tratamento de eventos de segurança de forma flexível, por meio da categorização dos tipos de ataque e associado a recursos de *whitelist* e *blacklist,* explorando uma das principais características do controlador de rede *OpenFlow* de ser extensível e programável. A validação do ambiente foi feita por meio da simulação de 5 (cinco) diferentes cenários e os resultados obtidos demonstram que as diferentes classes de ataques de rede foram identificadas e tratadas conforme a estratégia de tratamento definida. Além disso, os mecanismos de proteção e de análise de intrusão na rede real *OpenFlow/SDN* em ambiente de experimentação foram efetivos e tiveram o comportamento esperado, de acordo com cada classe de tráfego definida na proposta de trabalho.

Palavras-chave: Redes Definidas por Software – SDN. OpenFlow. Ameaças de segurança. Tratamento e resposta a incidente.

ABSTRACT

Software Defined Networking (SDN) paradimg decouples control plane and of data plane, providing high programmability and a global view of the network. The adoption of this approach is growing in business networks, data centers and critical infrastructures such as smart grids. However, it is a challenge not only to provide security in these new generation networks but also to allow a network attack to be susceptible to an incident handling and forensic expertise procedure. In this way, this master's degree dissertation proposes the implementation of a Environment for Flexible Attacks Detection and Prevention in OpenFlow/SDN Networks, which includes the setup of a real OpenFlow/SDN Testbed environment and implemented mechanisms for detection and response to threats capable of providing resources for intrusion and attack analysis. These mechanisms were implemented with the goal of providing security event monitoring and treatment in a flexible way, by categorizing the attack types and associated with whitelist and blacklist resources, exploiting one of the main characteristics of the OpenFlow network controller to be Extensible and programmable. The validation of the environment was done through simulation based on 5 (five) different scenarios and the obtained results demonstrate that the different classes of network attacks were identified and treated according to the defined strategy. Also, the mechanisms of protection and analysis of the intrusion into the OpenFlow/SDN network Testbed environment were effective and had the expected behavior according to each class of traffic defined in the proposal.

Keywords: Software-Defined Networking – SDN. OpenFlow. Security threats. Incidente response and treatment.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AC *Autoridade de Certificação*

ACID *Analysis Console for Intrusion Detection*

ACL *Access Control List*

ANSI American *National Standard Institute*

API *Application Programming Interface*

ARP *Address Resolution Protocol*

BGP *Border Gateway Protocol*

BASE *Basic Analysis and Security Engine*

CERT.BR *Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil*

COTS *Commercial offtheshelf*

CPQD *Centro de Pesquisa e Desenvolvimento em Telecomunicações*

CPU *Central Processing Unit*

DoS *Denial of Service*

DDoS *Distributed Denial of Service*

DHCP *Dynamic Host Configuration Protocol*

DNS *Domain Name System*

DPI *Deep Packet Inspection*

EAP *Extensible Authentication Protocol*

FIFO *FirstIn FirstOut*

FITS *Future Internet Testbed with Security*

GPL *General Public License*

HIDS *Hostbased Intrusion Detection System*

HTTP *Hypertext Transfer Protocol*

IDPS *Intrusion Detection and Prevention Systems*

IDS *Intrusion Detection Systems*

IPS *Intrusion Prevention Systems*

IEEE *Institute of Electrical Electronics Engineers*

IETF *Internet Engineering Task Force*

IGMP *Internet Group Management Protocol*

IPS	*Intrusion Prevention Systems*
ICMP	*Internet Control Message Protocol*
IP	*Internet Protocol*
ISO	*International Standards Organization*
ITUT	*International Telecommunication Union*
JSON	*JavaScript Object Notation*
L2	*Layer 2*
L3	*Layer 3*
LAN	*Local Area Network*
LCAP	*Link Agregation Control Protocol*
MAC	*Media Acess Control*
MPLS	*Multiprotocolo Label Switching*
NAT	*Network Address Translation*
NIDS	*Networkbased Intrusion Detection System*
NOC	*Network Operations Center*
OF	*OpenFlow*
OISF	*Open Information Security Foundation*
OPKG	*Open Package Management*
OS	*Operation System*
OSI	*Open Systems Interconnection*
OSPF	*Open Shortest Path First*
PCAP	*Packet Capture*
PSI	*Política de Segurança da Informação*
QOS	*Quality of Service*
RADIUS	*Remote Authentication Dial In User Service*
RAM	*Random Access Memory*
SDN	*Software Defined Networking*
IDPS	*Instrusion Detection and Prevention System*
SNOC	*Security Network Operation Center*
SSH	*Secure Shell*
SSL	*Secure Sockets Layer*
STP	*Spanning Tree Protocol*
TCP	*Transmission Control Protocol*

TCP/IP	*Transmission Control Protocol/Internet Protocol*
TLS	*Transport Layer Security*
UDP	*Unified Datagram Protocol*
VLAN	*Virtual Local Area Network*
VLAN ID	*Virtual Local Area Network Identifier*
VLAN PCP	*Virtual Local Area Network Priority Code Point*
VPN	*Virtual Private Network*
WSGI	*Web Server Gateway Interface*
WWW	*World Wide Web*

SUMÁRIO

1 INTRODUÇÃO

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.

Segundo Gomes e outros autores (2013), ao longo dos últimos anos, apesar de ter se buscado novas soluções e alterações no protocolos da arquitetura original das redes de computadores, a estrutura básica da rede não sofreu grandes mudanças e ainda gerou novos problemas de incompatibilidade com as novas e crescentes demandas do projeto original da internet.

O alto nível de complexidade das redes dificulta a inovação (MOREIRA et al., 2009), e a ausência de flexibilidade no controle do funcionamento interno dos equipamentos, associado ao alto custo da infraestrutura se apresentam como barreiras para a evolução das arquiteturas e a oferta de novos serviços e aplicações de rede (AWNER et al, 2010). A comunidade de redes de computadores tem trabalho ao longo dos últimos anos em algumas iniciativas que contemplem a substituição gradativa do protocolo IP (FARIAS et al, 2011) e a formulação e avaliação de arquiteturas alternativas para a Internet do Futuro (*Future Internet* – FI).

Uma das iniciativas nesse sentido foi a proposição do novo paradigma das Redes Definidas por Software (*Software Defined Networking* – SDN), tendo como uma das bases para sua implementação o protocolo *OpenFlow* (MCKEOWN et al, 2008). O novo paradigma computacional de redes SDN propõe o desacoplamento entre o plano de dados, implementado normalmente 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 realizadas pelo plano de dados (KREUTZ et al., 2014). No paradigma SDN são utilizadas abstrações que visam a programabilidade e a consequente simplificação da rede.

O *OpenFlow* é um protocolo que permite programar o comportamento dos fluxos de dados (FARIAS et al, 2011), por meio de uma interface de programação

de aplicação (*Application Programming Interface* – API) simples e extensível. O *OpenFlow* controla equipamentos (*switches*, ponto de acesso sem fio, multiplexadores ópticos, outros) com comandos padronizados através de um controlador que, efetivamente, programa o comportamento dos fluxos de dados nos equipamentos de rede e a funcionalidade da aplicação em questão (VAUGHAN-NICHOLS, 2011).

A concepção deste novo paradigma permite pesquisadores terem total autonomia para estender ou alterar a lógica de funcionamento dos equipamentos com suporte ao protocolo *OpenFlow*, e em contrapartida os fabricantes de *hardware* não têm a necessidade de expor detalhes sobre o funcionamento interno de seu equipamento (SHENKER et al., 2011).

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 pelo fato de que grande parte das ferramentas e técnicas utilizadas atualmente para segurança da informação como antivírus e *firewall* são mais eficientes para assegurar a segurança, ou ainda porque estima-se que 70% dos ataques a uma rede seja de usuários internos (LYNCH, 2006). Somado a isso, a natureza tipicamente centralizada do plano de controle (BOBBA et al, 2015), acaba por originar novos problemas exclusivos deste domínio, sendo que as mesmas características da SDN que são desejáveis, como por exemplo a gestão e configuração centralizada, acabam se tornando alvos de ataques, podendo causar grande impacto na rede, caso o controlador venha a ser comprometido.

Assim sendo, a segurança em redes SDN envolve pelo menos dois aspectos: a própria segurança da arquitetura SDN e a utilização da estratégia centralizada e a programabilidade do SDN como forma de tratar a segurança de rede. Esta dissertação de mestrado foca neste segundo aspecto evocado e não trata explicitamente as questões inerentes de segurança da arquitetura SDN.

A principal abordagem para estabelecer mecanismos de proteção contra ameaças internas e externas em uma rede consiste no uso dos Sistemas de Detecção e Prevenção de Intrusão (*Instrusion Detection and Prevention System* – IDPS) que além de complementar os métodos convencionais de segurança,

possibilita que o administrador de rede tenha ferramentas que possibilite o monitoramento, tratamento e análise dos eventos de segurança.

Logo, para se conceber uma rede tendo como base o paradigma *OpenFlow*/SDN que disponha de mecanismos de proteção eficazes e instrumentos de monitoramento e tratamento de eventos de segurança, este estudo buscou responder as seguintes questões:

a) Como o potencial do paradigma *OpenFlow* pode ser explorado para concepção de um ambiente de rede que contemple uma gerência de segurança de forma centralizada?
b) Como implementar mecanismos de monitoramento e tratamento de eventos de segurança no plano de controle?
c) Qual melhor estratégia de validação da proposta traria resultados mais próximos de um ambiente de rede real?

1.1 OBJETIVOS DA DISSERTAÇÃO

O objetivo geral do trabalho consiste em implementar um Ambiente para Detecção e Prevenção Flexível de Intrusão em redes *OpenFlow/SDN*, que contempla a construção de uma rede real *OpenFlow/SDN* em ambiente de experimentação e mecanismos de detecção e reação a ameaças de segurança, explorando o novo paradigma das redes SDN e fornecendo recursos para realização de análise de intrusão e de ataques em um ambiente de rede *OpenFlow* usando tecnologia de detecção de intrusão por meio de um IDPS.

Tendo em vista o objetivo geral deste estudo, foram definidos seis objetivos específicos:

1) Identificar as ameaças de intrusão que podem comprometer uma rede.
2) Detalhar a função dos componentes empregados para construção do ambiente de rede real *OpenFlow/SDN* para experimentação.
3) Apresentar uma estratégia de tratamento de eventos de intrusão baseado em Classes de Ataques.
4) Detalhar o modelo da solução no plano de controle para a implementação de um Ambiente para Detecção e Prevenção Flexível de Ataques em redes *OpenFlow/SDN*.

5) Apresentar os resultados de avaliação e validação do ambiente de rede real *OpenFlow* para experimentação proposta no trabalho.
6) Apresentar os resultados de avaliação e validação dos mecanismos de proteção implementados no plano de controle e definidos no trabalho.

1.2 TRABALHOS RELACIONADOS

Shim et al. propõem o sistema FRESCO (SHIN et al., 2013), que provê uma linguagem de programação e módulos de *software* para desenvolver aplicações de segurança, permitindo a integração de serviços de segurança sobre uma nova arquitetura de controladores. Consiste em um arcabouço de segurança para o controlador NOX *OpenFlow*, que visa impedir ataques por *Scan*, aplicação de detecção de *bots* na rede e aplicação para detectar P2P *malware*.

O uso dos IDPS como solução para problemas de segurança voltados para análise e tratamento de intrusão em *OpenFlow/SDN* já foi abordado e apresentou bons resultados em pesquisas anteriores, como, por exemplo, nas pesquisas de Ballard et al. (2010), Mattos et al. (2013), Porras et al. (2012), Xing et al. (2013) e Nagahama et al. (2012).

Ballard et al. (2010) propõem o OpenSAFE que funciona como um sistema de redirecionamento de fluxos para aplicações de monitoramento e segurança e especifica uma linguagem para um gerenciamento simplificado de fluxos. Implementa funções básicas para seleção de fluxos específicos, por exemplo usando filtros, e também desvios com o objetivo de encaminhar o tráfego para aplicações de monitoramento e segurança. Mattos et al. (2013) propõem BroFlow, um IDPS para defesa contra ataques do tipo de negação de serviço (*Denied of Service* - DoS) em uma rede SDN virtualizada, baseada *OpenFlow* e aliando as características da ferramenta de análise de tráfego de rede Bro com as características de visão global da rede OpenFlow. Contempla uma arquitetura modular flexível que facilita o acréscimo de políticas de análise e detecção de ataques, além de conseguir reagir rapidamente a um ataque e descartar os pacotes atacantes desde a sua origem.

Problemas de potenciais conflitos no estabelecimento de regras de fluxos entre diferentes aplicações nos controladores *OpenFlow* são tratados por Porras et

al. (2012). A proposta consiste em implementar uma extensão de segurança nos controladores NOX-OpenFlow, onde nesta extensão cada aplicação terá políticas com diferentes prioridades atribuídas, dependendo da sua função. Desta forma, ao serem aplicadas nos fluxos do controlador, as políticas voltadas para as aplicações de segurança são prioritárias.

Xing et al. (2013) apresentaram o SnortFlow como proposta de um IDPS em um ambiente de nuvem baseado na solução com XEN, baseado em redes *OpenFlow,* para auxiliar na captura de tráfego. Os autores do trabalho elaboraram um protótipo no qual o agente SnortFlow foi instalado no domínio de gerência do hipervisor XEN, com a finalidade de analisar e inspecionar todo o tráfego das máquina virtuais de uma máquina física.

Uma outra proposta que executa um mecanismo automático de tráfego malicioso é o framework IPSFlow (NAGAHAMA, 2012) que utiliza uma rede baseada na arquitetura SDN e o protocolo OpenFlow para a criação de um IDPS com ampla cobertura e que permite bloquear um tráfego caracterizado pelos IDPSs como malicioso no equipamento mais próximo da origem.

Uma série de pesquisas relacionadas à SDN empregando o paradigma *OpenFlow* foram realizadas nos últimos anos. Contudo, ainda são raros os trabalhos que focam no plano de dados dos *switches OpenFlow*, bem como suas implementações e mecanismos internos. Nesse sentido, a proposta deste trabalho visa estender os trabalhos relacionados por meio da implementação de um Ambiente de Detecção e Prevenção Flexível de Ataques em redes *OpenFlow/SDN*, onde os mecanismos de monitoramento e tratamento de eventos de intrusão implementados foram validados em um ambiente de rede *OpenFlow*, com tráfego de rede e equipamentos reais, dando subsídios para apresentação, demonstração e validação da pesquisa e que permita realizar: a captura do tráfego de rede, encaminhamento do tráfego para análise em um IDS e realizar ações que permitam tanto a remoção quanto a reconfiguração automática da tabela de fluxos do *Switch OpenFlow* quando identificado alguma ação maliciosa na rede.

1.1 Metodologia adotada nesta Dissertação

Quanto à natureza, o presente estudo caracteriza-se por ser uma pesquisa do tipo aplicada e tem por objetivo gerar conhecimentos para aplicação prática dirigidos à viabilidade de implementar mecanismos de proteção por meio da

programabilidade do plano de controle em em ambiente de rede real *SDN/OpenFlow* para experimentação. Para atingir o objetivo geral foram utilizadas as seguintes metodologias de pesquisa:

a) Pesquisa bibliográfica que, para sua consecução, teve por método a leitura exploratória e seletiva do material de pesquisa, bem como sua revisão integrativa, contribuindo para o processo de síntese e análise dos resultados de vários estudos, de forma a consubstanciar um corpo de literatura atualizado e compreensível. A seleção das fontes de pesquisa baseou-se em publicações de autores de reconhecida importância acerca do tema que é disruptivo e inovador, onde foi possível o aprofundamento no tema de pesquisa e possibilitou um melhor entendimento do arcabouço do controlador de rede *OpenFlow* empregado como solução no plano de controle, para construção de um ambiente de rede real *OpenFlow* para experimentação e para o desenvolvimento dos mecanismos de proteção desenvolvidos no contexto do trabalho, para atender aos objetivos de pesquisa propostos.

b) Pesquisa descritiva através de um estudo de caso aplicado em um ambiente de rede real *OpenFlow/SDN* para experimentação, proposta para dissertação de mestrado em sistemas de computação, tendo como foco avaliar e validar os mecanismos de proteção implementados. O estudo de caso foi limitado ao desenvolvimento de mecanismos de detecção e reação apresentados no contexto deste trabalho, descrevendo e comprovando a viabilidade da proposta. Como resultado prático o trabalho serve como referência e as soluções e estratégias da pesquisa podem ser empregadas para elaboração de trabalhos futuros nas áreas de graduação e pós-graduação na Universidade de Salvador (UNIFACS), no contexto do projeto FIBRE, em substituição a *softwares* de simulação que por própria natureza possuem limitações.

1.3 ESTRUTURA DESTA DISSERTAÇÃO

A dissertação está estruturada em seis capítulos: O primeiro capítulo apresenta os objetivos gerais e específicos do trabalho, trabalhos relacionados e a metodologia adotada na dissertação.

No segundo capítulo são detalhados os conceitos e componentes empregados na aplicação e implementação do paradigma das redes SDN, essenciais para o entendimento do princípio de funcionamento da arquitetura de rede *OpenFlow*. Ainda no segundo capítulo são descritas as diferentes abordagens (ambientes simulados, emulados e reais) disponíveis no contexto de uma pesquisa científica para se avaliar novas propostas de implementação baseado no protocolo *OpenFlow*.

O terceiro capítulo tem por finalidade apresentar uma visão geral de segurança da informação, definir conceitos de vulnerabilidades e ameaças e destaca a importância da implementação dos controles de segurança da informação nas redes de computadores com o objetivo de diminuir os impactos dos incidentes de segurança da informação. Neste capítulo também são abordados a importância dos IDPS como mecanismos de proteção de um ambiente de redes, bem como o detalhamento das funcionalidades da ferramenta Snort que foi empregada no contexto deste trabalho para viabilizar a proposta de pesquisa. Ao final do capítulo são detalhadas as vulnerabilidades mais conhecidas e que podem comprometer um projeto de construção de uma rede SDN e é apresentado, em linhas gerais, o cenário atual dos ataques de redes no Brasil, buscando sensibilizar o leitor da importância de se implementar mecanismos eficazes de proteção em qualquer projeto de redes, desde sua concepção.

O quarto capítulo consiste na apresentação da proposta do trabalho, onde são detalhados os aspectos de implementação do Ambiente para Detecção e Prevenção Flexível de Ataques em Redes *OpenFlow/SDN*, que contempla um ambiente de rede real em ambiente de *Testbed* e mecanismos de monitoramento e tratamento flexível de eventos de segurança aplicado às redes SDN, por meio da programabilidade do controlador, estendendo as funcionalidades do arcabouço do controlador de rede *OpenFlow*. Nesse sentido, o capítulo busca identificar e detalhar tanto os componentes de rede utilizados para construção do ambiente de

rede, quanto os blocos de código no nível de controlador, desenvolvidos para atender aos objetivos da dissertação e a estratégia adotada para permitir dar flexibilidade ao monitoramento e tratamento de eventos de segurança.

No quinto capítulo são apresentados os resultados da análise, avaliação e validação da proposta. Para isso inicialmente são apresentados os resultados dos testes aplicados no ambiente de rede *OpenFlow/SDN*, visando demonstrar como o mesmo é funcional e está em condições de servir como instrumento para o pesquisador avaliar e validar as implementações feitas no nível do plano de controle para atender os objetivos do trabalho. Em um segundo momento, o capítulo 5 apresenta os resultados dos testes aplicados no ambiente de rede real *OpenFlow/SDN*, por meio de simulação e baseados em cenários, com a finalidade de demonstrar a eficácia das diferentes estratégias de tratamento de intrusão e ataques de rede adotadas neste trabalho.

Já no sexto capítulo são apresentadas as conclusões do trabalho e recomendações para pesquisas futuras. Este trabalho é finalizado com as referências consultadas para a elaboração da dissertação e a construção da do Ambiente para Detecção e Prevenção Flexível de Ataques em Redes *OpenFlow/SDN*.

2 O PARADIGMA DAS REDES DEFINIDAS POR SOFTWARE

Neste capítulo será apresentado o conceito de SDN, seu princípio de funcionamento e será detalhado as funcionalidades de cada um dos diferentes componentes que compõem uma rede *OpenFlow/SDN*, destacando a importância exercida pelo controlador da rede no plano de controle. Também são descritas as diferentes abordagens disponíveis para se avaliar novas propostas de implementação baseado no protocolo *OpenFlow*.

2.1 VISÃO GERAL

Algumas das primeiras pesquisas relacionadas ao paradigma das redes SDN surgiram em 2008 nas Universidades de Stanford e da Califórnia em Berkeley - EUA, visando uma concepção que fosse aberta e disponível para toda comunidade de pesquisadores no contexto de novas soluções e tecnologias para área de redes de computadores (MCKEOWN et al., 2008). A proposta das redes SDN consiste fundamentalmente em desacoplar o plano de dados do plano de controle. Nas redes SDN é introduzido um novo componente na arquitetura de rede chamado controlador, que passa a ser responsável por definir as ações que devem ser tomadas pelos elementos comutadores (switches, roteadores, pontos de acesso sem fio, outros) no encaminhamento de pacotes (KREUTZ et al. 2015). A proposta de rede seguindo o paradigma SDN oferece uma perspectiva de "programabilidade" via o controlador no plano de controle e uma visão global da rede.

A ideia chave do desacoplamento é prover uma maior flexibilidade para a programação das funções de controle, mantendo o hardware especializado para comutar quadros em alta velocidade. Logo, as redes SDN oferecem uma alta programabilidade das funções de controle da rede usando comutadores com alto desempenho para o encaminhamento de quadros. Numa outra perspectiva, a ideia consiste poder definir de maneira simples os fluxos de dados e as ações sobre estes fluxos através de uma interface de programação de aplicação (MATTOS et al., 2013).

As redes SDN se caracterizam por conceber uma nova arquitetura para redes de computadores que permite a implementação de funcionalidades sem a

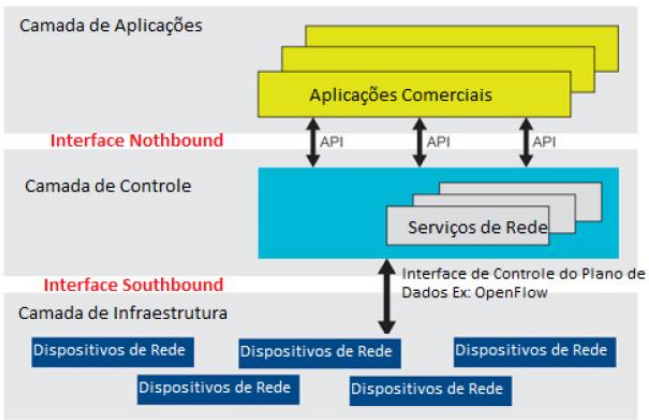
necessidade de intervenção do fabricante do equipamento e com alto grau de flexibilidade, já que estas funcionalidades podem ser implementadas no software do controlador (DURR, 2012). Este é um dos motivos que fazem das redes SDN um novo paradigma de redes, já que em equipamentos legados, a execução do plano de controle é toda feita no próprio equipamento, por meio de protocolos implementados em um sistema operacional embarcado, que não permitem que suas funcionalidades sejam estendidas além daquelas originalmente pensadas no seu projeto original (OLIVEIRA et al., 2012).

Neste sentido, o paradigma SDN facilita a implantação de novos protocolos e aplicações, simplifica e flexibiliza o gerenciamento da rede e possibilita múltiplas opções de controle quanto aos seus fluxos de tráfego, dentre outros, sendo amplamente favorável à inovação e evolução da rede, de modo geral.

2.2 O PARADIGMA SDN

A *Open Networking Foundation* (ONF) tem liderado o esforço para padronização do paradigma SDN e tem se dedicado a elaborar padrões e difundir o uso da tecnologias associadas ao SDN. Como ilustrado na Figura 1, o paradigma SDN, de acordo com a ONF (2014a), pode ser visualizado em uma estrutura de três camadas: a camada de aplicação, camada de controle e camada de infraestrutura.

Figura 1 - Separação dos elementos do paradigma das SDN



Fonte: ONF (2014a).

A camada de aplicação disponibiliza aplicações que podem introduzir novas funcionalidades de rede, tais como segurança, gerenciamento e novas formas de

encaminhamento de pacotes. A camada de aplicação tem uma visão abstrata e global da rede. A interface *Northbound* é responsável por controlar a interação entre o controlador, tipicamente um sistema operacional de rede, e as aplicações.

A camada de controle de uma rede SDN desacopla o controle (inteligência da rede) da camada de infraestrutura composta por equipamentos de comutação. Tipicamente, a camada de controle é realizada por um sistema operacional de redes (*Network Operating System* - NOS), que basicamente fornece uma interface para o desenvolvimento de novas aplicações, tornando o ato de programar uma rede SDN independente do fornecedor do equipamento de comutação. O sistema operacional de redes é responsável por fornecer abstrações para a a coleta de estado da rede, visão global e facilitar a programação dos dispositivos de encaminhamento por meio de uma lógica centralizada.

A camada de infraestrutura também é chamada de camada de encaminhamento ou simplesmente plano de dados. Ela é composta pelos dispositivos de rede, sejam eles físicos ou virtuais, e tem por responsabilidade encaminhar os pacotes, ou seja, escoar o tráfego da rede e manter a conexão com o controlador. Tal comunicação pode ou não utilizar um canal seguro de comunicação e é usualmente implementado por *Secure Socket Layer* - SSL ou por *Transport Layer Security* - TLS.

A interface *Southbound* ou API Sul é usada para realizar a conexão das camadas de controle com a camada de infraestrutura. Fundamentalmente, ela envia as instruções da camada de controle para a camada de encaminhamento. A API *Southbound* mais amplamente adotada é a definida pelo protocolo OpenFlow (ONF, 2014b) e a ONF tem sido responsável por sua padronização e o OpenFlow está atualmente na sua versão 1.5.1. Alguns outros organismos internacionais como a *Internet Engineering Task Force* (IETF) e *International Telecommunications Union* (ITU-T) também elaboraram documentos descrevendo a arquitetura SDN e que são similares ao que foi proposto pela ONF.

Mesmo sendo considerado um conceito novo, o paradigma SDN tem adquirido boa atenção da academia e do próprio mercado, se destacando como uma tendência a ser adotada para os próximos anos e tem grande espaço de desenvolvimento em pesquisa. A influência dessa tendência é ilustrada pela convenção Open Networking Summit (ONSUMMIT, 2012), realizada em abril de

2012 em Santa Clara, Califórnia, cujo tópico principal foi discutir a influência das redes SDN no estado atual das redes de computadores e apresentar ideias para o futuro.

2.3 REDE SDN COM OPENFLOW

Um dos protocolos amplamente usados na implantação das redes SDN é o *OpenFlow,* que é aberto e padronizado para a comunicação entre controladores e equipamentos de comutação (MCKEOWN et al. 2008). No entanto, convém ressaltar que o paradigma SDN e o protocolo *OpenFlow* não são sinônimos, na medida em que as redes SDN, como paradigma, definem a proposta de separação do plano de dados do plano de controle enquanto que o *OpenFlow* define as mensagens trocadas e procedimentos decorrentes, realizados entre o software controlador (como plano de controle) e os comutadores (como plano de dados).

Os principais componentes de uma rede SDN com protocolo *OpenFlow*, ilustrados na Figura 2, são descritos abaixo:

a) Tabela de Fluxo: Cada entrada na tabela de fluxo consiste, minimamente, dos campos de "fluxo", "ação" e "contadores" e contém, minimamente, uma ação associada ao fluxo definido. Os campos de fluxo da tabela de fluxo são utilizados para definir as ações, ou seja, como os pacotes devem ser processados e para qual destino devem ser encaminhados. Já os "contadores" são empregados para suportar o computo de diversas estatísticas referentes aos fluxos manipulados pelo equipamento comutador (FARIAS et al., 2011, p. 20). As entradas da tabela de fluxo podem ser interpretadas como decisões em cache (*hardware*) do plano de controle (*software*), sendo, portanto, a unidade mínima de informação (controle) no plano de dados da rede.

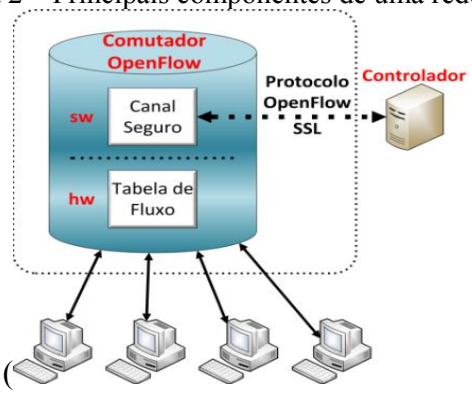
b) Canal seguro: O canal seguro é a interface que permite a comunicação entre o comutador e o controlador da rede utilizando uma conexão criptografada e que permite que mensagens *OpenFlow*. O canal *OpenFlow* usa normalmente criptografia Transport Layer Security - TLS, porém pode também executar diretamente sobre TCP (ONF, 2013a, p. 25).

c) Protocolo *OpenFlow*: Define um padrão aberto de comunicação para a troca de mensagens entre controlador e os equipamentos de uma rede. O protocolo permite a programação da tabela de fluxo do equipamento por meio interface

Southbound ou API Sul e possibilita que desenvolvedores e pesquisadores na área de redes implementem novos recursos e soluções sem se preocupar com as características do equipamento (abstração do *hardware*).

d) Controlador: Segundo Rothemberg et al. (2010), o controlador *OpenFlow* atua gerenciando e realizando o controle da rede, e oferecendo um ambiente baseado na reutilização de componentes e em diferentes níveis de abstração. Em outras palavras, o "controlador" controla a operação da rede abstraindo o encaminhamento e a especificação do equipamento comutador. Segundo Kreutz et al. (2014) um controlador SDN atua como um intermediário entre as aplicações e os elementos de redes, oferecendo uma plataforma com base na reutilização de componentes e na definição de níveis de abstração, por meio de comandos da API, permitindo que novas aplicações de rede possam ser desenvolvidas.

Figura 2 – Principais componentes de uma rede SDN



Fonte: Mckeown et al. (2008).

Conforme propõe Kreutz et al. (2014), este trabalho segue a definição das redes SDN baseada em quatro pilares principais:

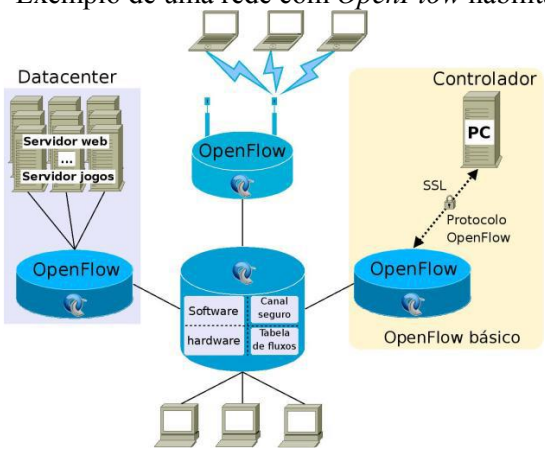
a) O plano de dados e de controle estão desassociados.
b) A lógica de controle é movida para uma entidade externa, denominada de controlador.
c) As decisões de encaminhamento são baseadas em "fluxos" (*Flow-based*) definidos por "tuplas" de campos dos cabeçalhos dos quadros, pacotes, mensagens e protocolos de aplicação, ao invés de utilizar somente o endereço destino (*IP-based*).
d) A rede é programável por meio de serviços baseado em aplicações (*app*) que executam no controlador.

2.4 O PROTOCOLO OPENFLOW

Segundo Benton et al. (2013), *OpenFlow* é a definição de um protocolo para comunicação com um dispositivo de rede e controle do funcionamento do seu plano de dados (*datapath*). Trata-se de um protocolo de baixo nível que permite implementar o controle dos nós da rede por meio da definição de uma interface externa para a troca de mensagens entre os dispositivos de rede e os controladores, como ilustrado na Figura 3. Essas mensagens podem ser simétricas (*hello, echo vendor*), assíncronas (*packet_in, flow removed, port*, um novo fluxo ou, em função do tipo de pacote e *status*, *error*) ou, ainda, iniciadas pelo controlador (*features, configuration, modify state, send packet, barrier*).

Por ser baseado em um padrão aberto busca a interoperabilidade entre diferentes fabricantes e permite controlar, gerenciar e monitorar equipamentos que suportem esse protocolo, com decisões de encaminhamento dos pacotes sendo feitas de maneira centralizada, possibilitando a construção de uma rede programada, independentemente dos comutadores de rede.

Figura 3 – Exemplo de uma rede com *OpenFlow* habilitado



Fonte: Rothenberg et al. (2010).

A principal abstração utilizada na especificação *OpenFlow* é o conceito de fluxo (ROTHENBERG et al., 2010, p. 66) que define tráfegos de rede baseado em regras predefinidas, configuradas de forma estática ou dinâmica por um controlador SDN (MCKEOWN, 2008). As regras e ações instaladas no comutador são responsabilidade do controlador, que pode ser implementado em um servidor comum, conforme Figura 3. Com *OpenFlow* habilitado nos comutadores, pesquisadores e administradores podem desenvolver novas aplicações que

interajam com as tabelas de fluxos visando avaliar novos protocolos e soluções de rede.

A solução, seguindo a abordagem *clean slate*[1], tem se destacado no âmbito das redes programáveis por software, fornecendo meios para validação de soluções alternativas para a Internet do Futuro (GOMES et al., 2013). Ao longos dos últimos anos o paradigma SDN, por meio do protocolo *OpenFlow*, tornou-se a plataforma de referência no desenvolvimento e em pesquisa (MACEDO et al. 2015) e cada vez mais vem sendo adotado de forma contínua pela indústria (FEAMSTER et al. 2014). Grandes players do mercado como HP, NEC, Pronto, Extreme, Cisco, Brocade, Juniper e Huawei estão implementando novas soluções com suporte ao protocolo *OpenFlow*.

Ballard et al. (2010) destaca como um outro benefício da especificação do protocolo a possibilidade de permitir que a programação da rede possa ser feita por operadores e fornecedores de software independentes e por usuários utilizando ambientes de programação comuns e não fique limitado aos fabricantes de equipamentos. Esta características abrem perspectivas para novas oportunidades quanto a geração de receitas, aumenta a confiabilidade da rede e da segurança, assim como estabelece uma visão e gestão centralizada da rede, sujeita a menos erros de configuração.

2.5 COMUTADOR OPENFLOW

O Comutador *OpenFlow* é um equipamento de rede que funciona de forma idêntica a um equipamento de uma rede convencional, realizando o encaminhamento dos pacotes no plano de dados, de acordo com o que estiver predefinido na tabela de fluxo. As entradas da tabela de fluxos podem ser interpretadas como decisões em cache (*hardware*) do plano de controle (*software*), sendo, portanto, a mínima unidade de informação no plano de dados da rede.

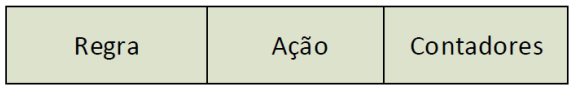
Segundo Mckeown (2008) o protocolo *OpenFlow* foi concebido a partir da premissa de que a maioria dos comutadores possuem tabelas de fluxos que

[1] A abordagem *Clean Slate* para a Internet é o mesmo que "reinventar a Internet, incorporando novas tecnologias, permitindo a criação de novas classes de serviços e aplicações, bem como, ser uma plataforma que suporte inovações e atenda as demandas atuais e futuras "

possuem basicamente um conjunto de funções que podem ser executadas em diferentes dispositivos de rede, mesmo estes equipamentos tendo implementações específicas definidas por cada fabricante.

Como ilustrado na Figura 4, uma tabela de fluxos é composta por um campo de *match* ou regra, de ação ou instrução e de contadores. Um comutador *OpenFlow* pode ter uma ou mais tabelas de fluxo, onde cada tabela de fluxo contém múltiplas entradas de fluxo.

Figura 4 - Principais campos de uma entrada de fluxo



Fonte: ONF (2013).

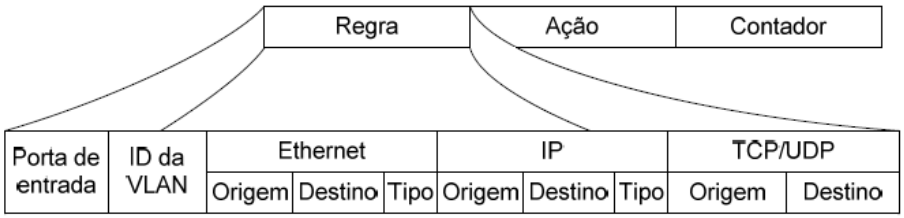
O campo de regra ou *match* é usado para verificar em qual entrada de fluxo um determinado pacote analisado se enquadra, sendo possível, por exemplo, extrair campos de cabeçalho do pacote, seja de camada 1, 2 ou 3 da arquitetura TCP/IP, por meio de comparações específicas com os valores do campo de *match* de cada entrada de fluxo, em ordem de prioridade de fluxo, até que uma regra seja idêntica aos valores extraídos do pacote. Segundo Kreuz (2014) a criação de regras em redes *OpenFlow* pode ser de forma proativa, no qual regras são inseridas em comutadores antes que sejam necessárias ou reativo, no qual regras são inseridas em comutadores em resposta a pacotes observados pelo controlador, através de mensagens de chegada de pacotes. As entradas na tabela de fluxo, em um comutador *OpenFlow,* relacionam o fluxo com um conjunto de ações (instruções) definidas pelo controlador de como o comutador deve tratar cada pacote que tenha as características definidas no fluxo.

Um determinado pacote ao chegar a um comutador *OpenFlow* é comparado com as entradas de fluxo contidas na tabela. Se houver uma entrada em que uma das regras coincida com o padrão de pacote, uma ação será executada. Dentre as ações possíveis, estão a fazer o encaminhamento de um pacote em uma determinada porta de saída, descartar o pacote ou modificar um determinado campo do cabeçalho do pacote. Por fim cada fluxo possui contadores associados a ele, e cada contador registra dados do fluxo descrito, como quantidade de dados transmitidos, duração do fluxo e quantidade de pacotes transmitidos.

A Figura 5 apresenta os principais campos de cabeçalho em uma tabela de fluxos definidos pela ONF. As tuplas podem ser formadas por campos das

camadas de enlace, de rede ou transporte, segundo o modelo TCP/IP (ROTHENBERG et al., 2010, p. 66).

Figura 5 – Detalhe dos campos de cabeçalho de uma tabela de fluxo



Fonte: Adaptado de Mckeown et al. (2008).

Os campos de cabeçalho (características) descrevem que tipos de pacotes pertencem àquele fluxo, ou seja, a descrição de um fluxo, que reúne características de várias camadas de protocolo, possibilitando que as regras de encaminhamento não fiquem restritas apenas a endereços de rede (endereço IP) ou endereços físicos (endereço MAC) (COSTA, 2013).

2.5.1 Implementações do plano de dados (*datapath*)

Existem variadas soluções para se trabalhar com plano de dados (*datapaths)* baseados em *hardware* ou *software*. A implementação em *hardware* normalmente fornece melhor desempenho, uma vez que pesquisas e encaminhamento de pacotes são realizados por circuitos especializados para executar essas tarefas, enquanto o implementação em software, essas tarefas seriam executadas pelo processador que está executando o switch. No entanto, as implementações em software têm a vantagem de permitir o uso de um computador convencional, bem como diferentes sistemas operacionais (CASADO, 2014), e se for necessária uma atualização do comutador bastaria atualizar o ambiente para uma nova versão do software.

Algumas soluções de *datapaths* baseados em software, que aceitam o protocolo *OpenFlow,* são Open vSwitch (OVS) (PFAFF et al., 2009), mais usado para pesquisa e pelo mercado, e o *OpenFlow* 1.3 *Software Switch* (ofsoftswitch13) (FERNANDES; ROTHENBERG, 2014), que caracteriza-se por ser um comutador brasileiro e pouco empregado com relação ao primeiro. Segundo Rezende (2016) os comutadores virtuais atualmente tendem a ser mais precisos e robustos que os

físicos, além de que a maioria deles são *open-source*, suportando o avanço das pesquisas. O Mininet (LANTZ et al., 2010), como um ambiente de virtualização no nível de sistema operacional, seria uma alternativa para implementar redes com diferentes topologias. Segundo Kreutz et al. (2014) os comutadores gerados pelo Mininet incluem o Open vSwitch e o ofsoftwitch13.

Com relação as soluções baseadas em hardware, que foi a adotada nesta dissertação, uma delas faz uso de interfaces NetFPGA[2] (NAOUS et al., 2008) combinadas a uma plataforma com arquitetura x86, que disponibiliza uma solução versátil e permite ao programador ter total controle sobre o funcionamento do *datapath*. Outras soluções de *datapath OpenFlow,* baseadas em hardware, já foram lançadas por grandes *players* do mercado que produzem comutadores dos mais variados portes. São exemplos de fabricantes, a HP, Cisco, Juniper, Brocade etc. Neste tipo de solução, tem-se uma dependência das alterações realizadas pelos fabricantes nos *firmwares* de seus equipamentos para suportar o protocolo.

Uma terceira solução, de menor custo, combina, de certa maneira, vantagens e desvantagens das soluções anteriores, fazendo uso da implementação do Linux OpenWRT, que é uma distribuição voltada para sistemas embarcados, tradicionalmente usada em roteadores sem fio (OPENWRT, 2016). Uma vez instalado o OpenWRT, este pode ser modificado de maneira a tornar o roteador sem fio num *Switch OpenFlow* de baixo custo. Esta foi a solução adotada nesta dissertação, que empregou o *hardware* do TP-LINK WR1043ND, que é um roteador sem fio homologado pela OpenWRT.org e pelo (CONSORTIUM, 2012).

Com relação ao método de tratamento de tráfego *OpenFlow* entre os comutadores e o controlador é possível implementar dois métodos gerais. Estes métodos são identificados como controle *in-band* e *out-of-band*, onde a diferença básica entre estas duas implementações é que no controle *out-of-band* os comutadores se comunicam com o controlador por caminhos de redes separados da rede de dados e por isso não são afetados pelas operações de *OpenFlow* (HU et al., 2014).

[2] Plataforma aberta que combina um FPGA (field-programmable gate array, ou arranjo de portas programável em campo) memória (SRAM e DRAM) e processadores de sinais numa placa com quatro portas Ethernet de 1 Gbps ou 10 Gbps, que permite desenvolvedores programar o FPGA e utilizar a memória para implementar novos mecanismos, protocolos e arquiteturas.

2.6 CONTROLADOR OPENFLOW

Segundo Rothemberg et al. (2010) o controlador atua como um sistema operacional de redes agindo sobre o plano de controle, oferecendo uma plataforma com base na reutilização de componentes e na definição de níveis de abstração, permitindo a comunicação com os dispositivos de encaminhamento através da API *southbound* e oferecendo serviços às aplicações através da API *northbound*, permitindo que novas aplicações de rede possam ser desenvolvidas. No entanto, os controladores não são aplicações per si, pois sua função principal consiste apenas em disponibilizar mecanismos de comunicação entre as aplicações e o plano de dados, ou seja, sem aplicações não é possível orquestrar uma rede SDN (KREUTZ, 2014).

Neste sentido é que Kreutz et al. (2014) classifica um controlador SDN como sendo muitas vezes uma plataforma de orquestração que atua como um intermediário entre as aplicações e os dispositivos de rede e que permite um certo grau de programabilidade da rede por meio do desenvolvimento de "plug-ins" entre a função de orquestração e os protocolos de rede. Ao ser responsável pela configuração de todos os dispositivos na rede, pela manutenção das informações da topologia e o pelo monitoramento do estado da rede (RODRIGUES, 2014), o controlador possui uma visão global e gerência centralizada de toda a rede e não se limita às funcionalidades e recursos básicos de um comutador convencional.

As aplicações que são executadas no controlador determinam a maneira como os fluxos irão se comportar na rede. O controlador se comunica com os dispositivos por meio do protocolo *OpenFlow* e cada nó da rede se comunica com o controlador solicitando instruções de como deve proceder a cada vez que o dispositivo não saiba como atuar sobre um determinado fluxo.

A programabilidade no nível do plano de controle permite a evolução em paralelo das tecnologias nos planos de dados e as inovações na lógica das aplicações de controle, sendo óbvio que ele precisa implementar o protocolo *OpenFlow* para se comunicar, por meio de mensagens, com os elementos encaminhadores enviando comandos para a rede. As mensagens enviadas pelo controlador (ONF, 2013, p. 26) podem ou não necessitar de resposta do elemento

encaminhador e são classificadas de acordo com os seguintes tipos apresentados na Tabela 1.

Tabela 1 - Tipos de mensagens enviadas por um controlador *OpenFlow*

Tipo de Mensagem	Descrição
Mensagens de características	Geralmente ocorrida após o estabelecimento do canal entre o comutador e o controlador *OpenFlow*, o controlador solicita ao comutador o pedido de conexão, bem como as características especificas e as capacidades do mesmo para que o controlador possa gerenciar melhor a rede.
Mensagens de configurações	O controlador é capaz de definir e consultar parâmetros de configuração no comutador. O comutador responde a solicitação de consulta do controlador.
Mensagens de modificações do estado	São mensagens enviadas pelo controlador para gerenciar os estados dos comutadores. Seu principal objetivo é adicionar ou remover entradas nas tabelas de fluxo dos comutadores, ou seja, adicionar e remover regras.
Mensagens de leitura do estado	São mensagens enviadas pelo controlador para ler os status e coletar as estatísticas (contadores) do controlador.
Mensagens de packet-out	Essas mensagens são usadas para enviar pacotes completos para o comutador. Quando não há uma correspondência entre o pacote e alguma entrada na tabela de fluxos do comutador, o pacote então é encaminhado, por completo, para o controlador (*Packet-In*), que este por sua vez reenvia, utilizando o *Packet-out*, o mesmo pacote com seu conjunto de ações a ele associado.
Mensagens de barreira	São mensagens utilizadas pelo controlador para garantir se a dependências de mensagens foram cumpridas ou para receber notificações de operações concluídas.

Fonte: ONF, (2013, p. 26).

Existem diversos tipos de controladores *OpenFlow/SDN*, como o Floodlight[3], NOX[4], OpenDayLight[5], POX[6], RYU[7], dentre outros. Basicamente

[3] Floodlight e um controlador *OpenFlow* desenvolvido em JAVA que é patrocinada pela *Big switch Networks*. <http://www.projectfloodlight.org/ floodlight/>

realizam a mesma função, suportam a versão do OpenFlow 1.0, 1.1, 1.2, 1.3 ou 1.4, e a diferença principal entre eles é que são codificados em diferentes linguagens de programação, como Java, C++, Python e C (KREUTZ et al., 2014).

2.7 AVALIAÇÃO DO PROTOCOLO *OPENFLOW*

Para a criação de uma rede seguindo o paradigma das SDN são necessários diferentes componentes em sua infraestrutura. Um dos componentes é o equipamento servidor que vai atuar como controlador executando algum sistema operacional de rede, como por exemplo, NOX, POX ou o Ryu. Também são necessários os elementos encaminhadores com seu plano de controle separado do plano de dados e implementando algum tipo de interface de programação, como por exemplo, o *OpenFlow*. E por último o canal seguro com o objetivo de assegurar a criação de uma SDN, possibilitando a interligação entre os elementos encaminhadores e o controlador. No entanto, nem sempre um pesquisador dispõe de uma arquitetura de rede contendo todos os componentes necessários para construção de uma rede OpenFlow/SDN necessários para viabilizar seus experimentos, principalmente porque essa infraestrutura possui um custo associado relativo à aquisição de equipamentos como servidores de rede, estações de trabalho, *Switches* com suporte ao *OpenFlow* e cabeamento de rede.

A prática de experimentação em redes reais acaba se tornando algo impraticável, seja pelo fato dos possíveis riscos que podem trazer a uma infraestrutura de rede em produção, seja pelo fato de que a construção de um laboratório envolve a aquisição de equipamentos de alto valor (MARCHESAN e MEDINA, 2015). Visando tratar a falta de disponibilidade de um ambiente real para validação de uma determinada pesquisa, os pesquisadores podem abrir mão de abordagens que empregam ambientes simulados ou até mesmo emulados,

---

[4] NOX é uma controlador Openflow para gerenciamento de redes de grande escala desenvolvido em PYTHON/C++ pela NICIRA, sendo um dos primeiros controladores OpenFlow. <https://github.com/noxrepo/nox>

[5] OpenDayLight é um controlador que suporta o OPNFV, que é um projeto de código aberto voltado para a evolução do NFV através de uma plataforma integrada e aberta. <https://www.opendaylight.org/lithium>

[6] POX é uma controlador Openflow para gerenciamento de redes de grande escala desenvolvido em Python pela NICIRA. <http://www.noxrepo.org./pox/about-pox/>

tornando possível a avaliação ou validação de implementação de um novo recurso, serviço ou protocolo aplicados às redes OpenFlow/SDN.

2.7.1 Ambientes Simulados

A utilização de ambientes de simulação vem aumentando de forma significativa uma vez que estes permitem o estudo e a avaliação de sistemas a custos reduzidos. Os simuladores de rede desempenham um papel importante na tarefa de desenvolver, analisar e aperfeiçoar protocolos de comunicação. Segundo (NUNO et al., 2005) destacam-se como três importantes vantagens do uso de simulação o fato de permitir a avaliação e teste do comportamento dos protocolos em diversas redes e ambientes, cuja preparação num laboratório ou em uma empresa poderia ser impraticável e com alto custo, o fato de facilitar a execução de testes em um ambiente controlado, onde em teste é mais fácil fazer variar parâmetros relevantes, mantendo os restantes parâmetros constantes e por último, mas não menos importante, o fato de facilitar a execução dos protocolos em múltiplos cenários de execução.

Segundo Pedgen et al. (1995) a simulação é uma das ferramentas disponíveis para análise de desempenho de sistemas computacionais a ser empregada nas mais diversas áreas de aplicação, principalmente devido à sua versatilidade, flexibilidade e baixo custo.

Existem diversos simuladores de rede disponíveis, porém poucos são os simuladores que oferecem suporte ao protocolo *OpenFlow*. O NS-3[8] (*Network Simulator 3*) e o EstiNet[9] são exemplos de simuladores com suporte ao protocolo *OpenFlow*. No entanto, Marchesan e Medina (2015) destacam que tanto o NS-3 quanto o EstiNet possuem algumas limitações que tornam difícil seu emprego e utilização.

---

[7] RYU é uma controlador baseado na linguagem Python extremamente flexível que possui um amplo suporte ao OpenFlow (1.0, 1.2, 1.3, 1.4, 1.5 e as extensoes Nicira). <https://osrg.github.io/ryu/>

[8] https://www.nsnam.org/

[9] http://www.estinet.com/

2.7.2 Ambientes Emulados

Segundo Conterato et al. (2013), a utilização de um ambiente emulado para prototipação em SDN do protocolo *OpenFlow* é uma prática atualmente comumente usada, com destaque para a ferramenta Mininet (Lantz et al., 2010), por sua simplicidade de obtenção e configuração e por ser um ambiente de testes utilizado em pesquisas acadêmicas.

O Mininet é um sistema que simula a criação rápida de protótipos de uma rede em apenas um único computador, originando redes escaláveis definidas por software. Segundo Marcondes (2011), esses recursos permitem que a ferramenta Mininet venha interagir, personalizar e partilhar seus protótipos rapidamente, capaz de emular vários *links*, *hosts*, *switches* e controladores *OpenFlow* em um único *Kernel*. O Mininet dispõe de comandos para a realização de testes que são similares ao encontrado no terminal Linux. É possível também personalizar topologias utilizando a linguagem de programação Python (COSTA, 2013).

No entanto, o MiniNet ainda não fornece desempenho e qualidade fiéis a uma rede real, embora o código utilizado nele sirva para uma rede real baseada em placas NetFPGAs, ou *Switches* comerciais. Segundo Liberato et al. (2014) o uso do ambiente emulado torna-se um grande limitador para os projetistas em SDN validarem suas aplicações, seja porque o ambiente não garante um alto grau de fidelidade quando comparado a um ambiente real, seja porque pequenas alterações neste ambiente afetam drasticamente a latência e a taxa de transferências nas redes.

2.7.3 Ambientes Reais

Alguns *Switches* com suporte ao protocolo *OpenFlow* já são disponibilizados por alguns fabricantes para criação de ambientes e realização de testes, sendo possível até mesmo a alteração do *firmware* original de alguns equipamentos para um *firmware Opensource* chamado Indigo (INDIGO, 2013).

O projeto NetFPGA (*Network Field Programmable Gate Array*), como plataforma aberta em *hardware* programável, com módulo *OpenFlow* habilitado, permite a construção de protótipos de redes a serem empregados na área de

pesquisa. Segundo Lopes (2015) o projeto consiste no uso de placas PCI-e, com portas gigabit, conectadas a um PC baseado na plataforma Linux, possibilitando a implementação de novas funcionalidades. A proposta de Unnikrishnan et al. (2010) mostra as vantagens de se usar NetFPGA em redes virtuais, focando em escalabilidade.

No entanto, uma das soluções mais adequadas e que apresentam custo de aquisição e implementação relativamente baixo consiste em empregar o OpenWrt[10], que é um sistema operacional baseado na plataforma Unix, utilizado principalmente para dispositivos embarcados, para uso em ambientes reais. Esta foi a solução adotada no contexto desta dissertação e que possibilitou a concepção e construção de uma rede real *OpenFlow/SDN* para ambiente de experimentação, empregando um roteador WiFi de prateleira, da marca TP-Link, que teve seu *firmware* alterado para dar suporte ao protocolo *OpenFlow*. As principais características que serviram como fator de decisão para sua adoção foram apresentar custo relativamente baixo e desempenho satisfatório, exigir um mínimo requisito de espaço de armazenamento e de memória *Randon Access Memory* (RAM), que permite, por exemplo, que ele seja portado em equipamentos com espaço de armazenamento reduzido e sua modularidade, permitindo a criação de aplicações que podem ser encapsuladas em um formato de arquivo gerenciável baseado no *Open Package Management*[11] (opkg), que é uma variante do gerenciador de pacotes do Debian Linux.

[10] https://openwrt.org/
[11] http://code.google.com/p/opkg/

3 SEGURANÇA DA INFORMAÇÃO

Este capítulo tem por objetivo dar uma visão geral dos princípios de segurança da informação, definir conceitos tais como ameaças e vulnerabilidades, assim como detalhar a funcionalidade de um IDS/IPS e de uma *honeypot*, que serão essenciais para o entendimento da proposta do trabalho. Neste capítulo também são identificadas e detalhadas as vulnerabilidades inerentes às redes SDN, assim como é apresentado o cenário atual dos ataques de redes no Brasil, identificando o porquê da necessidade de se implementar mecanismos eficazes de proteção visando evitar o comprometimento das redes.

3.1 VISÃO GERAL

A NBR ISO/IEC 27002:2005 (ABNT, 2005), em sua seção introdutória, define segurança da informação como "a proteção da informação contra diferentes tipos de ameaças para garantir a continuidade do negócio, minimizar o risco ao negócio e maximizar o retorno sobre os investimentos e as oportunidades de negócio".

A segurança da informação, segundo Donner e Oliveira (2008), consiste em um processo de proteção das informações contra possíveis ameaças que possam comprometer sua integridade, disponibilidade e confidencialidade. Neste sentido, segurança da informação pode ser definido com uma área de conhecimento que visa à proteção da informação das ameaças a sua confidencialidade, integridade e disponibilidade, de acordo com as definições abaixo:

a) Disponibilidade: É a garantia de que a informação deve estar disponível, sempre que seus usuários (pessoas e empresas autorizadas) necessitarem, não importando o motivo. Em outras palavras, é a garantia que a informação sempre poderá ser acessada. Garante que uma informação estará disponível para acesso no momento desejado. Diz respeito à eficácia do sistema, ao correto funcionamento da rede para que quando a informação for necessária ela poderá ser acessada. A perda da disponibilidade se dá quando se tenta acessar uma informação e não se consegue o acesso esperado.

b) Integridade: princípio que destaca que a informação deve ser mantida na condição em que foi disponibilizada pelo seu proprietário, garantindo a sua

proteção contra mudanças intencionais, indevidas ou acidentais. Há perda da integridade quando a informação é alterada indevidamente ou quando não se pode garantir que a informação é a mais atualizada, por exemplo.

c) Confidencialidade: princípio que visa garantir que a informação não será conhecida por quem não deve. O acesso às informações deve ser limitado, ou seja, somente as pessoas explicitamente autorizadas podem acessá-las. Perda de confidencialidade significa perda de segredo.Garantir que a informação só será acessível por pessoas autorizadas. A confidencialidade ou privacidade se dá justamente quando se impede que pessoas não autorizadas tenham acesso ao conteúdo da mensagem. Refere-se à proteção da informação contra a divulgação não permitida e a perda da confidencialidade se dá quando alguém não autorizado obtém acesso a recursos ou informações.

As formas atuais de implementação dos mecanismos de segurança em sistemas de informação, a exemplo da criptografia, são utilizadas como prevenção ou solução para possíveis falhas em segurança, e são na ampla maioria dos casos notadamente técnicas, haja vista o fato de as iniciativas apresentadas se basearem em atualizações e sofisticações da tecnologia (MARCIANO et al., 2006, p. 93).

Estas falhas ou brechas de segurança são potencializadas pelo desenvolvimento de novas soluções e recursos na área de TI sem seguir as melhores práticas do mercado, seja na construção de sistemas de informação sem adoção de uma metodologia de desenvolvimento seguro, seja na implementação de sistemas operacionais ou serviços de rede sem aplicar práticas de *hardening*.

Além disso, segundo o Relatório Anual de Cibersegurança Cisco (2017), mais de um terço das instituições que lidaram com brechas de segurança em 2016 reportaram perdas substanciais na sua carteira de clientes, oportunidades e queda nas receitas em mais de 20%. Somado a estes fatores, grande parte das ferramentas e técnicas utilizadas para segurança da informação como antivírus e firewall não são suficientes para assegurar a segurança e ainda estima-se que 70% dos ataques a uma rede são oriundo de usuários internos (LYNCH, 2006).

Um mecanismo notadamente técnico, eficaz e alinhado à proposta desta dissertação consiste na implementação de mecanismos de monitoramento e tratamento de eventos de segurança para se buscar uma forma oportuna de se evitar possíveis comprometimentos nos serviços e recursos de tecnologia da

informação (TI). Além disso, as ameaças que podem vir a comprometer o pleno funcionamento de uma rede de computadores podem ser de diversas naturezas e geralmente são classificadas como ameaças passivas[12], ativas[13], maliciosas[14] e não maliciosas[15].

Uma das soluções disponíveis para tratar estas ameaças e disponibilizar mecanismos de monitoramento e tratamento de eventos de segurança consiste no emprego de IDPS que visam a detecção e prevenção de intrusão, com o objetivo de evitar que usuários mal intencionados violem a política de segurança definida e aplicada em uma rede de computadores. Neste contexto é que se insere a proposta da dissertação de conceber um ambiente de rede SDN com recursos e mecanismos implementados para reação e proteção às ameaças de segurança da informação e disponibilizar recursos técnicos para análise pós-ataque.

3.2 VULNERABILIDADES

A NBR ISO/IEC 27002:2005 (ABNT, 2005) define vulnerabilidade como uma fragilidade de um ativo ou grupo de ativos que pode ser explorada por uma ou mais ameaças. Ainda segundo Willie e David (2013) vulnerabilidade é uma lacuna, um erro ou fraqueza na forma como um sistema é projetado, usado e protegido.

Os tipos de vulnerabilidades, segundo Peixoto (2006), podem ser classificadas como:

a) Físicas: instalações prediais sem seguir as normas e legislações de construção vigentes, riscos de explosões ou incêndios, falta de dispositivos de controle de acesso às instalações, etc.

---

[12] Ameaça passiva é aquela que, quando realizada, não resulta em nenhuma modificação da informação do sistema ou do estado do sistema

[13] A ameaça ativa quando realizada com sucesso, provoca alteração da informação ou no estado de um determinado sistema.

[14] A ameaça maliciosa ou *intencional é* provocada por acesso indevido a sistemas, fraudes e roubo de informações.

[15] A ameaça não maliciosa ou i*nvoluntária é provocada muitas vezes* por erros de desconhecimento no uso de um ativo de informação, onde aparecem erros inconscientes de usuários que não foram devidamente treinados, infecções por vírus ou até mesmo os acessos indevidos.

b) Naturais: envolvem condições naturais, tais como locais propensos a inundações, ambientes sem a devida proteção contra incêndios, terremotos, furacões, etc.., que podem trazer riscos para os equipamentos e as informações.

c) Hardware: desgaste do equipamento, obsolescência ou má utilização.

c) Software: erros na instalação e configuração, vazamento de informações, perda de dados ou recursos indisponíveis.

d) Mídias: dispositivos em mídia que podem ser perdidos ou danificados.

e) Comunicação: acessos não autorizados ou perda de comunicação.

f) Humanas: muito relacionado a atitudes intencionais ou não, e podem ser identificados por meio de uso de senha fraca, compartilhar credencial de acesso com outras pessoas, falta de capacitação dos colaboradores, falta de consciência em aplicar as diretrizes de uma política de segurança da informação, descontentamento de colaboradores, etc.

A implementação de soluções e recurso de TI, seja em *hardware* ou *software,* seguindo procedimentos, configurações e técnicas específicas são empregados como medidas de segurança para atenuar as vulnerabilidades, com o intuito de reduzir a probabilidade de ocorrência da ação de ameaças e, por conseguinte, os incidentes de segurança (MOREIRA, 2001, p.31), até porque as vulnerabilidades por si só não provocam incidentes, mas são apenas consideradas janelas de oportunidade que se exploradas por um usuário mal intencionado, de forma intencional ou não, pode comprometer toda a infraestrutura de uma rede, um sistema ou um recurso computacional.

3.3 AMEAÇAS

Segundo a NBR ISO/IEC 27002:2005 (ABNT, 2005) ameaça é definida como causa potencial de um incidente indesejado, que pode resultar em dano a um sistema ou instituição. Ou seja, as ameaças se aproveitam de alguma falha de segurança da instituição, seu ponto fraco, para provocar algum tipo de dano ou prejuízo aos negócios de uma instituição.

Os tipos de ameaças, segundo Peixoto (2006), podem ser classificadas como:

a) Naturais: ameaças decorrentes de fenômenos da natureza, como por exemplo, raios que podem danificar equipamentos, tempestades, umidades, terremotos, etc.

b) Voluntárias: ameaças causadas de forma proposital por um agente humano, como por exemplo, ações de um *cracker*, espiões, desenvolvedores de software malicioso de computador, ameaças persistentes, etc.

c) Involuntárias: ameaças inconscientes, quase sempre causadas pelo desconhecimento, erros ou acidentes, dentre outros.

Uma das primeiras medidas para tratar as ameaças consiste em identificar e eliminar os pontos fracos de um ambiente de TI. No entanto, este levantamento dentro de uma organização, cada vez mais dependente dos sistemas de informação e da internet para fazer negócios, está ficando cada vez mais desafiador para, que de forma eficaz, tenha condições de tratar toda e qualquer ameaça, tais como contaminação por *vírus/worm*, furto de informação confidencial, perda de informações, etc.

Este aspecto limitador, no entanto, não pode servir como justificativa para o fato das instituições não conseguirem implementar mecanismos eficazes que busquem a garantia dos princípios básicos de segurança da informação, até porque as instituições não estão propensas a aceitar uma interrupção em suas operações. Logo, torna-se necessário tratar as ameaças, de forma a evitar que uma vulnerabilidade seja explorada e venha a provocar um incidente de segurança, impactando diretamente no pleno funcionamento da infraestrutura de TI, na credibilidade e nas receitas de uma instituição.

3.4 INCIDENTE DE SEGURANÇA DA INFORMAÇÃO

Um Incidente de Segurança da Informação, segundo a NBR ISO/IEC 27002:2005 (ABNT, 2005) é indicado por um simples evento ou uma série de eventos de segurança da informação indesejados ou inesperados, que tenham grande probabilidade de provocar uma ameaça a segurança da informação e comprometer a continuidade dos negócios, seja causando obstrução ou erro na execução de um ou mais processos organizacionais, seja na obstrução ou erro decorre de dificuldades na ação dos agentes organizacionais humanos ou

computacionais ou ainda provocando a queda no desempenho de uma ou mais funções organizacionais.

Como demonstrado no levantamento feito por Bannwart apud Peixoto (2006), os próprios funcionários, quando mal-intencionados ou mesmo quando manipulam os recursos e sistemas de informação de forma correta, podem servir como vetor de ameaça para a instituição. Neste sentido, as medidas de segurança implementadas em uma instituição não devem ficar limitadas a tratar os riscos inerentes apenas às ameaças externas, mas também devem contemplar tratamento adequado e eficaz as possíveis ameaças internas, seja por meio da elaboração e implantação de políticas de segurança da informação, no nível gerencial, seja por meio de implementação de um IDPS, no nível técnico, alinhados com o princípio da proteção em camadas, consagrado na literatura.

3.5 DETECÇÃO E PREVENÇÃO DE INTRUSÃO

Os IDS são utilizados para monitoramento do tráfego de dados de uma rede ou de um segmento de rede e permitem, por meio da análise de dados coletados na própria rede ou em uma base de dados disponível, identificar atividades consideradas maliciosas (AZEVEDO, 2012), sendo um elemento essencial na estratégia de defesa em profundidade e não apenas uma solução baseada em uma ferramenta ou tecnologia única na segurança de redes. Podem ser implementados com soluções baseadas em *software* ou *hardware* e buscam automatizar o processo de monitoramento e de análise de eventos considerados maliciosos, sendo considerado hoje um recurso necessário e essencial para segurança de qualquer infraestrutura de rede.

Os tipos de IDS, segundo Scarfonte e Mell (2007), são classificados como:

a) Sistema de detecção de intrusão baseado em rede (*Network Based Intrusion Detection System* - NIDS): geralmente posicionados em locais estratégicos da infraestrutura de uma rede, processam informações capturadas, analisando o fluxo de pacotes que trafegam pela rede. A detecção dos ataques conhecidos é realizada por meio da análise dos dados que emprega a comparação de padrões e assinaturas em um banco de dados ou pela detecção de atividades consideradas maliciosas.

b) Sistema de detecção de intrusão baseado em máquina (*Host Based Intrusion Detection System* - HIDS): geralmente implementados em servidores ou estações de trabalho e os dados são processados no nível de aplicação, onde processam informações de uma um ou mais sistemas, incluindo dados do sistema operacional e arquivos de programas. São mais eficientes na detecção de ataques de origem interna e menos eficientes para deter ataques de origem externos.

c) Sistema de detecção de intrusão híbrido (IDS híbridos): consistem no emprego das funcionalidades de ambas soluções de NIDS e HIDS, analisando o tráfego de rede, os *logs* e recursos do sistema no qual está instalado. À primeira vista parece conceber um IDS mais eficiente, mas nem sempre são sistemas melhores, principalmente porque ainda é um grande desafio conceber uma solução que consiga a interação destas diferentes tecnologia com sucesso e eficiência.

Os NIDS são componentes essenciais em um ambiente de rede em produção e sua capacidade de detectar diferentes intrusões de rede auxilia na proteção do ambiente, onde sua localização é um dos pontos a serem definidos com cuidado (NAKAMURA 2007, p. 264). As principais vantagens dos NIDS, conforme Kizza (2005) são:

a. Detecção e resposta em tempo real: possibilita, ao ser implementado em uma ponto estratégico da rede, a detecção de intrusões e notificação ao administrador, de forma oportuna, e praticamente em tempo real.

b. Capacidade de detectar ataques que os Sistemas de Detecção de Intrusão baseados em *Hosts* (*Host-based Intrusion Detection System* - HIDS) não conseguem: possibilita uma análise nos pacotes que circulam na rede no nível da camada de transporte, não ficando limitado apenas a endereços de rede.

c. Guarda das evidências: ao serem implementados em máquinas dedicadas e isoladas da rede, os NIDS permitem um armazenamento mais confiável e dificultam sobremaneira uma possível ação do atacante visando a remoção de evidências.

d. Baixo custo: disponibiliza mecanismos de detecção a intrusão, em uma rede corporativa, com o emprego de apenas alguns sensores em pontos estratégicos da rede (WANG, 2009). E por não haver tráfego de rede normal que esteja totalmente protegido de instrusão, a monitoração passiva é considerada outro ponto positivo.

Já o IPS complementa as funcionalidades de um IDS, adicionando uma reação a uma ameaça identificada, tal como o bloqueio de uma intrusão que impeça a ocorrência de um dano maior e provoque o comprometimento da própria infraestrutura de rede. Os tipos de IPS, segundo Evangelista (2008), são classificados como:

a) Sistema de prevenção de intrusão baseado em máquina (*host based IPS*): são ferramentas de prevenção de intrusão implementados diretamente nos computadores.

b) Sistema de prevenção de intrusão baseado em linha (*inLine IPS*): são ferramentas de prevenção de intrusão, implementados em soluções de *hardware* ou *software*, habilitados a detectar e impedir ataques maliciosos por meio da verificação de anomalias no tráfego de rede.

Segundo o Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (CERT.BR, 2012), um dos recursos essenciais para resposta e tratamento de incidentes consiste na disponibilidade de *logs* nos recursos computacionais, que possibilite uma efetiva análise de incidente pós-ataque e descoberta de diversas informações relevantes, tais como: a data-hora em que uma atividade malicioso ou não ocorreu, o fuso horário do *log*, o endereço IP de origem da atividade em análise, as portas de comunicação que originou a troca de informações e o protocolo utilizado no ataque (TCP, *Unified Datagram Protocol* - UDP, ICMP, etc.), os dados completos enviados para o computador ou rede e o resultado da atividade (se ela ocorreu com sucesso ou não). Nos IDPS, a geração de *logs* pode se dar tanto para casos de tentativa de invasão sem sucesso, por exemplo, erro de senha por três vezes, ou quando um intruso obtém sucesso na intrusão (BARBOSA, 2013).

Para questões de entendimento da proposta desta dissertação, convém diferenciar a funcionalidade de um IDS de IPS. O primeiro caracteriza-se por ser um *software* que automatiza o processo de detecção de intrusão sem tomar qualquer tipo de medida ou ação contra a ameaça, enquanto o segundo é um *software* de prevenção de intrusão, que tem por objetivo impedir possíveis ataques por meio de ações que permitam interromper um determinado ataque que esteja ocorrendo na rede. Em resumo, o primeiro trabalha de maneira reativa e informativa, enquanto que o segundo tem por objetivo diminuir o risco de

comprometimento de um ambiente ao reagir e tomar ações para mitigar as ações maliciosas na rede.

3.6 METODOLOGIA DE DETECÇÃO

As soluções de IDPS empregam diversificadas metodologias de detecção, sejam empregando uma abordagem isolada ou integrada, com a finalidade proporcionar uma detecção mais exata. As principais metodologias de IDPS, segundo Scarfone e Mell (2008), são classificadas como:

a) Baseado em assinaturas (*signature-based*): este tipo de IDPS aplica uma técnica onde padrões conhecidos são usados para identificar e detectar atividades maliciosas. Boa parte dos IDPS de mercado usam a combinação de IDPS baseados em redes e assinaturas.

Este tipo de IDPS apresenta como vantagem o fato de não consumir tantos recursos como o método baseado em anomalias, o seu processamento de padrões de ataques ser otimizado e a possibilidade dos falsos positivos serem controlados apenas com ajustes (MANDUJANO, 2004). Entre as desvantagens se destacam a necessidade do operador, responsável por criar os padrões de ataques, possuir um bom nível de conhecimento, o fato de um padrão de ataque acabar gerando um grande número de falsos positivos por reconhecer uma atividade normal como um tráfego malicioso e a necessidade de constantemente ter que atualizar a base de assinaturas (padrões de ataque) para mitigar os novos tipos de ameaças.

b) Baseado na detecção de anomalias (*anomaly-based detection*): também conhecido como uma IDS baseado em comportamento (*behavior based IDPS*), tem por objetivo detectar atividades que apresentam um comportamento incomum em uma máquina ou em uma rede. Este tipo de IDPS emprega perfis (*profiles*) que são construídos com dados históricos coletados do ambiente, ao longo de um determinado período de tempo, que são usados para medir e monitorar as atividades de rede, visando identificar se estas estão ou não fora do tipo de tráfego considerado padrão ou normal (BACE; MELL, 2001). Uma das principais características deste tipo de IDPS é que novos ataques podem ser identificados pelo simples desvio de comportamento, sem a necessidade de se

conhecer detalhadamente a intrusão. Como desvantagem, este tipo de IDPS pode gerar um grande número de falsos alertas, pois qualquer tipo de modificação no comportamento legítimo da máquina ou da rede pode ser considerado uma ação maliciosa; outra desvantagem, que pode ser considerada dificil e extensiva, seria o processo de coleta de dados inicial para criação do perfil do comportamento (BACE; MELL, 2001).

c) Análise de estados de protocolo (*stateful protocol analysis*): este tipo de IDPS observa e mantém estados do uso de protocolos usados através do sistema. Cada estado do protocolo é comparado contra os eventos observados para identificar um desvio (SCARFONE; MELL, 2007), e o resultado desta análise viabiliza a detecção de possíveis ações maliciosas. A desvantagem desta metodologia é que ela não é usada sozinha em um IDPS, mas sim deve ser implementado em conjunto com uma ou mais metodologias diferentes (SCARFONE; MELL, 2007).

Dentre os IDPS existentes no mercado, os mais conhecidos são:

a) Suricata[16]: um recente sistema de detecção e prevenção de intrusos, de código fonte aberto, desenvolvido pela Fundação de Segurança da Informação Aberta (*Open Information Security Foundation* - OISF) e lançado inicialmente em 2010. É um NIDS/NIPS, multiplataforma, com licença GPLv2, altamente escalável, que possui um mecanismo multi thread que permite inspecionar uma grande quantidade de tráfego de rede, com maior velocidade e eficiência na análise dos pacotes.

b) Prelude[17]: é um arcabouço desenvolvido pela CS Group caracterizado por atuar como IDS híbrido que pode interoperabilizar as funções de um NIDS com um HIDS, sendo em sua essência um NIDS. Tem a capacidade de encontrar ameaças na rede em conjunto com outros tipos de sensores.

c) Snort[18]: sistema de detecção de intruso baseado em rede, capaz de executar registros dos pacotes e a análise do tráfego de uma rede em tempo real. Possui um motor de detecção de ataques ou ameaças que permite registrar do o tráfego considerado malicioso e consegue executar análise do protocolo, faz

[16] https://suricata-ids.org
[17] http://prelude-ids.org
[18] https://www.snort.org

combinações que pode detectar uma variedade de ataques e probabilidades, como *port scan*, ataques CGI, ataques pelo Samba (*smb*) e vários outros.

Neste trabalho foi empregado como solução de IDS de rede a ferramenta *Snort*, baseado em assinaturas, instalada e configurada em um sistema operacional Linux, na distribuição Ubuntu, com posicionamento centralizado na infraestrutura de rede, diferente das abordagens convencionais com aplicação na borda da rede, explorando a característica centralizada do controlador *OpenFlow* na infraestrutura de rede SDN para tomada de decisão. A escolha do Snort se baseou no fato da ferramenta ser uma solução de detecção de intrusão baseado em assinaturas utilizando uma linguagem flexível de regras para analisar o tráfego coletado, mas se deve principalmente, ao fato de o controlador de rede *OpenFlow,* definido no trabalho, possuir integração nativa com a ferramenta.

3.7 A FERRAMENTA SNORT

O Snort um IDS baseado em rede, de código fonte aberto e um dos mais empregados atualmente pelo mercado, principalmente por disponibilizar aos administradores de redes informações suficientes para que possam programar uma ação frente a uma possível ameaça (SNORT, 2016). Ao ser empregado no monitoramento de redes TCP/IP de pequeno e médio portes, em diferentes plataformas, tem condições de detectar uma ação considerada suspeita, por meio de análise de uma variedade de tráfego.

Foi criado inicialmente por Martin Roesch, e, segundo Pimentel e Fagundes (2009), é conhecido por sua flexibilidade nas configurações de regras e por sua boa frequência de atualização da sua base de assinaturas, que possibilita a ferramenta ter condições de tratar novas técnicas de invasão de forma oportuna. Também referenciado como *packet-sniffer*, tem como diferencial a capacidade de inspecionar o *payload* do pacote e registro de pacotes, ou seja, o recurso de detecção de intrusão da ferramenta é realizado por meio da correspondência do conteúdo de cada pacote com as assinaturas de ataque.

O Snort utiliza uma estrutura simples, baseada na coletada de pacotes de rede, empregando a biblioteca *libpcap*[19], e um analisador simples e eficiente que trata tanto informações de cabeçalho quanto a área de dados dos pacotes coletados. Os pacotes, em análise, que coincidem com alguma das regras previamente configurada na ferramenta, podem ser simplesmente descartados, armazenados ou podem gerar um alerta aos responsáveis pelo sistema.

Importante destacar que um dos fatores críticos de sucesso para obtenção de um resultado eficaz na implantação de uma solução de detecção de ameaças é definir em que local no perímetro da rede o IDS será posicionado (MONTORO, 2012).

O Snort foi projetado para ser executado em uma grande gama de sistemas entre eles Linux, FreeBSD, NetBSD, OpenBSD, Solaris, MacOS, Windows, entre outros. Existem alguns programas opcionais que podem ser instalados para melhorar e facilitar a administração do sistema, como por exemplo, para o ambiente Linux pode-se citar:

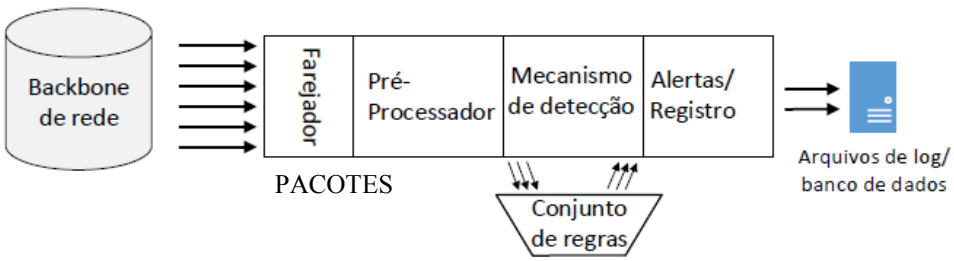
a) MySQL, para armazenamento dos alertas em banco de dados;
b) Guardian, para leitura em tempo real dos logs do Snort, e bloqueio via firewall;
c) Iptables, para bloqueio de ataques;
d) Smbclient, caso pretenda enviar mensagens WinPopup;
e) Apache, para disponibilização de informações via Web;
f) PHP, se houver *plug-ins* que o exija;
g) OpenSSH, para acesso remoto;
h) *Analysis Console for Intrusion Detection* (ACID), para apresentar de forma mais inteligível os logs do Snort armazenados em banco de dados.

A arquitetura do Snort (Figura 6) é composta de 4 componentes básicos: o farejador, o pré-processador, o mecanismo de detecção e os *plugins* de saída. O Snort captura pacotes e os processa por meio do pré-processador e depois verifica esses mesmos pacotes com relação a uma série de regras. Ele utiliza uma linguagem de regras flexível, para descrever o tráfego que deve ser analisado ou

[19] *Libcap* é uma biblioteca que provê uma interface portável para captura de pacotes de rede em baixo nível.

liberado, e também um mecanismo de detecção que utiliza uma arquitetura modular de *plug-ins*, que registram os ataques de diversas maneiras, dentre elas em banco de dados (MySQL, PostgreSQL, Oracle etc), em arquivo binário no formato do *tcpdump*, em arquivo texto ou em registros em um servidor de *syslog*.

Figura 6 - Arquitetura do Snort



Fonte: Baker (2007).

A base de assinaturas é baseada num conjunto de regras, que são verificadas pelo Snort no momento da análise do pacote, sendo utilizadas de acordo com a definição no arquivo de configuração. Em instalação padrão a ferramenta já vem pré-configurada com uma série de regras, sendo necessário apenas fazer pequenas alterações no serviço para configurar a atualização automática.

O Snort possui pré-processadores que realizam a detecção de instrusão que não podem ser feitas usando-se regras regulares ou detecção de anomalias de protocolos. Essa categoria de pré-processadores permite que o Snort possa ser facilmente estendido para detectar ataques, praticamente de qualquer maneira que um desenvolvedor possa imaginar. Alguns ataques simplesmente não podem ser detectados por correspondência de regra ou detecção de anomalia de protocolo, como por exemplo, a detecção de varreduras de portas.

A grande vantagem no uso do Snort reside no fato de ser uma ferramenta flexível e permitir que o próprio administrador possa escrever suas próprias regras, assim como alterar as já existentes, acrescentando mais funcionalidades. O Snort, como sensor, pode monitorar diferentes pontos da rede ao mesmo tempo, podendo se citar os seguintes:

a) Normalizar requisições *Hypertext Transfer Protocol* (HTTP);
b) Detectar ataques do tipo Unicode;
c) Detectar *Buffer Overflow*;
d) Detectar *portscan*;
e) Remontar os segmentos TCP; e

f) Ativar regras dinamicamente (regras podem ser ativadas por outras regras).

Por ser determinística, a detecção baseada em regras funciona bem devido a sua simplicidade e fácil otimização.

3.8 VULNERABILIDADES EM REDES SDN

Embora o paradigma SDN apresente muitos benefícios em comparação às redes tradicionais, algumas das vulnerabilidades das redes tradicionais persistem na SDN, principalmente pelo fato de ser implementado usando equipamentos de rede e servidores legados que necessitam seguir os mesmos padrões de segurança. Além disso, a SDN origina novos problemas exclusivos deste domínio que, por ainda ser uma tecnologia relativamente nova, ainda estão sendo descobertos pela comunidade acadêmica e pelos grandes players de tecnologia do mercado. Em particular, as características desejáveis da SDN como gestão e configuração centralizada se tornam alvos de ataques, possibilitando, por exemplo, que um usuário malicioso, caso tenha sucesso, venha comprometer o controlador ou até mesmo obter o controle sobre o mesmo, causando um impacto significativo no funcionamento da rede.

O usuário malicioso poderia, por exemplo, fazer uso de ferramentas para causar um aumento de consumo dos recursos do controlador, propositalmente, visando inundá-lo com solicitações forjadas e forçá-lo a responder lentamente a eventos do tipo *Packet_In*. Esta ação provocaria uma demora ou até perda no envio das mensagens *Packet_Out*, comprometendo o funcionamento da rede. Outro aspecto diz respeito a segurança do próprio equipamento servidor que hospeda o serviço do controlador, tendo em vista que muitos controladores SDN executam alguma forma de sistema operacional Linux, ou seja, um sistema operacional de propósito geral, que muitas vezes possui vulnerabilidades e é instalado e configurado sem que o administrador implemente as melhores práticas de segurança da informação ou *hardening* nos componentes que compõem a arquitetura de rede *OpenFlow*. O usuário malicioso se utilizando de técnicas já conhecidas poderia explorar vulnerabilidades já conhecidas deste sistema

operacional de propósito geral para ou comprometer o controlador ou obter acesso do mesmo.

Outro exemplo poderia ser o uso de tráfego forjado ou falsificado para atacar Switches e controladores, seja de forma não intencional por equipamentos defeituosos, seja de forma intencional por um usuário malicioso. Um usuário malicioso poderia usar elementos de rede, como por exemplo, Switches, servidores ou computadores pessoais, para lançar um ataque do tipo DoS. Dessa forma, o controle de acesso poderia ser empregado visando garantir a segurança no controle de admissão dos elementos de rede na rede SDN. Tanto a autenticação das estações de trabalho que necessitam acesso à rede, quanto o nível de privilegio atribuído a cada estação são essenciais para garantir a segurança das SDN.

Para tratar esse problema, o mecanismo de autenticação e controle de acesso conhecido como AuthFlow (MATTOS; DUARTE, 2014) poderia ser implementado em uma rede SDN. A autenticação do mecanismo AuthFlow é realizada direto na camada de enlace e, portanto, tem a vantagem prover uma baixa sobrecarga de controle quando comparada a uma autenticação na camada rede, ou na camada de aplicação, que dependem da atribuição de um IP a estação que está se autenticando e dependem da troca de informações da aplicação para a autenticação.

A autenticação da estação final na camada de enlace é realizada através do padrão IEEE 801.X que garante que as informações de autenticação sejam trocadas de forma padronizada entre a estação e o autenticador e, portanto, não requer nenhum tipo de alteração nas estações finais. O mecanismo de autenticação encapsula as mensagens no formato de protocolo de autenticação estendível (*Extensible Authentication Protocol* - EAP), o que permite a adoção de diferentes métodos de autenticação. Outra vantagem do mecanismo proposto é o provimento de um controle de acesso refinado, já que o AuthFlow permite definir políticas de acesso por fluxo para cada estação de acordo com sua credencial.

Assim, o controle de quais serviços uma estação pode acessar passa a ser realizado de acordo com as suas credenciais e não mais com seus endereços IP ou MAC. O mecanismo AuthFlow é composto por uma aplicação que executa sobre o POX, o controlador *OpenFlow*, e possui mais dois componentes principais: o autenticador e o servidor de serviço de autenticação e contabilização remota de

usuários (Remote Authentication Dial In User Service (RADIUS)). O autenticador recebe as mensagens no padrão IEEE 802.1x e valida as credenciais da estação com o servidor RADIUS.

O uso do mecanismo de autenticação simples pode mitigar o problema, mas se um intruso assume o controle de uma aplicação que armazena detalhes de muitos usuários, ele poderia facilmente usar as mesmas portas autenticadas e endereços MAC de origem para injetar fluxo autorizado, mas forjado, dentro da rede. Para tratar tal aspecto, sugere-se empregar um IDPS com suporte à análise da causa raiz, em tempo de execução, que poderia ajudar a identificação de fluxos considerados maliciosos.

Outro tipo de ameaça seria o ataque voltado contra o plano de controle, que poderia ser usado para gerar ataques de DoS ou roubo de dados. Como é bem conhecido na comunidade de segurança, usar TLS/SSL por si só não estabelece a garantia de comunicação segura, o que poderia ocasionar um possível comprometimento do enlace de comunicação entre os dispositivos da rede OpenFlow/SDN com o controlador. Vários trabalhos relatam as fraquezas de comunicações TLS/SSL e sua principal âncora de confiança, a infraestrutura de chaves públicas (HOLZ, 2012).

A segurança dessas comunicações seria tão forte quanto o seu elo mais fraco, seja pelo uso de certificado auto-assinado, uma Autoridade de Certificação (AC) comprometida ou aplicações vulneráveis e bibliotecas. Por exemplo, há muitas implementações de SSL vulneráveis ao ataque de homem do meio ou *main-in-the-middle*, sendo usadas em sistemas críticos em todo o mundo (GEORGIEV, 2012). Além disso, o modelo TLS/SSL não é suficiente para estabelecer e assegurar a confiança entre controladores e comutadores.

Uma das possibilidades seria proteger a comunicação com criptografia em todo controlador de réplicas (DESMEDT, 1994) (onde o comutador irá precisar de pelo menos n partes para obter uma mensagem válida do controlador). Além disso, o uso de mecanismos de associação de dispositivos dinâmico, automatizados e seguros podem ser considerados, de forma a garantir confiança entre os planos de controle e os planos de dados dos dispositivos.

Outra ameaça diz respeito a ataques e vulnerabilidades direcionadas a estações administrativas que são usadas nas SDN para acessar e administrar o

controlador da rede. Essas estações de trabalho já seriam um alvo fácil de serem exploradas nas redes tradicionais, com a diferença de que a superfície de ameaça nas SDN, caso estas máquinas venham a ser comprometidas aumenta substancialmente, possibilitando, por exemplo, que a rede seja programada por um usuário malicioso de uma simples localização. Como possível solução, seria possível o emprego de protocolos com dupla verificação de credenciais (por exemplo, exigindo as credenciais de dois diferentes usuários para acessar um servidor de controle). Além disso, a utilização de mecanismos de recuperação assegura o estado de confiança após a reinicialização.

Como último aspecto de segurança a ser considerado, ressalta-se a falta de recursos confiáveis para possibilitar uma análise forense e um ambiente de recuperação, que poderia permitir o entendimento da causa raiz de um determinado problema. A fim de investigar e estabelecer os fatos sobre os quais ocorreu um determinado incidente é preciso ter informações confiáveis de todos os componentes e domínios da rede. Além disso, esses dados só serão úteis se a sua confiabilidade (integridade, autenticidade, etc..) puder ser assegurada. Da mesma forma, remediação requer um registro instantâneo (snapshots) do sistema de forma segura e confiável para garantir uma recuperação rápida e correta dos elementos da rede a um estado conhecido de funcionamento correto.

Como solução este trabalho sugere implementar mecanismos de registro e rastreabilidade tanto no plano de dado quanto no plano de controle. No entanto, a fim de ser eficaz, esses mecanismos devem ser indeléveis (um logo que é garantido por ser imutável e seguro). Além disso, tão importante quanto os registros, devem ser armazenados em ambiente seguro e de preferência remoto.

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 crítica 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 CERT.br, só em 2015 foram quase 722.205 incidentes reportados. Dessa forma, implementar IDS e IPS tornam-se essenciais para

garantir a aplicação e acompanhamento das políticas de segurança da rede com o objetivo de buscar bloquear ou mesmo minimizar os impactos causados por uma atividade maliciosa na rede.

4 AMBIENTE PARA DETECÇÃO E PREVENÇÃO FLEXÍVEL DE ATAQUES EM REDES *OPENFLOW/SDN*

Este capítulo tem por objetivo detalhar a proposta de pesquisa, apresentando as estratégias adotadas para permitir realizar o monitoramento e o tratamento flexível de eventos de segurança em uma rede real *OpenFlow/SDN* em ambiente de *Testbed*, apresentar os blocos de códigos empregados no desenvolvimento do plano de controle e os aspectos gerais de implementação no nível de programação no plano de controle.

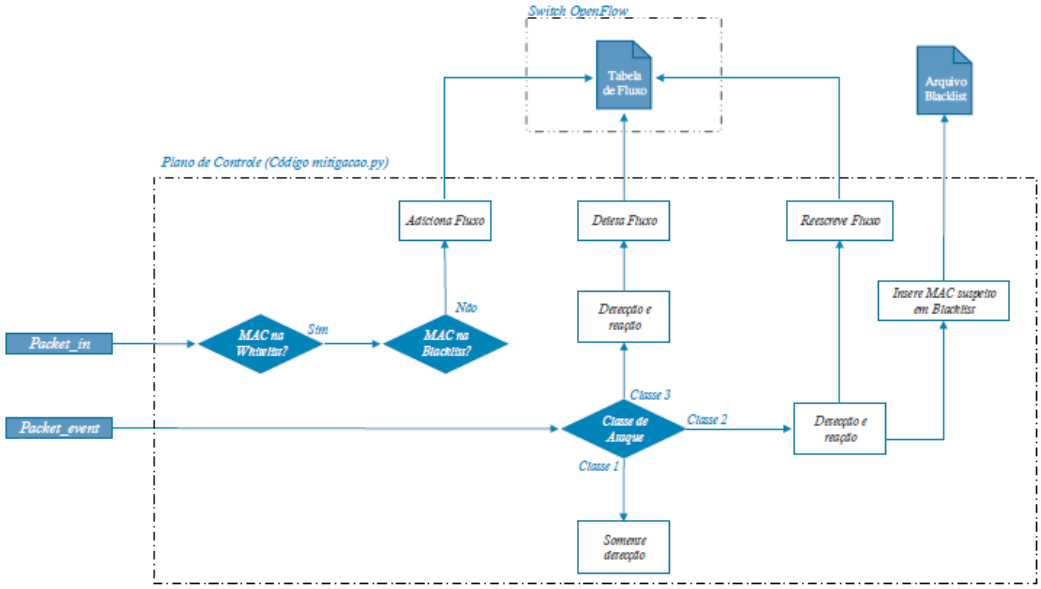
4.1 DESCRIÇÃO DA PROPOSTA

A proposta desta dissertação consiste na concepção e construção de uma rede real *OpenFlow/SDN* em ambiente de *Testbed* que associada à implementação de mecanismos de monitoramento e tratamento de eventos de segurança, possibilita o tratamento adequado para diferentes tipos de ataque em um ambiente *OpenFlow/SDN*.

A construção de uma rede real foi necessária para disponibilizar os recursos computacionais necessários para validação dos mecanismos de detecção e reação às ameaças de segurança propostos nesta dissertação. Como solução para detecção de intrusão foi definida a ferramenta Snort, que já era de domínio do autor da pesquisa, e possui integração nativa com o arcabouço do controlador de rede OpenFlow/SDN.

Para a implementação dos mecanismos de monitoramento e tratamento de eventos de segurança foi necessário estender as funcionalidades do arcabouço do controlador de rede *OpenFlow* adotada, com o desenvolvimento no plano de controle, sendo necessário modificar o código principal do controlador *Ryu* e a construção de um novo código chamado *mitigacao.py*, seguindo a lógica apresentada no processo de tratamento apresentada na Figura 7 para implementação das ações (envio de mensagens *OpenFlow*), que são enviadas ao *Switch OpenFlow* com o objetivo de remover ou reescrever as entradas de fluxos no *Switch* daqueles endereços MAC que originaram uma determinada ação considerada maliciosa na rede (MAC suspeito).

Figura 7 - Processo de tratamento de incidentes de segurança



No entanto, a simples ação de remover os endereços MAC suspeitos da entrada de fluxo no *Switch OpenFlow* não seria suficiente para a solução ser efetiva, na medida em que este mesmo equipamento poderia solicitar ao *Switch* a criação de novas entradas de fluxo e manter a sua conexão à rede. Para tratar este problema, foi empregado o recurso de *blacklist*, onde o MAC suspeito é gravado no arquivo *blacklist.txt* logo após ter suas entradas de fluxo removidas. Como todo tráfego de entrada é tratado no método *packet_in_handler()*, antes de realizar a chamada ao comando *addflow()* para adicionar uma entrada de fluxo no *Switch OpenFlow*, o MAC de origem da mensagem *Packet_in* é comparado a *blacklist*. Se o endereço MAC da mensagem *Packet_in* estiver na *blacklist,* a entrada de fluxo não é criada e o dispositivo não terá acesso à rede até que o administrador tome as ações necessárias para garantir que o dispositivo suspeito esteja em conformidade com a política de segurança e remova o endereço MAC suspeito da lista.

Outra solução adotada nesta dissertação foi o recurso de *whitelist*, possibilitando que somente os dispositivos com seus endereços MAC previamente cadastrados no arquivo *whitelist.txt* tenham acesso a rede e consigam criar entradas de fluxo no *Switch OpenFlow*. Nesse caso, quando o dispositivo envia uma mensagem *Packet_in* para o *Switch*, o endereço MAC de origem da mensagem é comparado à *whitelist*, sendo tomadas duas ações, quais sejam: se o endereço MAC estiver na *whitelist* o controlador envia o comando *addflow()* para

o *Switch* para criar a entrada de fluxo; senão estiver, no entanto, é apresentado na saída de comando na tela do controlador a informação de que o MAC não está na *whitelist* e o administrador deve ser procurado.

Com relação à solução adotada para implementação dos mecanismos de reação, a estratégia aplicada não seguiu uma abordagem convencional, onde toda e qualquer mensagem *Packet_event*, por exemplo, tivesse uma única ação a ser realizada, como estava sendo feito no início da pesquisa. A abordagem adotada no trabalho seguiu uma estratégia em que se fez uso da flexibilidade no plano de controlador para diferenciar o tratamento das mensagens *Packet_event*. Por meio das informações enviadas ao controlador, que na verdade são as assinaturas de ataque previamente cadastradas no arquivo *snort.rules* da ferramenta Snort, é possível, por meio da lógica implementada no arquivo de código *mitigacao.py*, que o controlador consiga dar um tratamento específico para cada tipo de Classe de ataque. A definição e decisão de quais tipos de tráfego são classificados em cada tipo de Classe fica a cargo do administrador da rede, que tem total flexibilidade para estabelecer a melhor política para realidade de sua rede. Além desse aspecto no tratamento dos eventos de segurança por tipo de Classe de ataque, outra abordagem inovadora empregada neste trabalho consiste no uso da ação de reescrever (*REWRITE*) uma entrada de fluxo no *Switch OpenFlow*, característica que está disponível apenas nas redes *OpenFlow/SDN* por meio da definição do próprio protocolo OpenFlow e não nas redes convencionais.

Atualmente, qualquer tipo de ação disponível nos equipamentos convencionais está limitada a remoção ou inclusão de uma entrada na tabela MAC dos dispositivos de Layer 2 ou a remoção ou inclusão de uma entrada de rota nos roteadores, dispositivos de Layer 3. Visando apresentar e demonstrar a estratégia adotada ao longo do trabalho, nos próximos subcapítulos serão detalhados os principais aspectos que envolveram a concepção e implementação do Ambiente para Detecção e Prevenção Flexível de Ataques em redes *OpenFlow/SDN*.

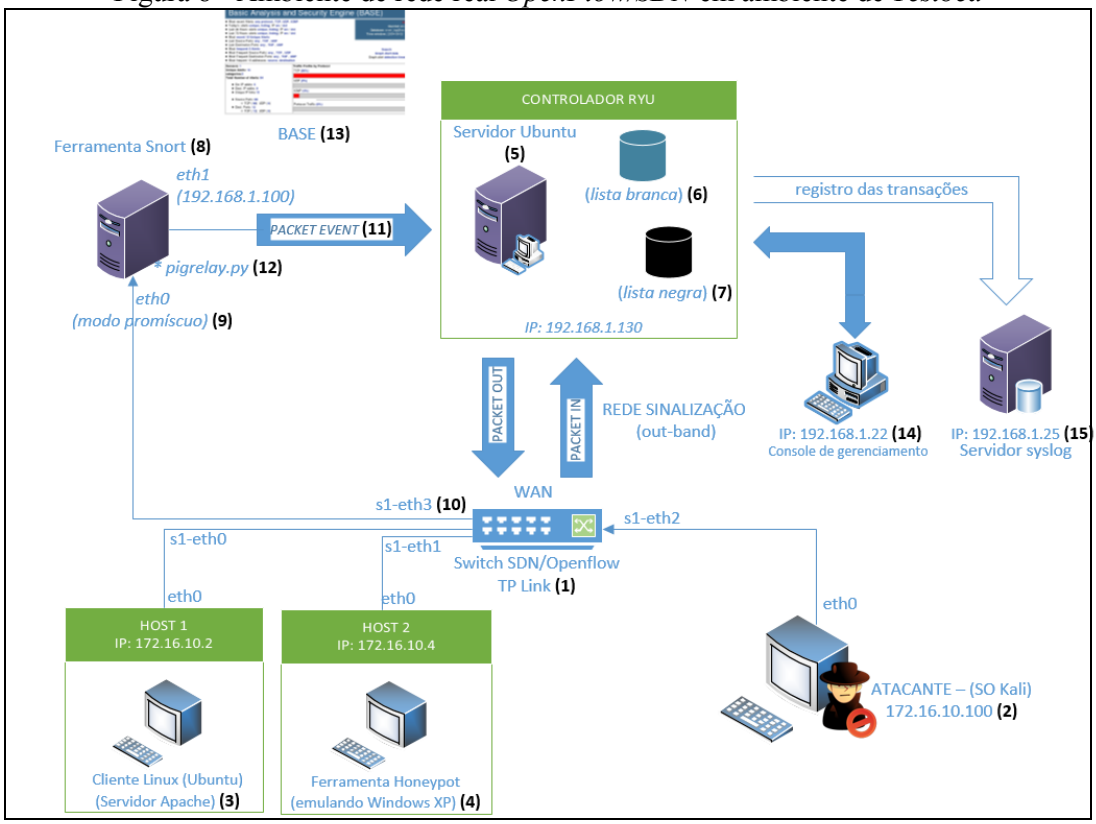
4.2 AMBIENTE DE REDE REAL *OPENFLOW/SDN* PARA EXPERIMENTAÇÃO

O ambiente de rede real *OpenFlow/SDN* em ambiente de *Testbed* (Figura 8) consiste de equipamentos e componentes necessários para validação da proposta

de pesquisa e obtenção de dados em rede *OpenFlow/SDN*. O ambiente de rede proposto foi implementado com o uso de servidores, estações de trabalho e um *Switch OpenFlow* **(1)**, visando estabelecer um ambiente computacional que apresente os melhores resultados por meio de tráfego de rede real e não apenas simulado.

Para o processo de teste e simulação foram usadas duas máquinas reais como clientes de rede, sendo uma instalada com o sistema operacional Kali **(2)** conectado na porta 2 do *Switch OpenFlow* para simular os ataques. A outra máquina, aquela sob ataque, foi instalada com a distribuição Ubuntu Server 16.04 **(3)**. Empregou-se também uma máquina *honeypot* **(4),** emulando máquinas na rede por meio da ferramenta *honeyd*, uma *honeyport* de baixa interação. O equipamento servidor **(5)** executa o controlador Ryu na rede *OpenFlow*. A configuração experimental inclui mecanismos de proteção associados ao uso dos recursos *whitelist* **(6)** e *blacklist* **(7)**, de acordo com os objetivos definidos no trabalho visando demonstrar a efetividade da solução.

Figura 8 - Ambiente de rede real *OpenFlow/SDN* em ambiente de *Testbed*



O controlador Ryu suporta um *Switch* L2 com integração nativa com a ferramenta Snort **(8)** usando a flexibilidade de programação do controlador. A máquina com o Snort tem a interface *eth0* configurada em modo promíscuo **(9)** que recebe todo o tráfego na porta 3 do *Switch OpenFlow* em modo de espelhamento **(10)** e a interface *eth1* configurada no IP 192.168.1.100/24 para enviar pacotes do tipo *Packet_event* para o controlador Ryu quando algum tipo de tráfego malicioso é identificado na rede.

Os mecanismos de detecção e reação são implementados por implementação de código em *python* no arquivo *mitigacao.py* rodando no controlador, que trata os eventos *Packet_event* gerados pelo Snort. Estes eventos, no entanto, são enviados para o controlador Ryu por meio de uma aplicação desenvolvida em *python* chamada *pigrelay.py* **(12)** (PIGRELAY, 2014). A ferramenta *pigrelay* está disponível na máquina Snort e quando em execução envia para o controlador no IP 192.168.1.130 e porta 6654 – eventos de segurança sendo gerados pela ferramenta Snort, armazenados em /var/log/snort/snort_alert. Para habilitar o armazenamento dos dados gerados pela ferramenta Snort foram usados um servidor de banco de dados MySql, para o armazenamento do tráfego suspeito, e uma ferramenta Web chamada *Basic Analysis and Security Engine* (BASE) **(13)** para disponibilizar recursos de análise de intrusão no ambiente de rede SDN proposto.

Para garantir maior segurança no acesso ao controlador Ryu e possibilitar que somente pessoas devidamente autorizadas possam fazer qualquer tipo de alteração em sua configuração o acesso ao equipamento servidor só é permitido por meio da console de gerenciamento **(14)** no IP 192.168.1.22. Com objetivo garantir o não-repúdio e o registro de todas as ações praticadas pelo administrador de rede na configuração do controlador Ryu, o ambiente de rede provê também um servidor de *syslog* **(15)** para viabilizar um recurso de auditoria na rede *OpenFlow/SDN* caso seja necessário.

4.2.1 **Equipamentos empregados no ambiente de *Testbed***

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, assim como pelo (CONSORTIUM, 2012) para funcionar como um *Switch OpenFlow* de baixo custo. Na Tabela 2 estão descritas as especificações do roteador sem fio TP-Link empregado como *Switch OpenFlow*, assim como na Tabela 3 é possível ter acesso as especificações de todos os equipamentos empregados:

Tabela 2 - 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

Tabela 3 - Equipamentos e configurações empregados no ambiente de *Testbed*

Equipamentos	Configuração
Nó 1	Processador com 7 núcleos, 8 GB de memória RAM, 500 GB de HD e 1 interface de rede (eth0)
	Versão 8 Single Language baseado no sistema operacional Windows 8 com plataforma Virtualbox para virtualização
Nó 2	Processador com 7 núcleos, 8 GB de memória RAM, 500 GB de HD e 1 interface de rede (eth0)
	Versão 8 Single Language baseado no sistema operacional Windows 8 com plataforma Virtualbox para virtualização
Atacante	Processador com 4 núcleo, 512 MB de memória RAM, 8 GB de HD e uma interface de rede (eth0)
	Distribuição Kali Linux 2.09 com ferramentas para testes de penetração e forense
Snort	Processador com 7 núcleos, 8 GB de memória RAM, 500 GB de HD e 2 interfaces de redes (eth0) e (eth1)
	Versão 16.04 LTS baseado na distruição Ubuntu
Console de Gerenciamento	Processador com 7 núcleos, 8 GB de memória RAM, 500 GB de HD e 2 interfaces de redes (eth0) e (eth1)
	Versão 16.04 LTS baseado na distruição Ubuntu
Servidor Syslog	Processador com 7 núcleos, 8 GB de memória RAM, 500 GB de HD e 2 interfaces de redes (eth0) e (eth1)
	Versão 16.04 LTS baseado na distruição Ubuntu
Controlador RYU	Processador com 7 núcleos, 8 GB de memória RAM, 500 GB de HD e 2 interfaces de redes (eth0) e (eth1)
	Versão 16.04 LTS baseado na distruição Ubuntu

Para o suporte ao protocolo *OpenFlow* 1.3 o equipamento teve seu *firmware* alterado, conforme procedimentos descritos no Apêndice 4 deste trabalho,

recebendo uma versão de sistema operacional OpenWrt[20] compilada a partir da versão "*trunk*" do OpenWRT (*Barrier Breaker*) (CPQD, 2012). 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 *Testbed*.

4.2.2 O Sistema Operacional OpenWrt

O OpenWrt (OPENWRT, 2016) é um *firmware* utilizado em dispositivos embarcados para comutar tráfego de rede, onde todos seus componentes são otimizados para requerer pouca memória, já que a maioria dos roteadores contam com poucos megabytes de armazenamento. É derivado da distribuição Linux do Debian para roteadores pessoais, com um sistema de arquivos totalmente modificado e que permite personalizar o dispositivo, incorporando aplicações por meio do uso de pacotes de instalação. Seu desenvolvimento foi induzido inicialmente graças a sua licença *General Public License* (GPL), que acabou impulsionando todos os fabricantes que modificavam e melhoravam o código a liberar novas versões e contribuir cada vez más com o projeto em geral

A diferença do OpenWrt para maioria dos *firmwares* baseados em Linux é que para ser empregado não é necessário compilar novamente para modificar o *software* instalado. O *firmware* OpenWrt permite adaptar o roteador para realizar funções próprias de uma ferramenta muito mais especializada e de maior custo, ao passo que o *firmware* padrão provê apenas uma interface de fácil configuração, sem explorar o verdadeiro potencial do equipamento. Algumas características gerais do projeto OpenWrt são:

a) Possui a mesma estrutura do Unix: *kernel*, *shell* e sistemas de arquivos;

b) O *kernel*, *shell* e o sistema de arquivos do OpenWrt realizam as mesmas funções que um sistema Unix;

---

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

c) O *kernel* do OpenWrt de forma igual ao *kernel* do Unix controla a memória do dispositivo, executa as instruções do *Shell* e se encarrega de cumprir as permissões do sistema de arquivos;

d) O *Shell* do OpenWrt é o ash que é muito parecido com o *bash shell* de um sistema Unix, onde ambos permitem que sejam executados comandos e programas; e

e) O sistema de arquivos tem a função de organizar os dados de arquivos, diretórios e dispositivos conectados.

A configuração do OpenWrt pode ser realizada, seja por meio da interface de linha de comandos (*ash*), seja por meio de uma interface *web* (LUCI) baseado em Ajax, que pode ser instalado por meio da linha de comandos e se utiliza fontes do núcleo oficial do Gnu's Not Unix (GNU/Linux) por meio do qual é possível acrescentar drivers para garantir a compatibilidade das interfaces de rede de diferentes dispositivos. O OpenWrt pode ser utilizado não só em roteadores, mas também em telefones inteligentes, por exemplo o *Neo FreeRunner*, computadores de bolso com Ben NanoNote e pequenos portáteis como One Laptop per Child.

A última versão estável do OpenWrt é denominada Backfire 10.3, e sua última revisão foi lançada em dezembro de 2011 e corresponde ao OpenWrt Backfire 10.03.1. O roteador que emprega o OpenWrt como *firmware* pode estender suas funcionalidades por meio da instalação e/ou criação de *scripts*, programas e pacotes, tais como servidor VPN, controle da emissão de sinal Wi-Fi, servidor de impressão, servidor web, acesso a base de dados MySQL entre outras.

A configuração do OpenWrt está dividida em vários arquivos localizados no diretório /etc/config/, onde cada arquivo dentro deste diretório corresponde a uma parte do sistema. A edição dos arquivos de configuração (Tabela 4) podem ser feitas por meio de um editor de texto, via web o empregando várias API de programação como *Bash Shell* ou usando linguagens de programação tais como Lua ou C. Os sistemas de arquivo (*filesystem*) estruturam a informação guardada em uma unidade de armazenamento, que logo será representada de forma textual ou graficamente empregando um gestor de arquivos. OpenWrt incorpora dois sistemas de arquivos: SQUASHFS e JFFS2, onde ambos manuseiam as partições da memória flash e memória RAM do dispositivo.

Tabela 4 – Arquivos de configuração do OpenWrt

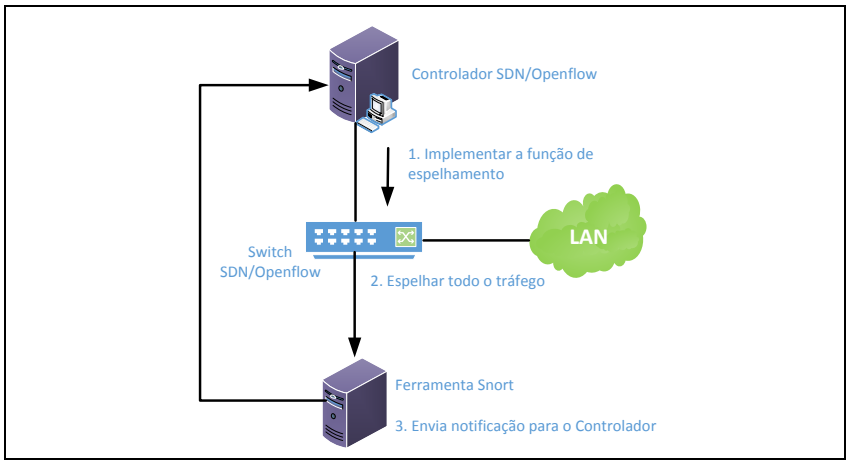
Arquivo	Descrição
/etc/config/dhcp	Configuração dos parâmetros do serviço de DHCP, DNS, IP, *Address Resolution Protocol* - ARP, etc.
/etc/config/firewall	Configuração do serviço de firewall
/etc/config/network	Configuração de rede, interfaces de rede e tabela de roteamento
/etc/config/ucitrack	Configuração dos serviços relacionados com arquivos
/etc/config/wireless	Configuração de rede sem fio e interface Wi-Fi
/etc/config/dropbear	Configuração do serviço SSH
/etc/config/luci	Configura os elementos essenciais da interface de acesso WEB
/etc/config/system	Configuração básica do sistema do roteador: nome do host, fuso horário, etc
/etc/config/uhttpd	Configura o servidor WEB

4.2.3 Integração da ferramenta Snort com o controlador Ryu

O envio da mensagem *Packet_event* para o controlador só é possível porque o arcabouço Ryu tem uma integração nativa com o Snort, viabilizada com o uso da ferramenta *pigrelay.py* (PIGRELAY, 2014), em execução na própria máquina do Snort. Essa integração contemplou a instalação e configuração do controlador Ryu e da ferramenta do Snort em máquinas diferentes, por questões de disponibilidade de cada serviço e escalabilidade da solução. O envio dos alertas do Snort para o controlador é realizado via soquete de rede (*Network Socket*) que ao entrar em estado de *listening,* quando da execução da ferramenta *pigrelay.py*, recebe os alertas do Snort por meio de um Unix Domain Socket.

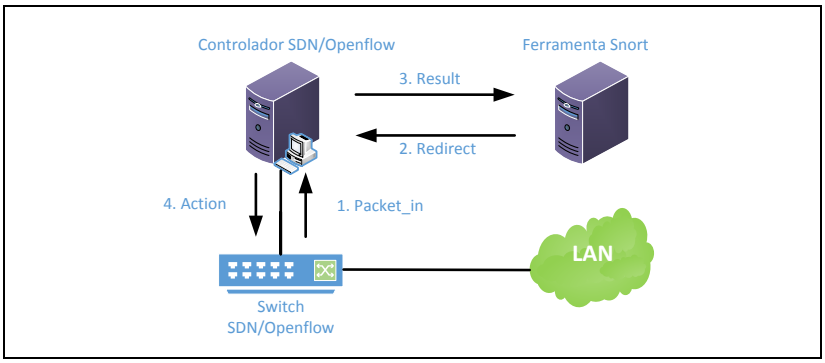
Existem duas formas de implementar o serviço Snort em um ambiente SDN. A primeira forma (Modo Espelhamento) consiste em implementar a função de espelhamento de porta em um *Switch OpenFlow*, como apresentado na Figura 9. Para implementar esta função de espelhamento, o controlador *OpenFlow* estabelece uma entrada de fluxo com duas portas de saídas, onde uma será empregada para a porta de encaminhamento regular e a outra será a porta para o Snort.

Figura 9 - Implementação do Snort em SDN (Modo espelhamento)



No Modo Espelhamento, todo o tráfego será encaminhado não somente para o destino, mas também para a análise do Snort. E uma vez que seja detectado um tráfego considerado malicioso, o Snort irá notificar o controlador *OpenFlow*, que ao tratar a mensagem tem condições de enviar um comando para o *Switch OpenFlow* com a ação para derrubar (*DROP*) o tráfego específico. A segunda forma (Modo *PACKET_IN*) é apresentada na Figura 10. Esta abordagem emprega uma porta Snort por meio de uma *daemon* para uma aplicação SDN.

Figura 10 - Implementação do Snort em SDN (Modo *Packet_In*)



Todo o tráfego será encaminhado para o controlador OpenFlow por meio de eventos PACKET_IN do protocolo OpenFlow. O controlador OpenFlow então trata os tráfegos recebidos pela aplicação Snort. O problema desta arquitetura é a grande carga no canal do controlador OpenFlow. Isso acontece porque todo o tráfego é transmitido em ambos os planos de dados e plano de controle. Assim, usar pacotes Packet_In como um método de encaminhamento tem uma tendência a inundar o sistema. A arquitetura de segurança de rede proposta neste trabalho adota a primeira solução de implementação do Snort para redes SDN apresentada

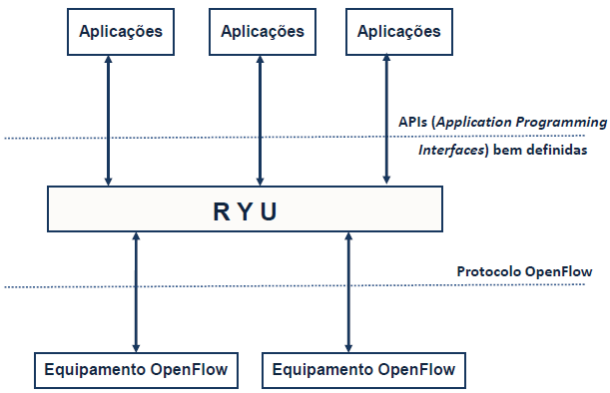
na Figura 9, que consiste em implementar a função de espelhamento de porta em um Switch OpenFlow.

4.3 MECANISMOS DE MONITORAMENTO E TRATAMENTO DE EVENTOS DE SEGURANÇA

Como plataforma para codificação de aplicações no plano de controle este trabalho emprega o Ryu (dragão, fluxo em japonês), que é um dos controladores disponíveis no mercado, desenvolvido pelo Centro de Inovação em *software* da empresa japonesa NTT e seu código está disponível gratuitamente sob a licença Apache 2.0 (RYU, 2014).

O projeto Ryu é um arcabouço baseado em componentes para programação de redes definidas por *software*, completamente escrito na 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, facilitando a criação de novas aplicações de controle, conforme apresentado na Figura 11.

Figura 11 - Controlador Ryu



Fonte: Ryu (2014).

Existe um grande número de controladores SDN que empregam o protocolo *OpenFlow* para a configuração do plano de dados. Logo, antes da definição de qual controlador seria mais adequado e alinhado aos resultados esperados e definidos nos objetivos deste trabalho, foi necessária uma fase prévia de investigação e levantamento. Serviram de apoio à decisão o curso *online* sobre

SDN disponibilizado no Coursera (COURSERA, 2016), artigos na internet e os documentos disponíveis na página oficial do projeto (RYU, 2014).

Os aspectos fundamentais para definição do controlador Ryu como solução adotada neste trabalho foram:

a) Suporte ao protocolo *OpenFlow* 1.3: foi um dos fatores determinantes, 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. Tendo em vista que um dos focos do trabalho é apresentar aspectos de detecção de ameaças, se fosse empregado o protocolo *OpenFlow* 1.0 a pesquisa ficaria limitada devido à disponibilidade de poucas informações do cabeçalho *OpenFlow*.

b) Linguagem de programação Python: a facilidade de programação e assimilação da linguagem foi considerada como aspecto facilitador no desenvolvimento do projeto.

c) Curva de aprendizagem: o fato do esforço de assimilação dos detalhes do controlador demandar um curto espaço de tempo facilita o desenvolvimento do trabalho.

d) Segurança: o fato de o 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 foi outro fator determinante para a decisão.

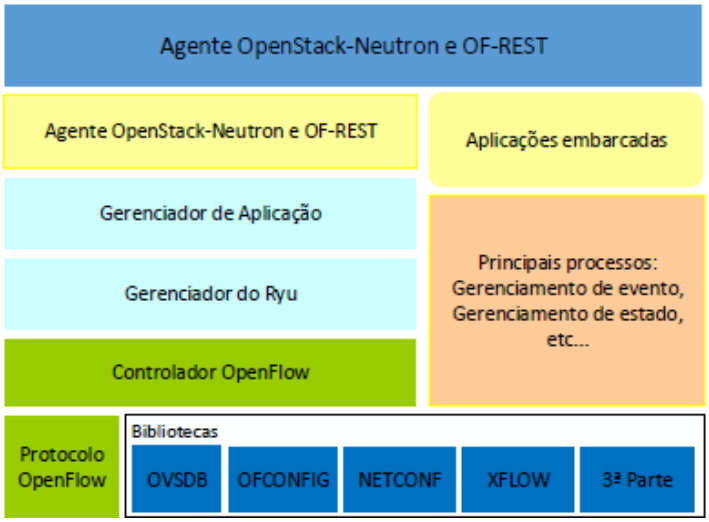
e) Integração com outras aplicações: além da integração com a ferramenta Snort o arcabouço Ryu permite integração com outras aplicações como, por exemplo, *OpenStack* (mediante o *plugin* Neutron) e Zookeeper para conseguir alta disponibilidade.

f) Boa documentação, suporte e comunidade de usuários: a despeito das soluções SDN serem recentes, o projeto Ryu conta com ampla documentação de projeto e arcabouço, sendo mais um dos fatores determinantes para sua aplicação. Destacando-se ainda se destaca pela disponibilidade de fóruns no site do projeto.

g) Interface REST: o arcabouço Ryu conta com uma interface REST para a interação com a rede usando *OpenFlow,* que permite criar novas API REST utilizando *Web Server Gateway Interface* (WSGI).

A plataforma que dá suporte ao controlador Ryu consiste em um executável chamado "/bin/ryu-manager" (Figura 12), responsável pelo carregamento e inicialização do próprio controlador Ryu e das diferentes aplicações 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), que permite que os *Switches OpenFlow* estabeleçam conexão com o controlador.

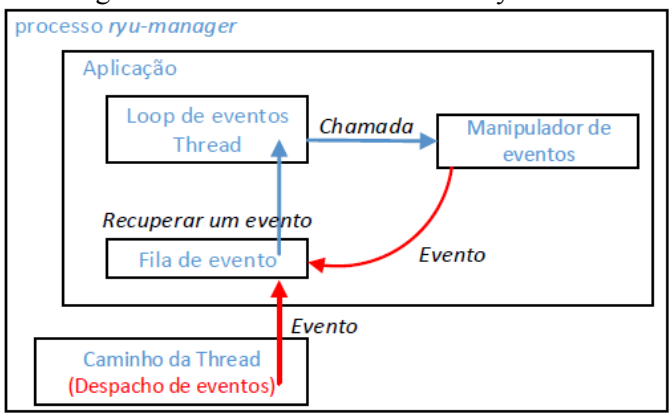
Figura 12 - Arquitetura do Ryu



Fonte: Ryu (2014).

No nível mais baixo se encontram uma série de bibliotecas de protocolos *Southbound* para comunicação com os dispositivos de rede. No nível de cima estão os processos do Ryu e o controlador *OpenFlow*, que atua como um intermediário entre as aplicações e os dispositivos, fazendo o tratamento das mensagens *OpenFlow*. *App manager* é o componente que faz a gestão das aplicações em execução no Ryu, sendo que toda aplicação é uma classe derivada da classe RyuApp do espaço de nomes de *App manager*. A Figura 13 mostra o modelo de eventos do Ryu.

Figura 13 - Modelo de eventos do Ryu



Fonte: Ryu (2014).

As aplicações são entidades monohilo do Ryu, que se comunicam umas com as outras por meio de eventos assíncronos gerenciados pelo núcleo do Ryu. Além dos eventos gerados pelas aplicações, também existem eventos gerados a partir de entidades próprias do Ryu, como o controlador *OpenFlow*. Um exemplo pode ser um evento gerado pelo controlador *OpenFlow* em resposta à recepção de uma mensagem *Packet_in*.

A lógica implementada pelo programador (como a que foi desenvolvida neste trabalho) resulta em uma aplicação que segue 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 a versão 2.7 empregada no contexto deste trabalho. Por sua vez, essas aplicações se comunicam umas com as outras por meio de geração de eventos, através de objetos da classe "ryu.controller.event.EventBase". Cada aplicação tem sua própria fila First-In First-Out (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 dada atenção especial para que as funções que tratam os eventos não sejam não bloqueantes[21].

O arcabouço Ryu é distribuído com uma série de aplicações, incluindo scripts que emulam o comportamento de um *Switch* L2 utilizando várias versões do *OpenFlow* e distintas variações de *Switches* que implementam *Link Agregation Control Protocol* - LACP (agregação de enlaces), Internet Group Management

[21] 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.

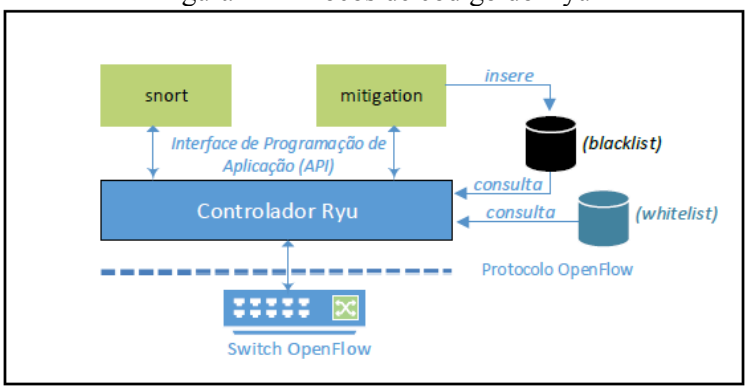
Protocol - IGMP (encaminhamento *multicast*) e *Spanning Tree Protocol* - STP (árvore de expansão), além de contar com integração nativa com o IDS Snort. Também inclui várias REST API que permitem emular o comportamento de um roteador, políticas de *Quality of Service* (QoS), regras para um firewall ou para gerenciamento da topologia de rede.

No entanto, as funcionalidades do código do Ryu, disponibilizado no site projeto, não atendem as exigências desta dissertação, limitando-se apenas a disponibilizar mecanismos de detecção de eventos de segurança, não sendo suficiente para, no plano de controle, conseguir prover a rede *OpenFlow/SDN* com os mecanismos de proteção, conforme definidos no trabalho e que foram formalmente expressos no objetivo geral e nos objetivos específicos. Nesse sentido, buscando atender a proposta desta dissertação, foi necessário um esforço no entendimento do arcabouço Ryu e desenvolvimento de novas funcionalidades no plano de controle, por meio de um modelo de solução que será apresentado abaixo.

4.3.1 O Arcabouço Ryu como solução no plano de controle

Como apresentado na Figura 14, duas aplicações foram implementadas no topo do controlador Ryu. A primeira aplicação foi o *switch_snort.py*, que é o código principal, implementado para dar suporte a um *Switch* L2, sendo responsável por habilitar o *Switch OpenFlow* a redirecionar todo o tráfego da rede em modo promíscuo em uma de suas portas para a máquina que executa a ferramenta Snort, que irá avaliar e detectar qualquer tipo de comportamento considerado malicioso na rede. A segunda aplicação chamada "*mitigation*" (*mitigation.py*) tem por objetivo tratar as mensagens *Packet_event* enviadas da ferramenta Snort para o controlador Ryu quando é reportado algum tipo de evento associado com uma assinatura de ataque na rede *OpenFlow*. As duas aplicações foram concebidas para prover no plano de controle os recursos necessários para implementar os mecanismos de detecção e resposta.

Figura 14 - Blocos de código do Ryu



O método *packet_in_handler* é o método principal no arquivo *switch_snort.py*, onde são processadas todas as mensagens *OpenFlow*. Nele espera-se a criação de entradas de fluxo no *Switch* apenas se o MAC de origem associado as mensagens *Packet_in* estiver registrado previamente em uma lista branca (*whitelist*) e não estiver na lista negra (*blacklist*). Nesse método também foi implementado o tratamento para as mensagens *Packet_event* enviadas pelo Snort.

O conceito proposto é baseado no mecanismo de classificação de fluxo SDN por tipo de tráfego como apresentado na Tabela 5. Um importante aspecto dessa abordagem é o ambiente de teste para geração de tráfego na rede SDN que permite verificar a efetividade dos mecanismos de tratamento por tipo de ataque.

Tabela 5 - Classes de tráfego disponíveis na rede real de *Testbed*

Classe de tráfego	**Atividades**	**Exemplo de atividades**	**Ferramentas para geração de tráfego**
Classe 1	Normal	Comunicação entre cliente e servidor	FTP, SSH, SMB, Apache, WEB, Postgres, Telnet, Ping ICMP
Classe 2	Probe	Port probe, Scan de vulnerabilidade e versão	Metasploit, nmap
Classe 3	DoS	Negação de serviço	Metasploit, hping3, nping

Fonte: Lopes (2016).

A Tabela 6 apresenta as possíveis ações programadas no controlador Ryu a serem executadas depois de identificado um tráfego malicioso na rede enviadas pelo Snort ao controlador para tratamento. Dois métodos implementados no arquivo *mitigacao.py* executam o tratamento dos eventos de segurança conforme apresentado na Tabela 6, permitindo para cada tipo de classe de ataque uma ação específica ou um conjunto de ações.

Tabela 6 - Tratamento flexível dos eventos de segurança

Classe de ataque	Atividades	Ação	Tipo de tratamento
Classe 1	Normal	Somente detecção	1. Somente armazena em BD o tráfego suspeito
Classe 2	Probe	Detecção e reação	1. Armazena o tráfego suspeito em BD 2. Reescrever (*REWRITE*) as entradas de fluxo
Classe 3	DoS	Detecção e reação	1. Armazena o tráfego suspeito em BD 2. Retira (*DROP*) todas as entradas de fluxo 3. Insere o MAC suspeito em uma *Blacklist*.

A própria máquina do Snort trata os ataques de Classe 1. Nesse caso, por ser um tipo de tráfego de baixo impacto para o funcionamento da rede, ao serem detectados por meio das regras definidas no Snort a única ação consiste em armazenar todo o tráfego em um banco de dados MySQL com o objetivo de viabilizar uma análise futura usando uma ferramenta, também implementada, de análise de eventos denominada BASE. Esse recurso de armazenamento do tráfego considerado suspeito na rede está disponível para os tipos de ataque Classe 1, e também é feito, por padrão, para os tipos de ataque das Classes 2 e 3.

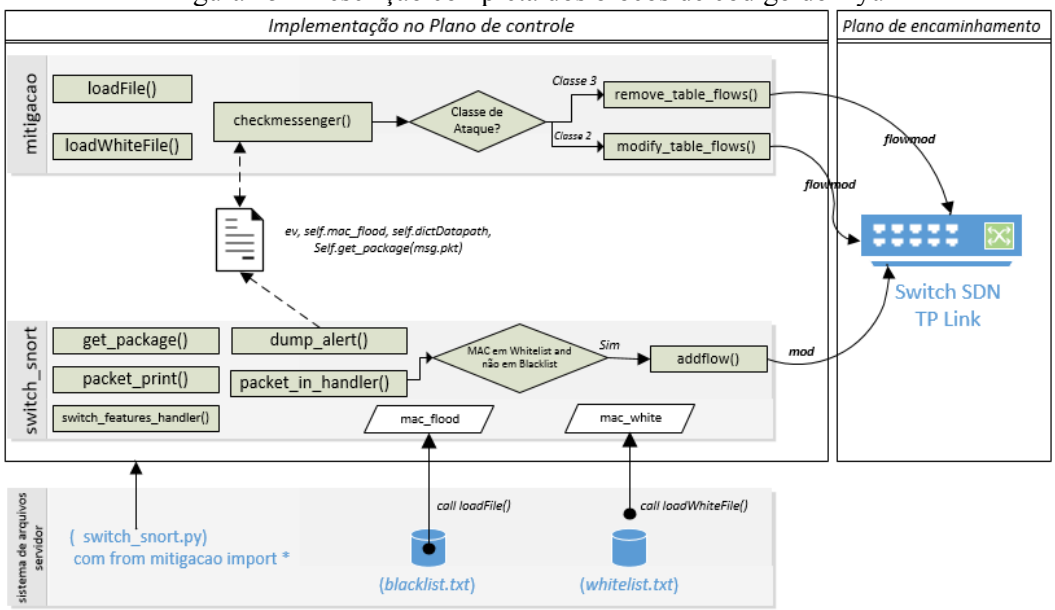
O tratamento aos ataques de Classe 2 é dado pelo método *modify_table_flow*, que foi programado para executar uma ação que consiste em enviar um comando para o *Switch OpenFlow* reescrever (*REWRITE*) todas as entradas de fluxo que estejam associadas com o endereço MAC que originou o ataque, com o objetivo de redirecionar o ataque para uma máquina preparada para executar uma *honeypot* para enganar o atacante e possibilitar ao administrador entender (analisar) a assinatura de ataque empregada.

Para os tipos de ataque Classe 3 o tratamento é dado pelo método *remove_table_flow*, programado para executar duas ações: (1) enviar um comando para o *Switch* retirar (*DROP*) todas as entradas de fluxo que estejam associadas ao endereço MAC que originou o ataque, com o objetivo de interromper o tráfego malicioso e (2) inserir o endereço MAC que originou o ataque em um arquivo de lista negra (*Blacklist*) para prevenir novas conexões deste equipamento considerado suspeito.

4.3.2 Detalhamento dos blocos de códigos do Ryu

Neste subcapítulo serão detalhados os aspectos que envolveram a solução por meio da aplicação do arcabouço do Ryu e o desenvolvimento de blocos de códigos apresentados na Figura 15 que possibilitaram a implementação dos mecanismos de proteção propostos nos objetivos do trabalho, através da programabilidade no plano de controle.

Figura 15 - Descrição completa dos blocos de código do Ryu



4.3.2.1 Código *switch_snort.py*

O arquivo de código *switch_snort.py* está disponível no site do projeto do arcabouço Ryu em (RYU, 2014) e serviu de referência para os estudos iniciais do projeto. Ele já contempla, por padrão, em sua implementação, o tratamento para os pacotes na rede com comportamento de um *Switch* na L2 e ainda possibilita que todo o tráfego na rede *OpenFlow* seja espelhado em uma de suas portas. Esse código padrão, no entanto, está limitado a receber as informações de ataque, por meio de uma mensagem *Packet_event*, para que seja tratado no controlado.

Como o tratamento das mensagens *Packet_event* na forma base do código *switch_snort.py* não está contemplada no arcabouço, foi necessário estender as funcionalidades no plano de controle por meio do desenvolvimento do código *mitigacao.py*, que será apresentado abaixo, possibilitando a implementação dos

mecanismos de reação com decisão centralizada no controlador. O arquivo *switch_snort.py* contempla os seguintes métodos apresentados na Tabela 7.

Tabela 7 - Métodos do código *switch_snort.py*

Método	Descrição
get_package()	Recebe como parâmetro o pacote da mensagem (*pkt*) e retorna um objeto do tipo *packet*
packet_print()	Printa informações do pacote (*pkt*) recebido
switch_features_handler()	Este método é responsável por configurar as entradas nas tabelas dos *Switches* indicando que enviem uma mensagem Packet_in para o controlador de forma que ele decida o que fazer com o pacote. Ele é executado quando o controlador recebe a resposta de um *Switch* para uma das mensagens de *features* enviadas anteriormente. Ryu envia mensagens *features* toda vez que é estabelecida uma conexão com um *Switch OpenFlow* por meio da execução deste método para cada *Switch* quando a rede é inicializada.
dump_alert()	Método que recebe as mensagens Packet_event enviadas pela ferramenta desenvolvida em python *pigrelay.py*, rodando no servidor de controlador Ryu, quando a ferramenta Snort identifica um determinado tipo de tráfego malicioso na rede que coincide com as assinaturas de ataque pré-configuradas no arquivo de configuração snort.rules do Snort.
packet_in_handler()	Este método é chamado quando o Ryu recebe uma mensagem do tipo Packet_in. Sua principal função é encaminhar o pacote referenciado pelo evento e instalar a entrada na tabela de fluxos correspondente e inclui outras funcionalidades.
addflow()	Método que adiciona uma entrada na tabela de fluxos do *Switch* correspondente. Este método para ser executado precisa receber como argumentos o dpid do Switch em que vai se instalar a entrada, os campos de *match* que identificam a entrada, assim como as *actions* a tomar com os pacotes correspondentes. Opcionalmente, se pode especificar a prioridade, o ID do *buffer* dos pacotes e o tempo de expiração (*idle timeout*)

Toda a lógica para permitir ou não o acesso de um determinado endereço MAC a àrede *OpenFlow*, com o recurso de *whitelist* ou *blacklist*, foi implementada dentro do método *packet_in_handler()*, responsável por receber e tratar todas as mensagens do tipo *Packet_in* da rede.

Neste sentido, o código abaixo identifica por meio de um bloco de comando usando *if* se um determinado MAC tem permissão ou não para acessar a rede por ter sido registrado previamente pelo administrador na *whitelist* ou se ele está na *blacklist* por ter sido considerado suspeito por um determinado tráfego malicioso detectado anteriormente.

```
1.  if src in self.mac_white:
2.          if src not in self.mac_flood and dst not in self.mac_flood:
3.              print ('Criando regra de fluxo...')
4.              if out_port != ofproto.OFPP_FLOOD:
5.                      match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
6.                      self.add_flow(datapath, 1, match, actions)
7.                      data = None
8.                      if msg.buffer_id == ofproto.OFP_NO_BUFFER:
9.                              data = msg.data
10.                     out = parser.OFPPacketOut(datapath=datapath,
                                buffer_id=msg.buffer_id,in_port=in_port, actions=actions,
                                data=data)
11.                    datapath.send_msg(out)
```

A primeira verificação feita no tratamento de qualquer tipo de mensagem *Packet_in* (Linha 1) é identificar se o MAC de origem da mensagem está na relação de lista branca (*whitelist*), lista esta que é atualizada pelo administrador e contém todos os endereços físicos que estão autorizados a se conectarem na rede *OpenFlow/SDN*. Se o MAC não estiver na *whitelist,* o pacote é simplesmente descartado e a entrada de fluxo na tabela do *Switch OpenFlow* não é criado, pois o comando *addflow()* não é chamado. Em contrapartida, se o MAC estiver na *whitelist* a segunda linha faz duas verificações, sendo uma para saber se o MAC de origem não está na *blacklist*, ou seja, se este endereço não é considerado suspeito e a outra tem por finalidade identificar se o MAC de destino também não é um equipamento suspeito. Essa última verificação tem por objetivo impossibilitar a criação de uma regra de fluxo que permita um outro dispositivo qualquer na rede comunicar-se com a máquina suspeita. Dessa forma, o equipamento com o MAC suspeito ficará totalmente isolado da rede, até que o administrador faça as correções necessárias neste equipamento, garantindo conformidade com a política da rede.

Após essas validações de garantir que o MAC está na *whitelist*, não é suspeito e que a solicitação de criação de entrada de fluxo não tem como destino um MAC também considerado suspeito, é que serão executados os comandos padrão do código do Ryu, disponível no site do projeto, seguindo o comportamento para criação das entradas de fluxo com a execução do comando *add_flow()* e o envio da mensagem para *datapath.send_msg(out)* para o *Switch OpenFlow*.

4.3.2.1. 2 Código *mitigacao.py*

A solução dada pelo desenvolvimento do código *mitigacao.py* estende tanto as características do Snort quanto as funcionalidades do plano de controle do arcabouço Ryu. Com relação ao Snort, o trabalho trata o fato de a ferramenta ser limitada a um IDS e permite que a solução apresentada por meio do ambiente derede disponibilize mecanismos de proteção com características de um IPS, realizando determinadas ações como bloquear um certo tipo de tráfego, considerado suspeito, ou até mesmo redirecionar esse tráfego para uma porta interligada a uma *honeypot*. No plano de controle, o código *mitigacao.py* foi desenvolvido com toda a lógica necessária para possibilitar que no controlador Ryu, de forma centralizada, trate de forma flexível, todas as mensagens *Packet_event* enviadas pelo Snort, na medida em que o tratamento foi dado por Classes de Ataques. No entanto, é importante destacar que a documentação disponível não foi suficiente para o entendimento completo das funções específicas do controlador *OpenFlow*, por falta de um detalhamento mais específico acerca das variáveis de entrada de cada método do arcabouço, onde foi exigido muitas vezes ao longo do esforço de implementação a realização de testes para verificar as diferentes e possíveis entradas até encontrar a entrada correta que atendesse ao contexto do trabalho. E como não era achado nenhum exemplo na *stack* ou em sites de dúvidas se apresentou como um dificultador para utilização dos mesmos e exigiu um maior esforço de tempo no desenvolvimento da solução. O arquivo *mitigacao.py* contempla os seguintes métodos apresentados na Tabela 8.

Tabela 8 - Métodos do código *mitigacao.py*

Método	Descrição
load_file()	Carrega em memória o conteúdo do arquivo *blacklist.txt*, arquivo este armazenado no diretório do projeto ryu no servidor que está rodando o controlador, em tempo de execução, retornando uma lista chamada de *mac flood*. Este método é empregado para possibilitar a verificação dos endereços MAC suspeitos, no tratamento das mensagens Packet_in, e evitar que eles estabeleçam uma nova conexão à rede quando forem bloqueados e incluídos em blacklist.
load_Whitefile()	Carrega em memória o conteúdo do arquivo *whitelist.txt*, arquivo este armazenado no diretório do projeto ryu no servidor que está rodando o controlador, em tempo de execução, retornando uma lista chamada de mac white. Este método é empregado para possibilitar a verificação dos endereços MAC autorizados a acessarem a rede

	no tratamento das mensagens Packet_in e somente permitir a criação de entradas de fluxos nos *Switches* daqueles dispositivos que foram incluídos em *Whitelist* pelo administrador.
checkmessenger()	Método responsável por toda a lógica de tratamento das mensagens Packet_event enviadas pelo Snort para tratamento no controlador.
remove_table_flow()	Remove as entradas de fluxos de um *Switch* cujos campos de *match* que coincidam com os especificados como argumento.
modify_table_flows()	Modifica as entradas de fluxos de um *Switch* que contém os campos de match que coincidam com os especificados como argumento.

Tanto o método de remoção (*remove_table_flows*) quanto o de modificação (*modify_table_flows*) empregados no desenvolvimento do código mitigacão estão disponíveis no arcabouço Ryu e foram empregados sem qualquer tipo de alteração. No entanto, o que viabilizou a execução dos mecanismos de proteção propostos no trabalho foi a manipulação dos argumentos passados para cada um destes métodos. O método empregado para remoção das entradas de fluxos no *Switch* tem a seguinte assinatura:

def remove_table_flows(self, datapath, table_id, match, instructions)

self:	É o próprio objeto
datapath	Identificador único do *Switch OpenFlow*
table_id	Identificador único da tabela de fluxo no *Switch OpenFlow*
match	Estrutura de correspondência para estabelecer uma comparação
instructions	Instruções que descrevem um processamento *OpenFlow* que ocorre quando um pacote corresponde a uma entrada de fluxo. Pode tanto modificar um processamento ou conter uma lista de ações a serem aplicadas.

Mesmo estando o método de remoção disponível no arcabouço do Ryu foi empregado um bom esforço de tempo para entender que informações deveriam ser passadas como argumentos na função. Esse esforço para se obter uma melhor compreensão de como se empregar o método se resumiu basicamente na leitura do *JavaScript Object Notation* (JSON[22]) que é enviado pelo método. Outro detalhe é

[22] É um modelo para armazenamento **e** transmissão de informações no formato texto, bastante utilizado por aplicações Web devido a sua capacidade de estruturar informações de uma forma

que esse processo de remoção das entradas de fluxo do *Switch* por padrão não pode ser feito com mais de uma condição o que ia de encontro à proposta deste trabalho, pois a solução proposta envolve duas ações, uma de remover todas as entradas de fluxos que têm o MAC suspeito como porta de entrada (*in_port*) e a outra de remover todas as entradas de fluxos que tem o MAC suspeito como endereço MAC de destino (*dst*), para evitar se ter uma tabela com regras sem nenhuma finalidade e principalmente para que um outro dispositivo qualquer tente manter conexão com o dispositivo suspeito. Por isso, como será apresentado abaixo foi necessário o emprego de duas regras de comparação *match*, assim como duas chamadas ao método *remove_table_flows* para execução das duas ações separadamente. O método empregado para reescrita das entradas de fluxos no *Switch* tem a seguinte assinatura:

def modify_table_flows(self, datapath, table_id, match, instructions)

self	É o próprio objeto
datapath	Identificador único do *Switch OpenFlow*
table_id	Identificador único da tabela de fluxo no *Switch OpenFlow*
match	Estrutura de correspondência para estabelecer uma comparação
instructions	Instruções que descrevem um processamento *OpenFlow* que ocorre quando um pacote corresponde a uma entrada de fluxo. Pode tanto modificar um processamento ou conter uma lista de ações a serem aplicadas.

O método *modify_table_flows* em comparação com o tratamento anterior tem menos detalhes na documentação, mas como a base de funcionamento de comunicação entre o Ryu e o *Switch* já era de conhecimento do autor da pesquisa, devido ao trabalho de implementação do mecanismo de remoção, isso facilitou o seu desenvolvimento. Apesar disso, o entendimento completo de como funcionam algumas constantes passadas como parâmetros ainda vai demandar um estudo mais aprofundado, como por exemplo, a diferença entre OFPFC_MODIFY e OFPFC_MODIFY_STRICT. Toda a lógica de programação concebida nos mecanismos de reação propostos neste trabalho foi implementada por meio do método *checkmessenger()*, que é responsável pela análise e tratamento por Classes

bem mais compacta do que a conseguida pelo modelo XML, tornando mais rápido o *parsing* dessas informações.

de ataques das mensagens *Packet_event* enviadas pela ferramenta Snort ao controlador.

O *checkmessenger* recebe da função *_Dump_alert,* disponível no código *switch_snort.py,* e as informações da mensagem *Packet_event* como parâmetros, como detalhado abaixo:

```
1. def _dump_alert(self, ev):
2.       msg = ev.msg
3.       self.mac_flood = self.mitigacao.checkmessenger(ev,self.mac_flood,
                              self.dictDatapath,self.get_package(msg.pkt))
```

ev	Dados da mensagem Packet_event
mac_flood	Lista de MAC
dictDatapath	Dicionário contendo o Datapath, table_id e todos os endereços MAC e suas respectivas portas de conexão
get_package	Pacote da mensagem Packet_event com informações como MAC e IP do dispositivo considerado suspeito

A função *_Dump_alert* recebe como retorno do método *checkmessenger* a lista de MAC atualizada, e é armazenada na variável *mac_flood*. A assinatura do método *checkmessenger* está definida da forma apresentada abaixo:

def checkmessenger(self,ev,mac_flood,dictDatapath,pkt)

ev	Dados da mensagem Packet_event
mac_flood	Lista de MAC
dictDatapath	Dicionário (*lista*) contendo o *Datapath*, *table_id* e todos os endereços MAC e suas respectivas portas de conexão
pkt	Pacote da mensagem *Packet_event* com informações como MAC e IP do dispositivo considerado suspeito

Para uma análise inicial mais detalhada do método *checkmessenger*, é apresentado o trecho abaixo, no qual são definidas algumas variáveis importantes para o tratamento dos eventos de segurança.

```
def checkmessenger(self,ev,mac_flood,dictDatapath,pkt):
   msg = ev.msg
   msgconvert = str(msg.alertmsg[0])
…..
```

Tão logo os dados da mensagem *Packet_event*, recebidos por meio do parâmetro de entrada ev (*evento*), chegam ao método *checkmessenger*, são

```
alert icmp any any -> $HOME_NET any (msg: "ICMP test detected"; GID:1; sid:10000001; rev:001; classtype:icmp-event)
```

armazenadas na variável msg. O método *checkmessenger* não precisa tratar todas as informações de ataque, mas apenas a parte da mensagem que informa o tipo de alerta gerado e recebido no controlador. Por esse motivo que a variável *msgconvert* recebe apenas o primeiro campo [0] de *alertmsg* da variável *msg*, que por padrão vem no formato *unicode*. Como foi adotada a estratégia de tratar os eventos de segurança de forma flexível e não convencional, a implementação exige uma busca de *substrings* nas informações passadas pelas mensagens *Packet_event*, permitindo assim identificar a que Classe de Ataque o evento de segurança está relacionado. Desse modo, a informação recebida no formato *unicode* foi convertida para texto, por meio da função *str*, e armazenada na variável *msgconvert*. Esta conversão viabilizou a identificação da Classe de ataque, especificada previamente como uma assinatura de ataque do Snort e definida no arquivo *snort.rule*, como apresentado abaixo:

O tratamento dado para os ataques de Classe 3 foi definido dentro do bloco de código abaixo:

```
        ....
1.  if msgconvert.find ("Class_3") !=-1:
2.      lista = open("blacklist.txt","a+")
3.      escrevendo = str(pkt.get_protocol(ethernet.ethernet).src+"\n")
4.      l = lista.readlines()
5.      print("#+++   Packet_Event treatment - Class 3 attack   +++#")
6.      print("#**  Attempt to suspicous activity on the network **#")
7.      print (l)
8.      if escrevendo not in l:
9.          lista.write(escrevendo)
10.         mac_flood.append(pkt.get_protocol(ethernet.ethernet).src)
11.     print("#*** Inserting suspect MAC Address in Blacklist ***#")
12.     print("alertmsg: %s" % "".join(msg.alertmsg))
13.     lista.close()
14.     print(pkt.get_protocol(ethernet.ethernet).src)
15.     mc = pkt.get_protocol(ethernet.ethernet).src
16.     parser = dictDatapath[mc][0].ofproto_parser
17.     empty_match = parser.OFPMatch(in_port=dictDatapath[mc][2])
18.     empty_match2 = parser.OFPMatch(eth_dst=mc)
19. self.remove_table_flows(dictDatapath[mc][0],dictDatapath[mc][1],empty_match,[])
20. self.remove_table_flows(dictDatapath[mc][0],dictDatapath[mc][1],empty_match2,[])
21. return mac_flood
        ….
```

Se a *substring* "Classe 3" do comando *find* na variável *msgconvert* (Linha 1), retornar qualquer coisa diferente de -1, conclui-se que se trata de uma assinatura de ataque de Classe 3 informada pelo Snort. A partir daí a execução

inicia-se dentro do laço do *if*, onde o primeiro comando busca abrir o arquivo de *blacklist.txt* e todos os endereços MAC que estejam registrados neste arquivo são armazenados na variável *lista*. Na terceira linha a variável *escrevendo* recebe o endereço MAC que originou o ataque, por meio do método *get_protocol* aplicado ao pacote da mensagem (*pkt*) usando o parâmetro *ethernet* da classe *ethernet* que obtêm o endereço MAC do atacante que está no atributo *.src* .

Já na quarta linha é definida uma variável *l* que armazena todos os endereços MAC que foram armazenados na variável *lista*, separados por linha, por meio da função *readlines()*. Na quinta, sexta e sétima linha são impressas como saída de comando no controlador as informações de que o tratamento dos ataques de Classe 3 estão em andamento, sendo que na sétima linha tem-se ainda como saída a impressão na tela de todos os endereços MAC que já estejam em *blacklist*, para que facilite o acompanhamento por parte do administrador.

Logo após, na Linha 8, é feita uma verificação para identificar se o endereço MAC que originou o ataque já não se encontra em *blacklist*. Esta verificação é necessária para evitar que os endereço MAC considerados suspeitos venham ser registrado mais de uma vez no arquivo. Nas primeiras versões do código as simulações de ataques que foram realizadas demonstraram a necessidade de implementar esta parte do código, pois a ausência deste tratamento permite que um mesmo endereço MAC seja registrado inúmeras vezes.

Já para o caso do endereço MAC considerado suspeito (variável *escrevendo*), não incluso ainda em *blacklist* (l), os comandos dentro do laço do *if* serão executados. Dentro do laço de execução o primeiro comando na Linha 9 atualiza o arquivo *blacklist.txt* por meio da função *write* aplicada em *lista* usando a variável *escrevendo*. O segundo comando na Linha 10 atualiza a lista de endereços MAC, na variável *mac_flood*, em tempo de execução, por meio da função *append* que obtêm o endereço MAC por meio do método *get_protocol* aplicado ao pacote da mensagem (*pkt*), usando o parâmetro *ethernet* da classe *ethernet* que obtêm o endereço MAC do atacante que está no atributo *.src* . Estas duas ações são necessárias para manter atualizada tanto a lista de endereços MAC quanto o arquivo *blacklist.txt* e dar efetividade ao tratamento de controle de acesso à rede *OpenFlow*, evitando que um dispositivo suspeito mantenha sua conexão à rede após ser bloqueado.

Para melhor acompanhamento do administrador, a Linha 11 apresenta na saída de comando o tipo de tratamento em andamento, na Linha 12 é impresso na saída de comando o tipo de mensagem de alerta enviado pela ferramenta do Snort e na Linha 13 é apresentado qual o endereço MAC foi considerado suspeito. Já a partir da Linha 14 inicia-se a lógica que permite que para todo o tipo de ataque Classe 3, todas as entradas de fluxos no *Switch OpenFlow* relacionado ao MAC suspeito sejam removidas, assim como permitirá também remover todas as entradas de fluxos que tenham o endereço MAC suspeito como destino, evitando que um outro dispositivo tente manter conexão com o equipamento suspeito.

Logo, na Linha 15, a variável *mc* recebe o endereço MAC suspeito por meio do método *get_protocol* aplicado ao pacote da mensagem (*pkt*) usando o parâmetro *ethernet* da classe *ethernet* que obtêm o endereço MAC do atacante que está no atributo .*src*. Na Linha 16 é feito inicialmente uma busca na lista *dictPath* usando como chave a variável *mc* para encontrar todas as referências das entradas de fluxos que estejam atualmente aplicadas no *Switch* e que tenham como endereço de origem no primeiro campo da lista [0] o endereço MAC suspeito. As referências as entradas de fluxos encontradas no *Switch OpenFlow* são armazenadas na variável *parser*. No entanto, as informações obtidas precisam sofrer um *parser* por meio do objeto *ofproto_parser*, necessário para converter o formato da informação de forma a permitir passar como parâmetro para função OFPMatch que será apresentada abaixo e possibilitar a comunicação com o *Switch OpenFlow* no formato esperado pelo *Switch*. O *ofproto_parser* é um objeto que modifica o formato da informação (*mc*) com o objetivo de compatilizar o conteúdo no formato adequado para o *Switch OpenFlow*.

Já nas Linhas 17 e 18 são criados dois padrões de *match*, ou seja, padrões de comparação para serem enviados como argumento para o método *remove_table_flows*. A variável *empty_match* recebe como padrão de comparação a porta de entrada (*in_port*) que está conectada ao MAC considerado suspeito, por meio de uma procura na lista *dictDatapath* usando como chave a variável *mc* no campo [2]. E como segundo padrão de comparação foi usada a variável *empty_match2* que recebe o próprio MAC considerado suspeito como MAC de destino. Desta forma foi possível por meio da primeira chamada ao método *remove_table_flow*, na Linha 19, enviar um comando ao *Switch OpenFlow* para

remover todas as entradas de fluxos que tenham como padrão de comparação o endereço MAC de origem o próprio MAC suspeito. E para garantir que nenhum outro dispositivo consiga manter uma conexão com o dispositivo suspeito foi feita uma segunda chamada ao método *remove_table_flow*, na Linha 20, removendo também todas as entradas de fluxos que tenham como destino o endereço MAC suspeito. Com este tratamento foi possível garantir o bloqueio do dispositivo considerado suspeito, que nenhum outro dispositivo irá tentar estabelecer conexão com este dispositivo e uma tabela de fluxos atualizada.

O tratamento dado para os ataques de Classe 2 foram definidos dentro do bloco de código abaixo:

```
    ....
1.  if msgconvert.find ("Class_2") !=-1:
2.      print("#+++   Packet_Event treatment - Class 2 attack   +++#")
3.      print("#**  Attempt to suspicous activity on the network **#")
4.      print("alertmsg: %s" % "".join(msg.alertmsg))
5.      mc = pkt.get_protocol(ethernet.ethernet).src
6.      parser = dictDatapath[mc][0].ofproto_parser
7.      dst = pkt.get_protocol(ethernet.ethernet).dst
8.      empty_match = parser.OFPMatch(in_port=dictDatapath[mc][2],eth_dst=dst)
9.      actions = [parser.OFPActionOutput(4)]
10.       instructions = [parser.OFPInstructionActions(dictDatapath[mc][0].
                              ofproto.OFPIT_APPLY_ACTIONS,actions)]
11.     self.modify_table_flows(dictDatapath[mc][0],dictDatapath[mc][1],
                                    empty_match,instructions)
12.     return mac_flood
```

Se a substring "Classe 2" do comando *find* na variável *msgconvert*, expresso na Linha 1, retornar qualquer coisa diferente de -1, conclui-se que se trata de uma assinatura de ataque de Classe 2 informada pelo Snort. A partir daí inicia-se a execução do laço dentro do *if*, onde nas segunda, terceira e quarta linhas são impressas como saída de comando informações de que o tratamento dos ataques de Classe 2 está em andamento, sendo que na quarta linha tem-se como saída a impressão na tela o conteúdo da variável *msg*, apenas relativo ao alerta (*alertmsg*) enviado na mensagem *Packet_event*.

Já a partir da Linha 5 inicia-se a lógica que permitirá que para todo o tipo de ataque Classe 2, todas as entradas de fluxos no *Switch OpenFlow* relacionado ao MAC suspeito sejam reescritas de forma a possibilitar que o tráfego suspeito seja redirecionado para uma *honeypot*. Na Linha 5 a variável *mc* recebe o endereço MAC suspeito por meio do método *get_protocol* aplicado ao pacote da

mensagem (*pkt*) usando o parâmetro *ethernet* da classe *ethernet* que obtêm o endereço MAC do atacante que está no atributo .*src*.

Na Linha 6 é feito inicialmente uma busca na lista *dictPath* usando como chave a variável *mc* para encontrar todas as referências as entradas de fluxos que estejam atualmente aplicadas no *Switch* que tenham como endereço de origem no primeiro campo da lista [0] o endereço MAC suspeito. As referências as entradas de fluxos encontradas no *Switch OpenFlow* são armazenadas na variável *parser* e de idêntico modo como foi apresentado no tratamento de ataques de Classe 3 precisam passar pelo objeto *ofproto_parser* para ser possível compatibilizar o formato do argumento ao formato esperado pelo *Switch OpenFlow*.

Como padrão de comparação foi usado a variável *empty_match* na Linha 8, que recebeu a porta de entrada (*port_in*) onde estão conectados o MAC suspeito e o endereço MAC do dispositivo atacado. E diferentemente do tratamento dado pelo *remove_tables_flow* o método *modify_table_flows* necessita receber como argumento as ações a serem realizadas e as instruções que serão passadas para que seja possível reescrever as entradas de fluxos na tabela do *Switch OpenFlow*. Neste sentido foi definido em *actions* que a porta de saída *out_port* será a porta 4 do *Switch* que está conectado a honeypot e em instructions, Linha 10, é aplicado a todas as regras correspondentes ao *empty_match* descrito acima. E por fim o método *modify_table_flows* envia um comando para o *Switch OpenFlow* passando como argumento o *datapath* e o *id_table* do *Switch* a qual o MAC suspeito está associado com a ação de reescrever todas as entradas de fluxos do MAC suspeito e trocar as portas de saída para a porta 4 do *Switch* que está interligado a *honeypot*.

5 SIMULAÇÃO E RESULTADOS OBTIDOS

Este capítulo tem como objetivo apresentar os procedimentos adotados para simulação de testes e validação, tanto do ambiente de rede real *OpenFlow/SDN* quanto dos mecanismos de proteção propostos no contexto deste trabalho. A avaliação e validação do ambiente envolveu procedimentos para se verificar as funcionalidades da rede *OpenFlow* e a integração do controlador Ryu com a ferramenta Snort.

Os procedimentos adotados para avaliação e validação dos mecanismos de proteção e análise de intrusão foram realizados por meio de simulação, a partir da construção de diferentes cenários, para identificar se cada tipo de ataque aplicado a um cenário específico teria o tratamento adequado, de acordo com a solução desenvolvida e implementada pelo pesquisador no plano de controle do arcabouço Ryu. A validação do ambiente foi realizada por meio de testes conduzidos pelo pesquisador e direcionados aos clientes da rede SDN e ao próprio *Switch OpenFlow*, ressaltando que neste ambiente, uma parte substancial do tráfego capturado terá origem ilícita ou maliciosa, ou seja, estará comprometida por códigos maliciosos.

O objetivo é demostrar 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 SDN e sua flexibilidade por meio da programabilidade do controlador da rede.

5.1 AVALIAÇÃO E VALIDAÇÃO DO AMBIENTE DE REDE

Toda a estrutura do projeto foi implementada utilizando *hardware* dedicado para garantir o melhor desempenho na avaliação e validação do ambiente de segurança proposta. Como apresentado na Figura 7, o ambiente possui três redes distintas: a rede SDN 172.16.10.0/24 em que estão conectados os nós da rede OpeFlow; a rede do servidor que está executando 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) 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 empregada uma máquina real, como um dos clientes da rede, com o sistema operacional Linux Kali (IP 172.16.10.251) conectado à porta 4 do *Switch OpenFlow* para simular alguns ataques. Com a finalidade de demonstrar os mecanismos de detecção e reação às ameaças de segurança foi empregado no ambiente proposto um *Switch* comercial de prateleira (*Commercial off-the-shelf* - COTS), modificado (CPQD, 2012), 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.

O tráfego foi gerado com o uso das ferramentas t50[23] e Nmap[24] para Linux e para implementação do mecanismo de detecção de intrusão foi empregada a ferramenta Snort. Como forma de viabilizar o armazenamento dos dados gerados pela ferramenta Snort fez-se uso do gerenciador de banco de dados MySQL. Por fim, para aumentar a eficiência na gravação dos registros dos possíveis ataques, utilizou-se 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.

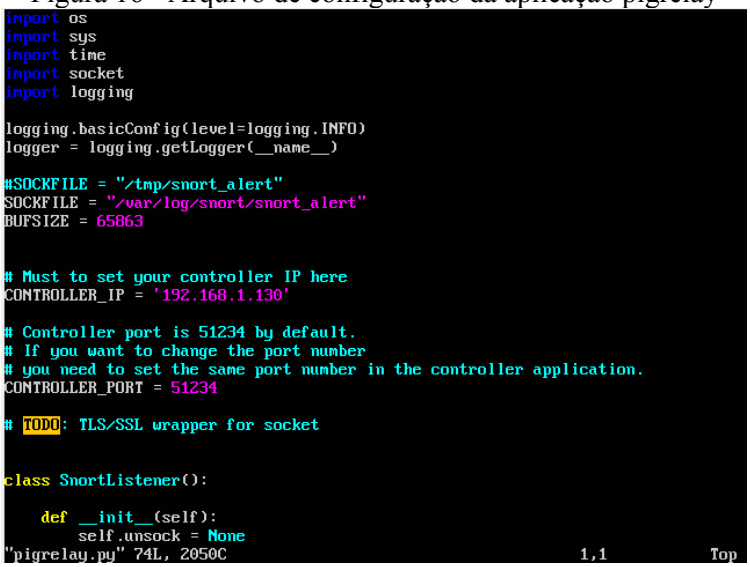
Os mecanismos de reação foram implementados por meio de codificação no arquivo *mitigacao.py* no controlador, encarregado de tratar os eventos *Packet_event* gerados pela ferramenta Snort e enviados ao controlador Ryu por meio de uma aplicação desenvolvida em python chamada *pigrelay*, disponível em (PIGRELAY, 2015). Essa aplicação em execução na máquina do Snort foi

[23] 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.

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

configurada com os parâmetros apresentados na Figura 16, para se adequar o ambiente de rede proposto, com a finalidade de possibilitar que as mensagens de alerta geradas sejam encaminhadas ao controlador por meio de um *Unix Domain Socket* usando soquetes de rede.

Figura 16 - Arquivo de configuração da aplicação pigrelay



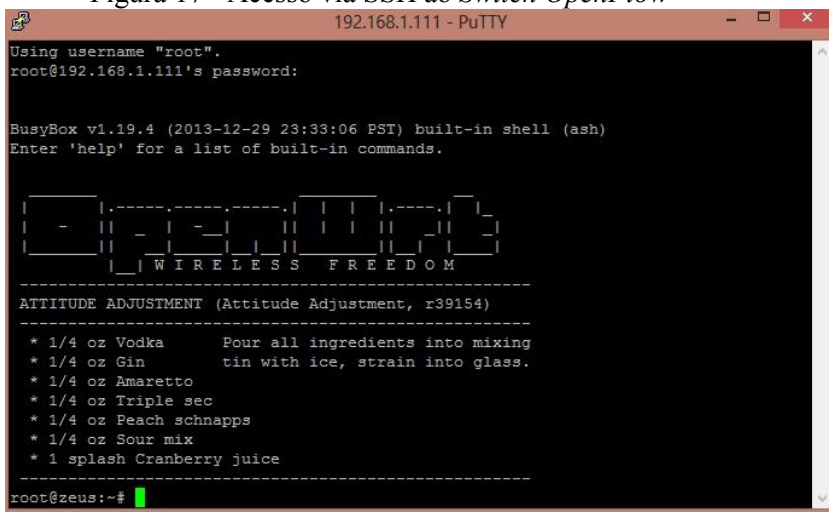
```
import os
import sys
import time
import socket
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

#SOCKFILE = "/tmp/snort_alert"
SOCKFILE = "/var/log/snort/snort_alert"
BUFSIZE = 65863


# Must to set your controller IP here
CONTROLLER_IP = '192.168.1.130'

# Controller port is 51234 by default.
# If you want to change the port number
# you need to set the same port number in the controller application.
CONTROLLER_PORT = 51234

# TODO: TLS/SSL wrapper for socket


class SnortListener():

    def __init__(self):
        self.unsock = None
"pigrelay.py" 74L, 2050C                                  1,1           Top
```

Em linhas gerais a ferramenta *pigrelay* ao ser executada envia ao controlador no IP 192.168.1.130 e porta 51234 os eventos de segurança gerados pela ferramenta do Snort e armazenados em /var/log/snort/snort_alert. Como ferramenta de análise para os alertas gerados pelo Snort está disponível no ambiente de rede proposto a ferramenta BASE que foi instalada na mesma máquina do Snort.

A validação do ambiente foi feita por meio de testes de conectividade do *Switch* TPLink com os nós da rede. A verificação do funcionamento da ferramenta Snort, que coleta tráfego de dados real de forma promíscua na interface *3* do roteador, que empregou o *hardware* TP-Link WR1043ND.

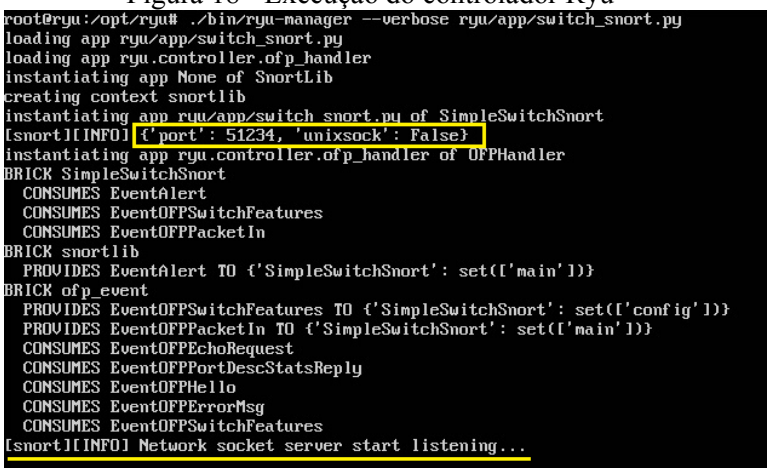
Para o suporte ao protocolo *OpenFlow* 1.3 o equipamento recebeu uma versão de sistema operacional OpenWrt, compilada a partir da versão “trunk” do OpenWRT (*Barrier Breaker*). A Figura 17 abaixo demonstra uma conexão bem-sucedida via acesso remoto usando o comando SSH.

Figura 17 - Acesso via SSH ao *Switch OpenFlow*



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 18 apresenta o comando *ryu-manager* sendo executado em modo *verbose* e rodando o arquivo *switch_snort.py,* implementado em *python,* que, por sua vez, executa outra aplicação (*mitigacao.py*), desenvolvida no contexto deste projeto e apresentada anteriormente.

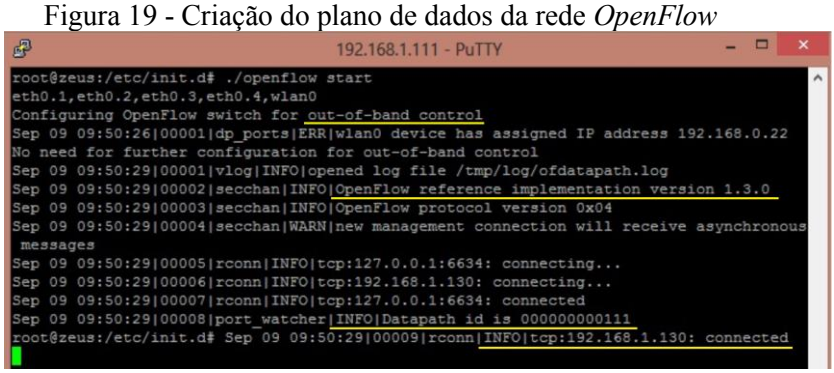
Figura 18 - Execução do controlador Ryu



Como demonstrado na Figura 18, ao final da execução do comando, o controlador Ryu fica aguardando a conexão do *Switch OpenFlow* autorizado e a outra informação destacada demonstra que o controlador ficará aguardando na porta *51234* as notificações *Packet_event* que serão encaminhadas por meio da aplicação *pigrelay.py* rodando na mesma máquina da ferramenta Snort. 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 19.

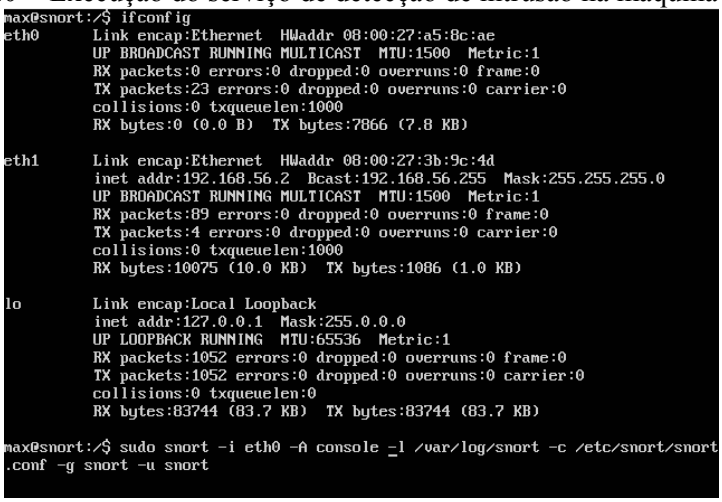
Figura 19 - Criação do plano de dados da rede *OpenFlow*



O serviço *OpenFlow* configura o *Switch* para funcionar em modo *out-of-band*, com suporte ao OpenFlow 1.3, criando um *datapath* ou plano de dados, com o *id* 000000000111 que possibilitará 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*.

Com a rede pronta, chegou a hora de verificar o funcionamento da integração da ferramenta Snort com o controlador Ryu. Na Figura 20 é possível ver o comando empregado para executar a ferramenta Snort:

Figura 20 - Execução do serviço de detecção de intrusão na máquina Snort

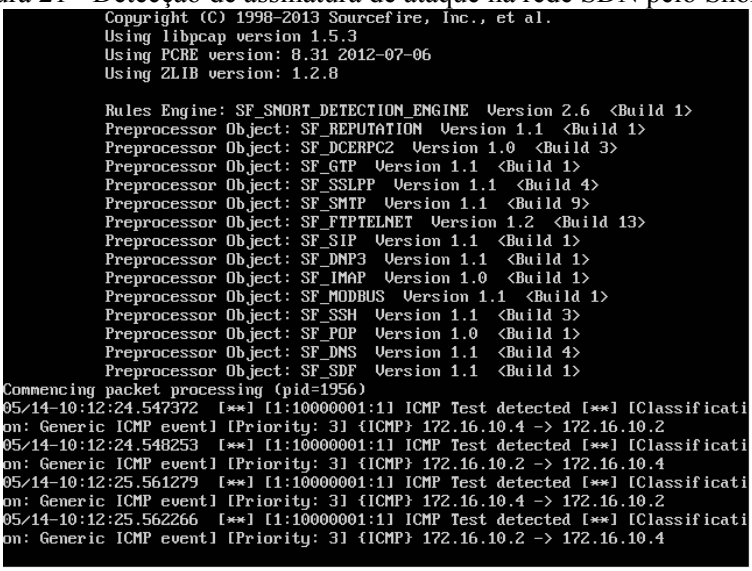


Para realização do teste de validação inicial do ambiente, o Snort foi configurado para detectar um simples tráfego de ICMP entre os nós da rede por meio de uma regra inserida no arquivo *local.rules* conforme descrito abaixo:

```
alert icmp any any -> any any (msg: "ICMP test detected";
GID:1; sid:10000001; rev:001; classtype:icmp-event)
```

Conforme apresentado na Figura 21 é possível identificar o recebimento do tráfego ICMP pelo Snort, considerado no contexto da validação do ambiente como um tráfego malicioso, gerado a partir de um comando ping do nó com o IP 172.16.10.4 direcionado ao nó 172.16.10.2.

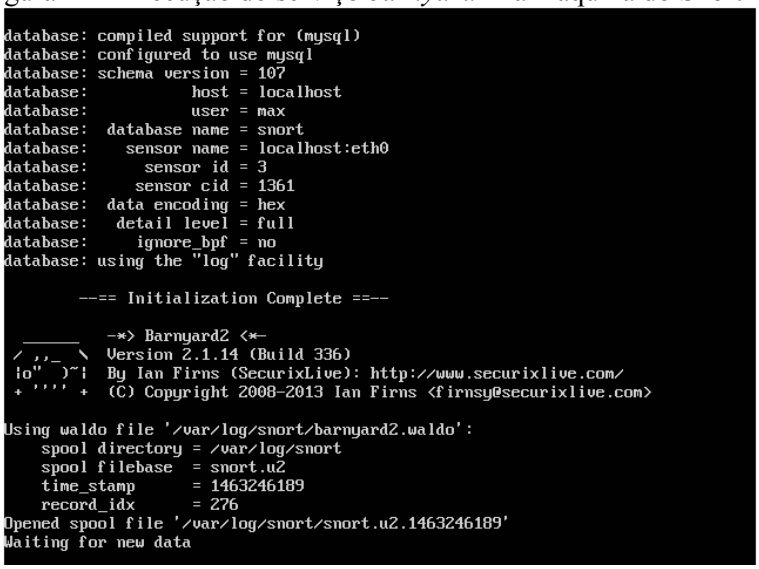
Figura 21 - Detecção de assinatura de ataque na rede SDN pelo Snort



Para viabilizar o armazenamento dos dados gerados pela ferramenta Snort foi utilizado o gerenciador de banco de dados MySQL. Visando 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 (FLORES, 2004), 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.

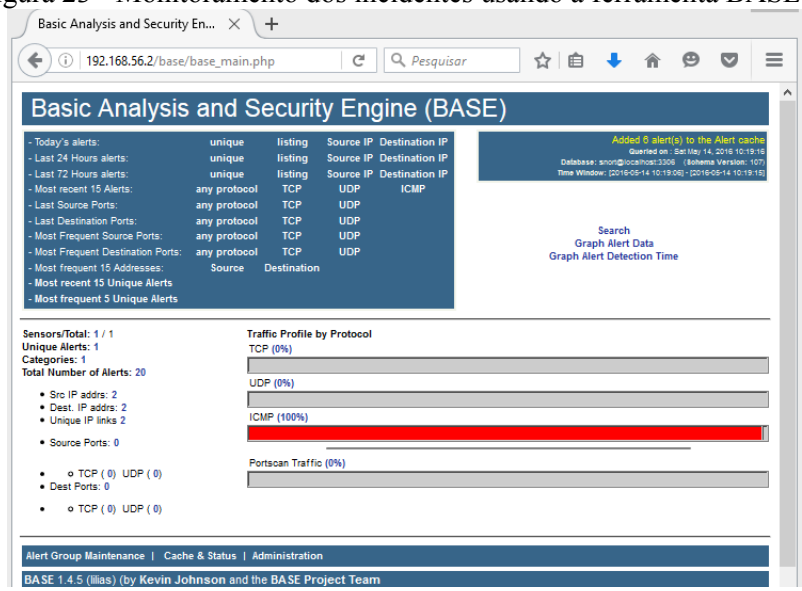
Na Figura 22 é possível ver o comado empregado na execução do serviço *barnyard2*.

Figura 22 - Execução do serviço *barnyard2* na máquina do Snort



Na Figura 23 é mostrada a tela de alertas da ferramenta BASE apresenta os registros de tentativas de intrusão coletados pelo Snort e salvos no banco de dados pelo pós-processador Barnyard. Nesse caso específico da validação do ambiente a ferramenta BASE está apresentando apenas os testes realizados por meio do tráfego icmp de validação inicial. A ferramenta BASE é uma ferramenta de navegação para análise de dados, construída utilizando a linguagem de programação PHP.

Figura 23 - Monitoramento dos incidentes usando a ferramenta BASE



Após a descrição sucinta do ambiente de rede e de todo processo adotado em sua avaliação e validação, será detalhado abaixo a metodologia empregada

neste trabalho para avaliaçãoe validação dos mecanismos de proteção implementados no plano de controle aplicados no ambiente de rede real OpenFlow/SDN em ambiente de *Testbed.* A solução adotada no contexto deste trabalho foi simular tráfegos de rede aplicados em diferentes cenários abaixo descritos, que se apresentaram bem consistentes e foram importantes à demonstração de como os objetivos desta dissertação foram alcançados:

a) CENÁRIO 1: validação do recurso de ***whitelist***.

b) CENÁRIO 2: validação do mecanismo de detecção.

c) CENÁRIO 3: validação do mecanismo de reação com ação de ***DROP*** e inclusão do MAC suspeito em ***blacklist***.

d) CENÁRIO 4: validação do recurso de ***blacklist***.

e) CENÁRIO 5: validação do mecanismo de reação com ação de reescrever (***REWRITE)*** e envio (reencaminhar) de tráfego suspeito para uma ***honeypot***.

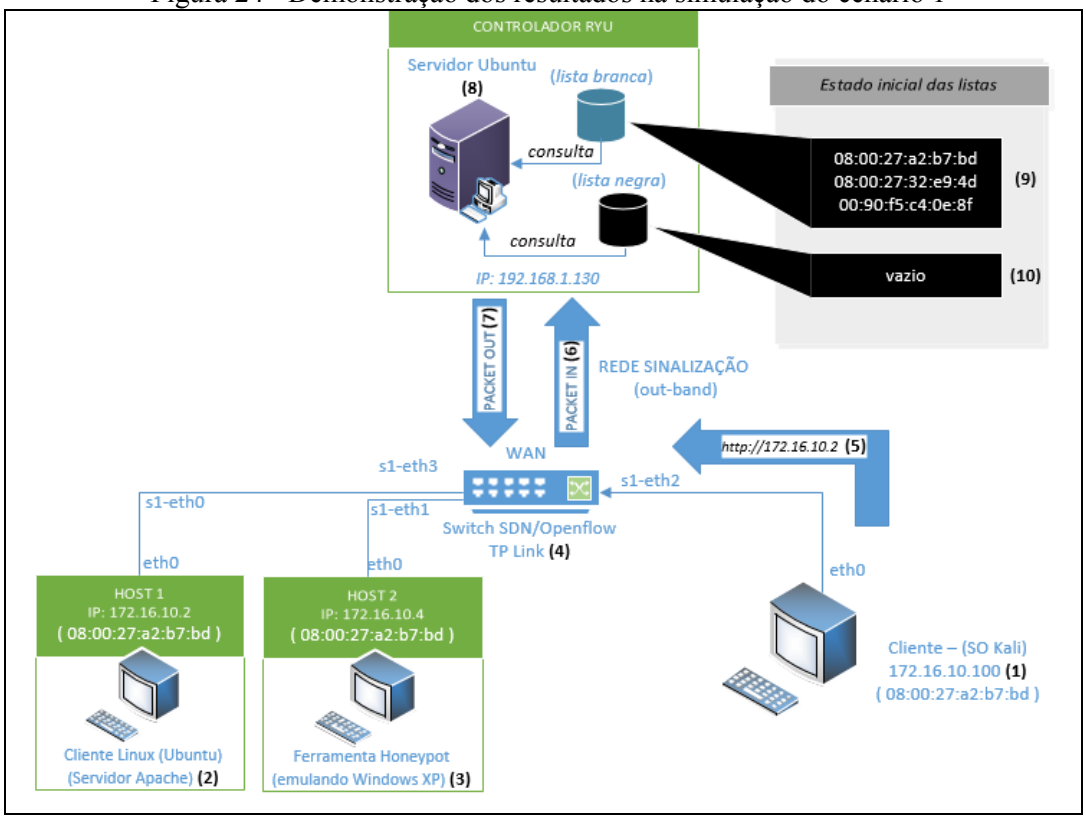
5.2 CENÁRIO 1: VALIDAÇÃO DO RECURSO DE *WHITELIST*

O primeiro recurso de proteção a ser demonstrado é o processo de admissão dos equipamentos clientes na rede *OpenFlow*, Figura 24, por meio da verificação de quais equipamentos estão autorizados e foram previamente cadastrados pelo administrador da rede em uma lista branca (*whitelist*), possuindo permissão para acessar a rede. Esse recurso foi o último a ser implementado, sendo resultado do esforço e aprendizado de implementação do recurso de *blacklist,* demonstrado nos cenários 3 e 4, que utiliza a mesma lógica e linha de raciocínio no desenvolvimento, mas com propósito diferente.

Neste cenário a simulação inicia a partir do momento em que o cliente de rede, rodando o OS Kali **(1)** no endereço MAC | 08:00:27:a2:b7:bd |, tenta acessar o site web em execução no *host* 1 **(2),** no endereço MAC | 08:00:27:a2:b7:bd |. Como não é encontrada uma entrada de fluxo no *Switch OpenFlow* que permita a comunicação entre os *hosts*, o *Switch OpenFlow* **(4)** encaminha ao Controlador Ryu uma solicitação de criação de entrada de fluxo por meio de uma mensagem *Packet_in* **(5)**. O Controlador **(6)** em execução no servidor com endereço IP | 192.168.1.130 |, ao tratar a mensagem *Packet_in* por meio do método *Packet_in_handler()*, realiza duas verificações. A primeira consiste em identificar

se o MAC | 08:00:27:a2:b7:bd | é de um dispositivo autorizado e a segunda consiste em verificar se esse mesmo MAC não é considerado suspeito. Essas duas verificações garantem o acesso somente a dispositivos autorizados e confiáveis.

Figura 24 - Demonstração dos resultados na simulação do cenário 1



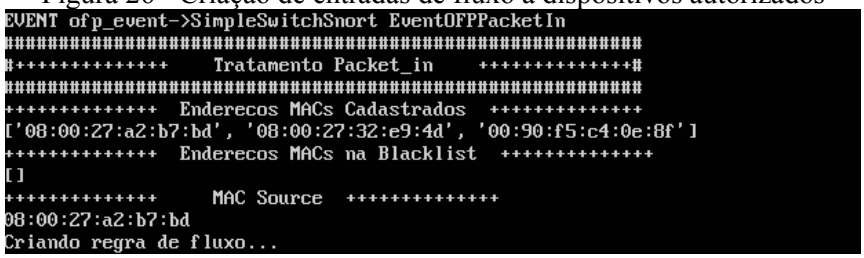
Os endereços MAC dos dispositivos autorizados são previamente registrados pelo administrador em uma lista branca (w*hitelist*) **(9)** no arquivo *whitelist.txt*, que será consultado e os endereços MAC cadastrados serão instanciados e armazenados em uma *lista* criada, em tempo de execução, quando o controlador estiver em execução. O mesmo tratamento é dado ao recurso de *blacklist*, sendo que neste caso a inserção de endereços MAC na lista se dá por meio de tratamento no controlador para ataques do tipo de Classe 3 e que foi implementado no contexto deste trabalho. Na Figura 25 é possível identificar o estado inicial dos arquivos *whitelist.txt* e *blacklist*.

Figura 25 - Estado inicial dos arquivos *whitelist* e *blacklist* no cenário 1

```
root@ryu:/opt/ryu# cat whitelist.txt
08:00:27:a2:b7:bd
08:00:27:32:e9:4d
00:90:f5:c4:0e:8f
root@ryu:/opt/ryu# cat blacklist.txt
root@ryu:/opt/ryu#
```

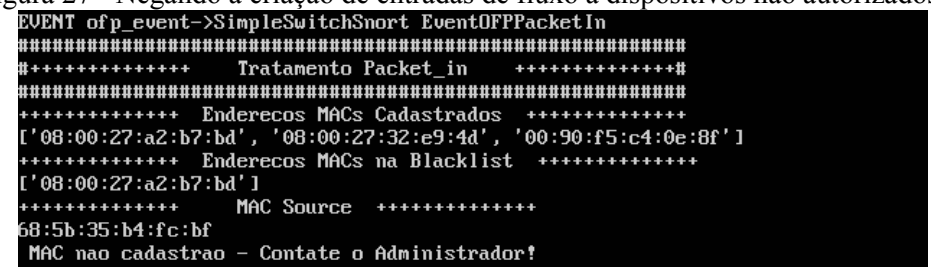
Para esta demonstração, como apresentado na Figura 26, o dispositivo com o endereço MAC | 08:00:27:a2:b7:bd |, quando tenta acessar o *host* 1 e o *Switch* requisita a criação de uma entrada de fluxo ao controlador, por meio de mensagens do tipo *Packet_in* **(5)**, terá sua requisição aceita e o controlador enviará ao Swich uma mensagem Packet_out **(7)** com a ação de criar a respectiva entrada de fluxo solicitada, uma vez que o MAC | 08:00:27:a2:b7:bd é de um dispositivo autorizado e não está registrado como suspeito na *blacklist*.

Figura 26 - Criação de entradas de fluxo a dispositivos autorizados



Em contrapartida, como apresentado na Figura 27, um dispositivo com o endereço MAC | 68:5b:35:b4:fc:bf |, quando tenta acessar o *host* 1 e o *Switch* requisita a criação de uma entrada de fluxo, por meio de mensagens do tipo *Packet_in,* terá sua requisição negada e receberá a informação de que deve contactar o Administrador da rede porque o endereço em questão não consta como dispositivo autorizado a estabelecer conexão na rede *OpenFlow/SDN*.

Figura 27 - Negando a criação de entradas de fluxo a dispositivos não autorizados



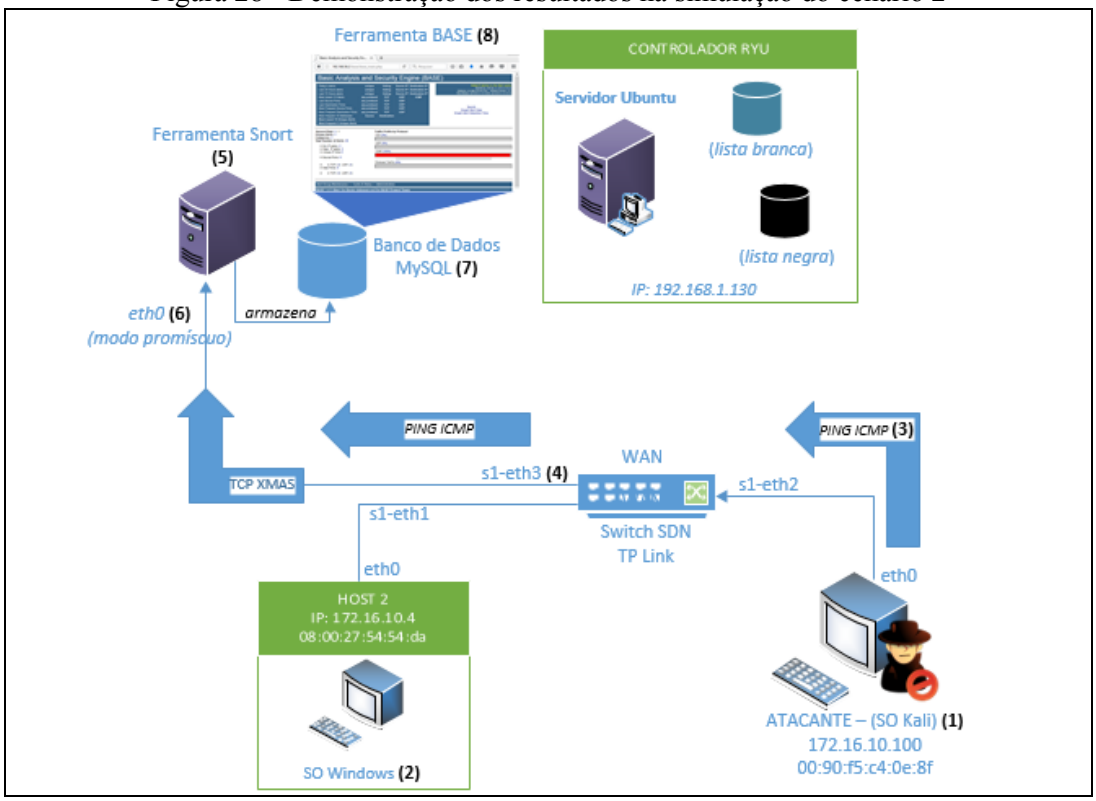
No caso de se ter um dispositivo com a requisição de criação de entrada de fluxo negada, o usuário necessariamente deverá procurar o administrador de rede, sendo este usuário o único que possuirá a permissão para editar o arquivo *whitelist*.txt e incluir este MAC na lista branca para autorizar o acesso. Com essa demonstração é possível concluir que o recurso de *whitelit* foi validado, sendo um importante instrumento para garantir que o administrador de rede disponibilize acesso à rede *OpenFlow/SDN* somente a dispositivos autorizados e em conformidade com as políticas de segurança implementadas.

Percebe-se que, por meio do desenvolvimento no plano de controle, é possível o reuso de códigos para finalidades diferentes, sendo importante ressaltar que, no contexto deste trabalho, foi empregado como processo de admissão apenas a verificação de endereços MAC, mas seria possível usar diferentes parâmetros de rede para implementar diferentes abordagens e garantir a eficácia da aplicabilidade da política de segurança da instituição.

5.3 CENÁRIO 2: VALIDAÇÃO DOS MECANISMOS DE DETECÇÃO

A realização dos testes de validação dos mecanismos de detecção, podem ser acompanhados por meio da Figura 28 e consistem na execução da ferramenta Snort e uso do comando ICMP ping[25] na máquina ATACANTE para geração de **tráfego Classe 1** direcionado a máquina 172.16.10.4.

Figura 28 - Demonstração dos resultados na simulação do cenário 2



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

[25] Uma das primeiras ações de um usuário malicioso consiste na execução de comandos de escaneamento e varredura de rede com o objetivo de lentar informações da topologia de rede. Por isso a estratégia de se empregar no Cenário 2 o uso do comando ICMP ping e no Cenário 3 o uso da ferramenta nmap.

todo o tráfego da rede, de forma promíscua, que está entrando pela interface *eth0* da própria máquina do Snort.

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

A opção -A *unixsock* irá 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. O parâmetro -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. 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,* permitindo identificar tráfego suspeito na rede como um comando ICMP com finalidade de identificar se um servidor está ativo na rede:

```
alert icmp any any -> any any (msg: "Classe_1 - ICMP test detected"; GID:1; sid:10000001; rev:001; classtype:icmp-event)
```

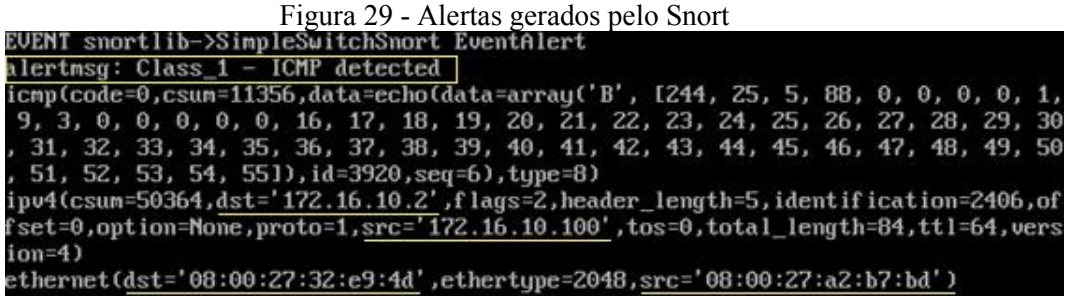
Com isso, é possível iniciar a simulação para validação da regra criada no Snort, por meio da execução do comando ICMP ping na máquina do ATACANTE, direcionado ao nó com o sistema operacional Windows na máquina com o IP 172.16.10.4, da seguinte forma:

 # ping 172.16.10.4

Como a interface *eth0* da máquina Snort recebe todo o tráfego que passa pela interface *eth3* do *Switch OpenFlow* (Modo de espelhamento), o pacote *ICMP ping* é identificado pelo Snort como tráfego suspeito, sendo armazenado em banco de dados MySQL e pode ser analisado posteriormente, por meio da ferramenta BASE, pelo administrador de rede para registro e análise. Nesse cenário, o tráfego suspeito de Classe 1, conforme Figura 29, é enviado ao controlador apenas para acompanhamento do administrador, na medida em que a estratégia desta dissertação definida na Tabela 2 e apresentada em detalhes no subcapítulo 5.3, o

único tratamento para tráfegos de Classe 1 na rede *OpenFlow/SDN* consiste no armazenamento em banco de dados para futuras análises.

Figura 29 - Alertas gerados pelo Snort



É possível identificar, na saída de comando do controlador Ryu, que foi enviado ao controlador pelo *pigrelay* a mensagem de alerta do Snort, identificando o tráfego como sendo do tipo "Class 1 – ICMP detected". Dessa forma comprovou-se que o tráfego detectado pelo Snort corresponde ao comando ICMP ping executado a partir da máquina do ATACANTE, no endereço IP 172.16.10.100 e endereço físico 08:00:27:54:54:da, com destino à máquina Windows no IP 172.16.10.4 e endereço físico 00:90:f5:c4:0e:8f, tendo por objetivo identificar se a máquina alvo de ataques está ativa.

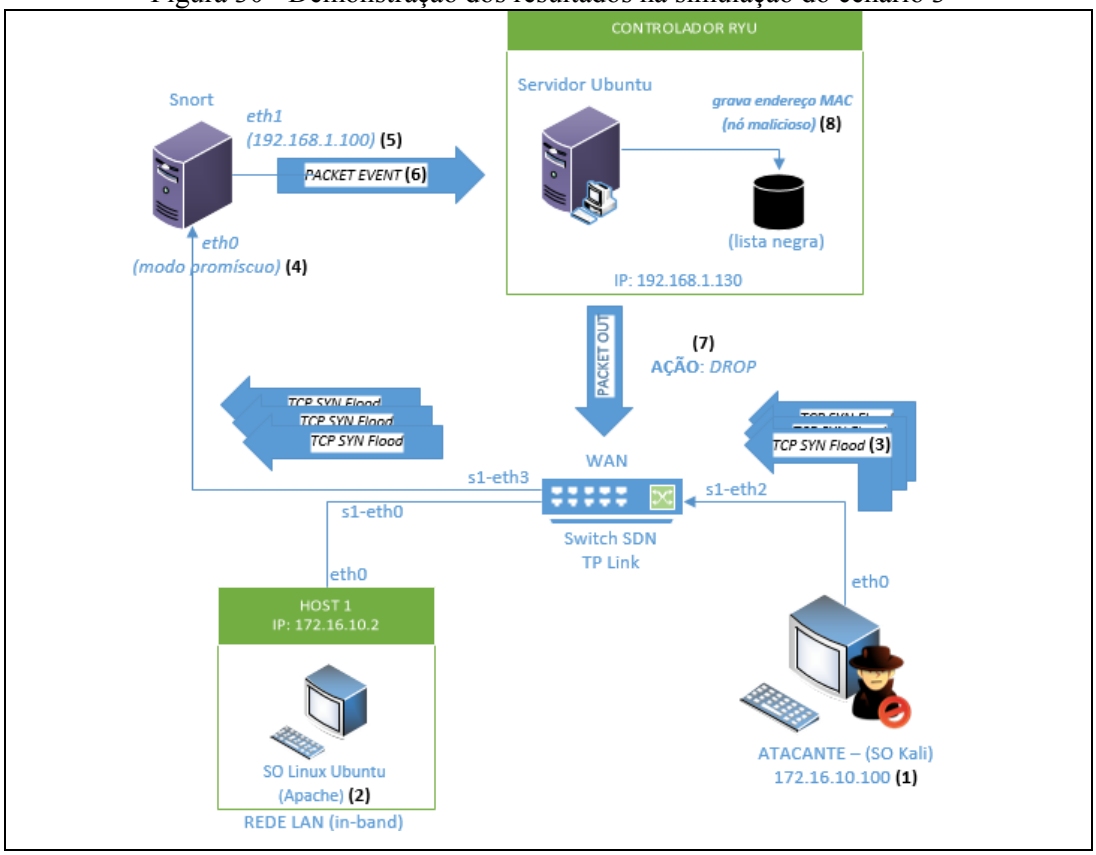
Com essa demonstração é possível concluir que o mecanismo de detecção foi validado e cumpre os objetivos propostos no trabalho de identificar tipo de ataque do tipo Classe 1, com armazenamento do tráfego de dados considerado suspeito em uma base de dados, permitindo ao administrador a realização de análise e tratamento de intrusão por meio da ferramenta BASE.

5.4 CENÁRIO 3: VALIDAÇÃO DOS MECANISMOS DE REAÇÃO COM AÇÃO DE *DROP* E INCLUSÃO DO MAC SUSPEITO EM *BLACKLIST*

A realização dos testes de validação dos mecanismos de reação podem ser acompanhados por meio da Figura 30 e consistem na execução da ferramenta Snort e uso da ferramenta hping3[26] 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.

[26] É um software livre gerador e analisador de pacotes para o protocolo TCP/IP com foco em segurança para testar e auditar redes e aplicações de segurança.

Figura 30 - Demonstração dos resultados na simulação do cenário 3



Ao final da execução do comando abaixo a ferramenta Snort inicia no modo de detecção de intrusão de rede, idêntico a execução do Snort apresentado no cenário 2.

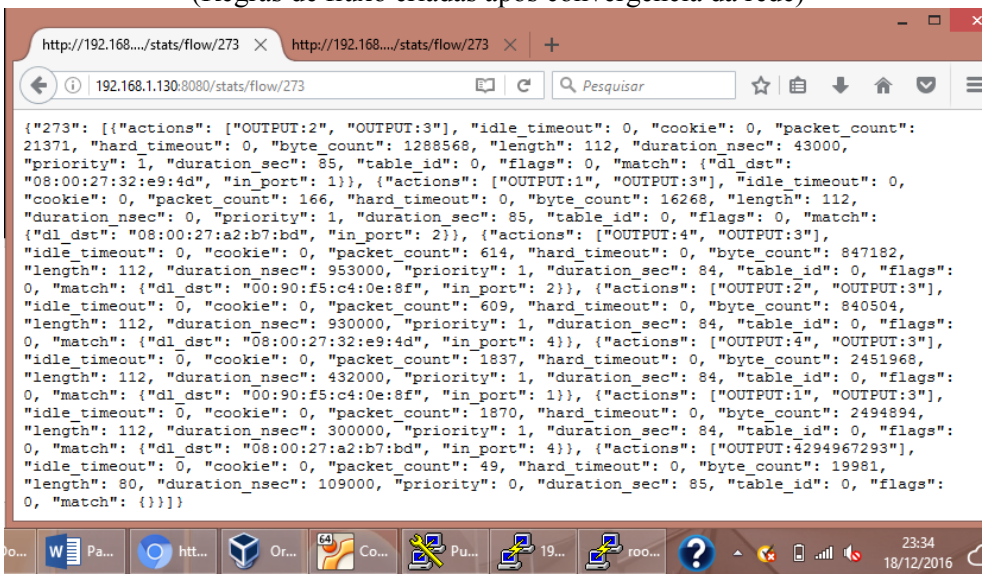
```
#sudo snort –i eth0 –A unixsock –l /var/log/snort –c /etc/snort/snort.conf –g snort –u
```

Para validação do funcionamento do Snort criou-se uma regra local no arquivo de configuração local.rules, permitindo identificar tráfego suspeito na rede *OpenFlow* do tipo DoS:

```
alert tcp any any -> [172.16.10.100] 80 (msg: "Classe_3 - DDoS Detected"; threshold: type both, track by_src, count 100, seconds 1; sid:1004; rev:1;)
```

O cenário 3 prevê que todos os clientes de rede apresentados na Figura 31 possuem uma entrada de fluxo na tabela do *Switch OpenFlow/SDN* que possibilite a comunicação entre os nós da rede. Nesse sentido, para possibilitar validar o processo de reação com *DROP* é apresentado abaixo o estado inicial da tabela de fluxos do *Switch* com todas as entradas atuais, obtidas por meio de comando REST API utilizadas via *browser*:

Figura 31 - Resultado da saída do comando REST API em *browser*
(Regras de fluxo criadas após convergência da rede)



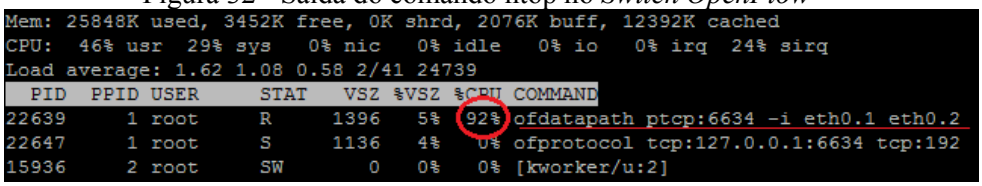
Com isso, é possível iniciar a simulação de ataque para validação da regra criada, por meio da execução da ferramenta t50 na máquina do ATACANTE, direcionado ao *host* 1 que executa o serviço de aplicação Apache na máquina Linux com o IP 172.16.10.2. A ferramenta t50 é um injetor de pacotes utilizado na execução de testes de stress em uma rede, permitindo simular ataque DoS e DDoS, permitindo disparar simultaneamente pacotes de diferentes protocolos. Para fins de simulação, usou-se o seguinte comando na máquina do atacante:

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

Nesse 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 ao ataque, é possível identificar, por meio da ferramenta de monitoramento e gerenciamento de rede ntop (Figura 32), em execução no *Switch OpenFlow*, que no pico do ataque o consumo de CPU no plano de dados (*datapath*) chegou a ordem de 92%.

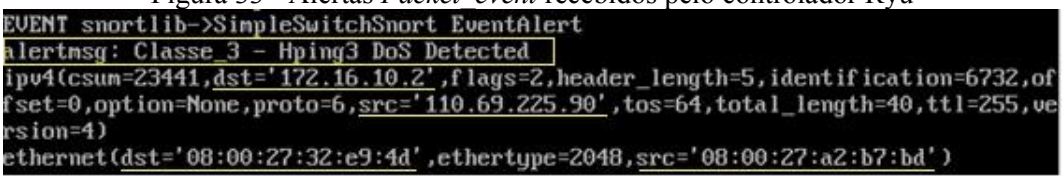
Figura 32 - Saída do comando ntop no *Switch OpenFlow*



Seguindo essa tendência e não sendo adotada nenhuma medida de mitigação para conter o ataque do tipo DoS, a rede estaria comprometida e seria impossível o *Switch OpenFlow* tratar novos tipos de tráfegos na rede. Nesse sentido, foram implementados mecanismos de reação aplicados ao ambiente proposto para demonstrar a efetividade da estratégia de solução adotada, visando fazer frente a uma ameaça de segurança que poderia comprometer toda a rede *OpenFlow/SDN*.

Assim que a máquina do Snort começa a identificar o tráfego malicioso na rede, após execução da ferramenta t50, 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 Figura 33 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 33 - Alertas *Packet_event* recebidos pelo controlador Ryu



Na Figura 33 é possível comprovar na saída de comando do controlador Ryu, em modo *verbose,* que o controlador Ryu recebe uma mensagem de alerta, por meio da biblioteca *snortlib* do Linux, identificando um ataque do tipo de Classe 3 "Classe_3 – DoS Detected", 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, destinados 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.

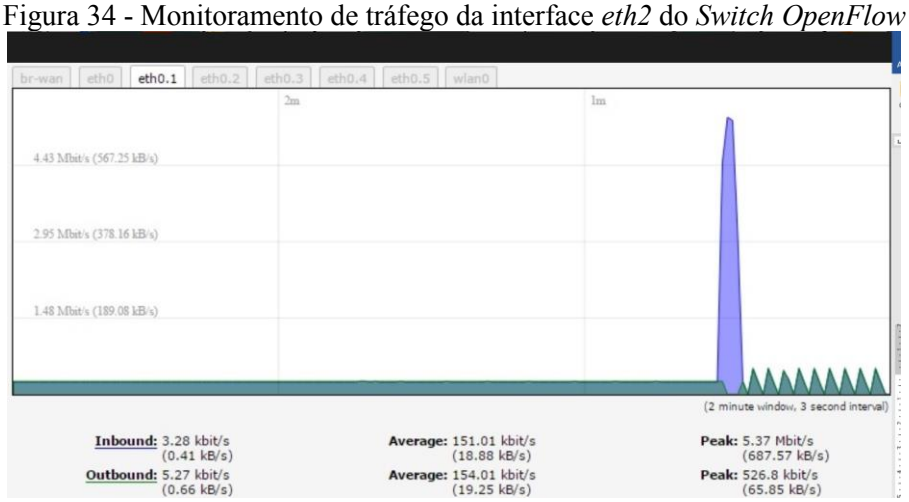
No caso deste trabalho o tratamento dado a tráfegos de ataque do tipo Classe 3 consiste em enviar uma mensagem *OpenFlow* com ação *DROP* ao *Switch* para remover todas as entradas de fluxo no *Switch* que tenha por origem o host responsável pela ação maliciosa, assim como remover todas as entradas de fluxo que tenham como destino o host responsável pela ação maliciosa.

Outra ação programada na aplicação *mitigacao.py* consiste em inserir na *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 tenha condições de 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 e verificação do arquivo de *blacklist*.

Dessa 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 à rede, já que a política da rede segue o padrão *deny-by-default*. A validação dos mecanismos de reação pode ser comprovada por meio os resultados obtidos e apresentados na Figura 34, que apresenta o monitoramento do tráfego de fluxos na rede na interface *eth2* do *Switch OpenFlow,* conectado ao nó do servidor Apache no momento do ataque DoS.

No momento da execução da ferramenta t50 o tráfego na interface *eth2* alcançou um volume de tráfego de entrada de 900Kbps e esse tráfego foi restabelecido após tomadas as medidas de mitigação apresentadas neste trabalho.

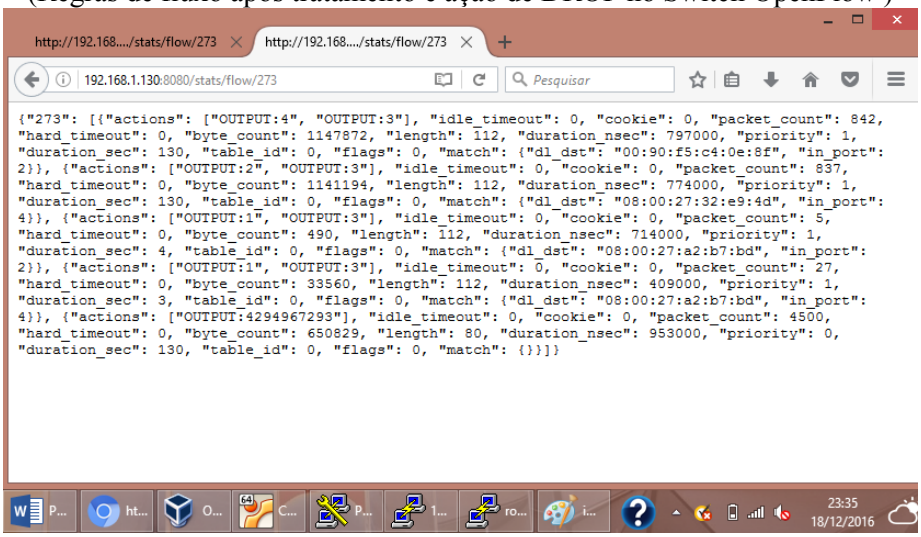
Figura 34 - Monitoramento de tráfego da interface *eth2* do *Switch OpenFlow*



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 2 a 3 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 está conectada a máquina Linux que sofreu o ataque DoS, demonstrado por meio da Figura 33.

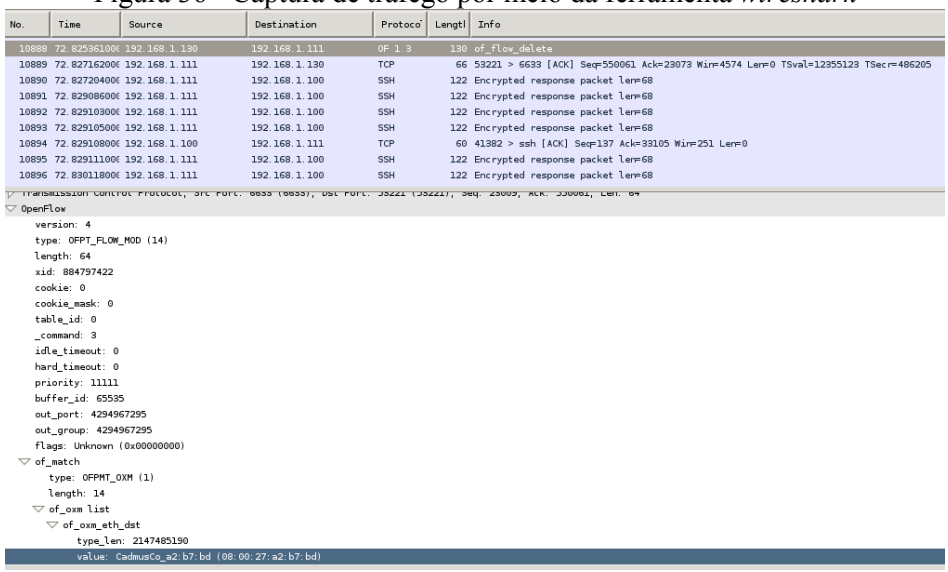
Uma forma de se comprovar a efetividade do mecanismo de reação com *DROP* é a comparação entre o estado da tabela de fluxos do *Switch OpenFlow/SDN* antes da simulação e depois da simulação por meio do comando t50. Na Figura 35 é possível verificar que todas as entradas de fluxos que antes da simulação possibilitavam a comunicação do *host* atacante com as outras máquinas foram removidas.

Figura 35 - Resultado da saída do comando REST API em browser
(Regras de fluxo após tratamento e ação de DROP no Switch OpenFlow)



Outra forma de comprovar que o controlador efetivamente enviou um comando de *DROP* ao *Switch OpenFlow/SDN* se dá por meio de análise de tráfego no próprio controlador (Figura 36), feita através da ferramenta *wireshark,* que mostra o envio, pelo controlador (IP 192.168.1.130), do comando ***of_flow_delete*** ao *Switch* (IP 192.168.1.111) para remover todas as entradas de fluxo que tenham o MAC 08:00:27:a2:b7:bd do dispositivo atacante.

Figura 36 - Captura de tráfego por meio da ferramenta *wireshark*

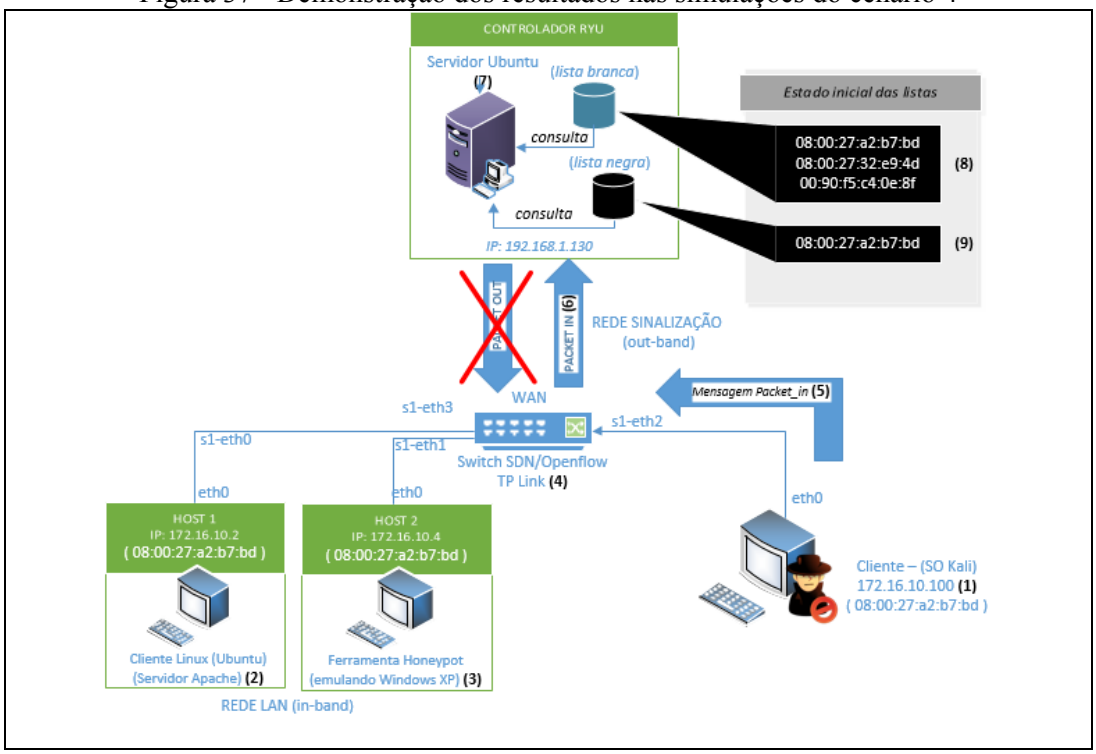


Com essa demonstração conclui-se que o mecanismo de reação foi validado, cumprindo os objetivos propostos no trabalho de identificar ataques do tipo Classe 3, com bloqueio do tráfego por meio da remoção de todas as entradas de fluxo no *Switch* que tenham como origem e destino o MAC considerado suspeito e inserção do MAC suspeito em *blacklist* para isolar o dispositivo da rede, evitando a criação de novas entradas de fluxos no *Switch OpenFlow*.

5.5 CENÁRIO 4: VALIDAÇÃO DO RECURSO DE *BLACKLIST*

Neste cenário, apresentado na Figura 37, a simulação inicia a partir do momento que o ATACANTE, rodando o OS Kali **(1)** no endereço MAC | 08:00:27:a2:b7:bd |, tenta acessar o site web em execução no *host* 1 **(2)** no endereço MAC 08:00:27:a2:b7:bd |, logo após ter realizado uma ação maliciosa demonstrada no cenário 2. Como não é encontrado uma entrada de fluxo no *Switch OpenFlow*, porque toda e qualquer entrada de fluxo relacionado ao MAC suspeito foi removida, a comunicação entre os *hosts* não é permitida, exigindo que o *Switch OpenFlow* **(4)** encaminhe ao Controlador Ryu uma solicitação de criação de entrada de fluxo por meio de uma mensagem *Packet_in* **(5)**. O Controlador **(6)** em execução no servidor com endereço IP | 192.168.1.130 |, ao tratar a mensagem *Packet_in*, realiza duas verificações. A primeira consiste em identificar se o MAC | 08:00:27:a2:b7:bd | é de um dispositivo autorizado e a segunda consiste em verificar se este mesmo MAC não é considerado suspeito.

Figura 37 - Demonstração dos resultados nas simulações do cenário 4



Em tempo de execução, quando um dispositivo requisita a criação de uma entrada de fluxo no controlador, um método no arquivo de código *mitigacao.py* foi programado para ler tanto o arquivo de *whitelist.txt* quanto o arquivo *blacklist.txt*. A inserção de novos endereços MAC no arquivo *blacklist.txt* é feita por meio do tratamento dado aos eventos de segurança de Classe 3 como demonstrado na Tabela 2, apresentada no subcapítulo 5.3 desta dissertação. Assim, se o dispositivo requisitar a criação de uma entrada de fluxo, por meio de uma mensagem *Packet_in*, por exemplo, a mensagem é tratada no controlador e o método *Packet_in_handler()* verifica não só se ele é autorizado, mas também se o endereço MAC do dispositivo é considerado suspeito. Para esta simulação, o estado final da *whitelist* e *blacklist* após as simulações realizadas nos cenários anteriores pode ser observada na Figura 38.

Figura 38 - Estado final dos arquivos *whitelist* e *blacklist* no cenário 4

```
root@ryu:/opt/ryu# cat whitelist.txt
08:00:27:a2:b7:bd
08:00:27:32:e9:4d
00:90:f5:c4:0e:8f
root@ryu:/opt/ryu# cat blacklist.txt
08:00:27:a2:b7:bd
root@ryu:/opt/ryu# _
```

Seguindo o processo de demonstração, a primeira verificação realizada é validada, na medida em que o MAC de origem da mensagem *Packet_in* é de um dispositivo autorizado em *whitelist* **(7)**. No entanto, uma vez que o dispositivo do ATACANTE realizou uma ação maliciosa, demonstrada no cenário 3, seu endereço MAC consta da *blacklist* **(8)** e a segunda verificação não é validada, impossibilitando que o controlador envie a mensagem *Packet_out* ao *Switch* para criar a entrada de fluxo solicitada. No exemplo em questão, o endereço MAC de origem da mensagem *Packet_in* agora é considerado suspeito, por estar em *blacklist*, como apresentado na Figura 39, impossibilitando a criação de novas entradas de fluxo no *Switch* para esse endereço.

Figura 39 - Bloqueio de um endereço MAC considerado suspeito

Com esta demonstração, conclui-se que o recurso de *blacklist* foi validado, sendo um importante instrumento para garantir a proteção do ambiente de rede *OpenFlow/SDN*, na medida em que garante que o dispositivo fique totalmente isolado da rede quando for detectada a geração de tráfego contrário à política de segurança definida. O dispositivo só conseguirá permissão novamente para acessar a rede quando o administrador retirar o endereço MAC da *blacklist*, após tratar o incidente e garantir que o dispositivo esteja em conformidade com as políticas de segurança implementadas.

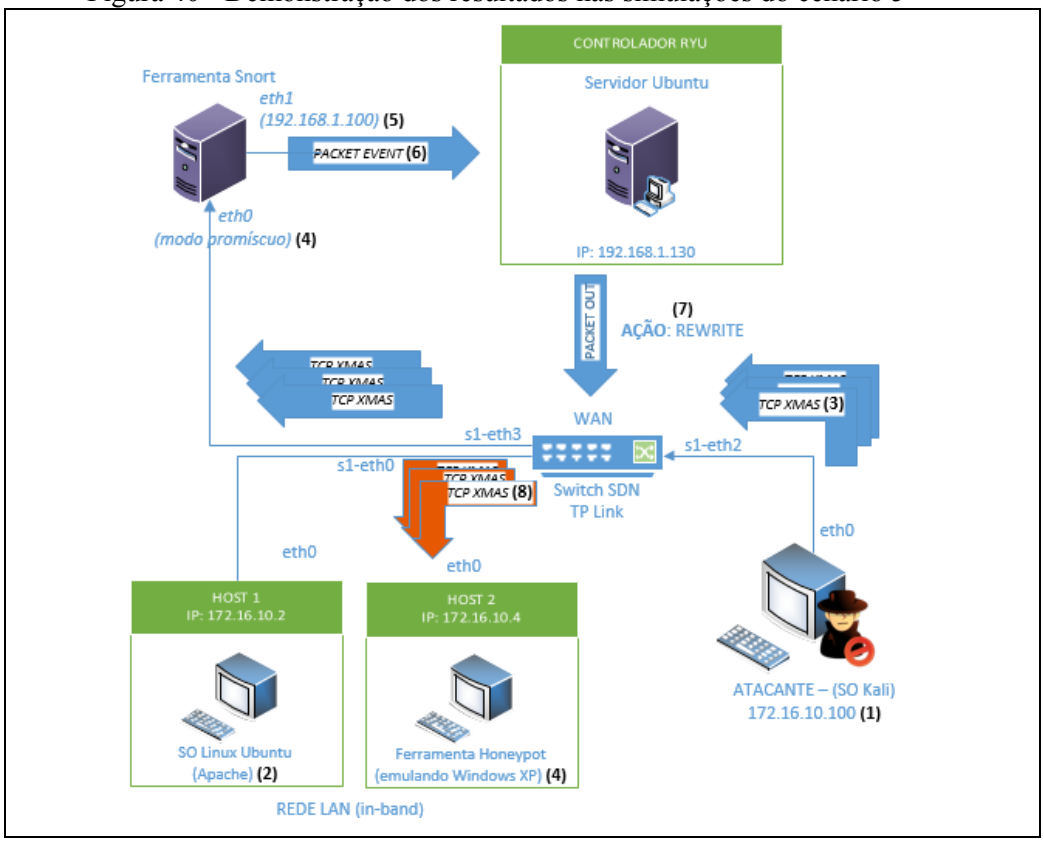
5.6 CENÁRIO 5: VALIDAÇÃO DOS MECANISMOS DE REAÇÃO COM AÇÃO DE REESCREVER (*REWRITE*) E ENVIO (REENCAMINHAR) DE TRÁFEGO SUSPEITO PARA UMA *HONEYPOT*

A realização dos testes de validação dos mecanismos de reação podem ser acompanhados na Figura 40 e consistem na execução da ferramenta Snort e uso da ferramenta nmap[27] na máquina do ATACANTE para geração de tráfego malicioso, com o objetivo de identificar: serviços (nome da aplicação e versão); o

[27] O Nmap ("Network Mapper") é uma ferramenta de código aberto para exploração de rede e auditoria de segurança. Ela foi desenhada para escanear rapidamente redes amplas, embora também funcione muito bem contra hosts individuais.

sistema operacional; e tipos de filtros ou *firewall* em uso no servidor de aplicação de páginas, que está executando o serviço Apache na máquina 172.16.10.2.

Figura 40 - Demonstração dos resultados nas simulações do cenário 5



Este cenário busca comprometer a fase de enumeração[28] de um ataque de rede, pois praticamente torna impossível ao ATACANTE definir o perfil da máquina-alvo. Para geração de tráfego foi usada a ferramenta nmap da máquina do atacante **(1)** para simular um trafego do tipo probe na rede ao Host 1 **(2)**:

 # nmap -sX -O 172.16.10.2

Nesse comando, os seguintes parâmetros são definidos: "-sX" também conhecido como *scan* Xmas porque ele marca as flags FIN, PSH e URG no pacote; e o "-O" habilita a detecção do sistema operacional da máquina-alvo. No caso de um retorno RST, a porta está fechada e caso não seja recebida resposta a porta poderá estar aberta.

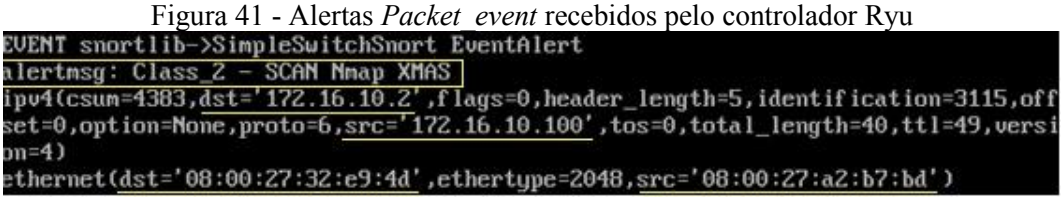
Ao executar o comando nmap o *Switch OpenFlow* na interface *eth3* espelha todo o tráfego no *datapath* para o Snort na sua interface *eth0* que está habilitada

em modo promíscuo. O teste conduzido aqui consiste em identificar o tipo de ataque classificado na proposta como sendo de Classe 2. Para detectar o tráfego da simulação, o Snort foi configurado por regras no arquivo *rules.local*, conforme apresentado abaixo, que possibilita que ao ser identificado seja enviado para o controlador por meio de mensagens *Packet_event* que é tratado por meio do código *mitigacao.py*. Neste código, o tráfego de Classe 2 é tratado por meio de mecanismos de reação implementados por meio da chamada ao método *modify_tables_flow.py*.

```
alert tcp any any -> any any (msg: "Class_2 - SCAN Nmap XMAS"; flow:stateless; flags:FPU,12; GID:1; sid:10000007; rev:001; classtype:attempted-recon;)
```

Assim que a máquina do Snort começa a identificar o tráfego malicioso na rede, a aplicação *pigrelay.py* inicia o envio de mensagens *Packet_event* pela interface *eth1* do Snort ao controlador por meio de uma *daemon* no servidor Linux. Como apresentado na Figura 41 o IDS Snort se mostrou eficaz na detecção da simulação de ataques gerados pela ferramenta nmap, detectando a intrusão e gerando alertas para posterior análise.

Figura 41 - Alertas *Packet_event* recebidos pelo controlador Ryu



Na Figura 40 é possível comprovar na saída de comando do controlador Ryu, em modo *verbose,* que ele recebe uma mensagem de alerta por meio da biblioteca *snortlib* do Linux, indicando um ataque do tipo **Classe 2** "Classe_2 – Scan Nmap XMAS", tendo como origem a ferramenta *nmap,* em execução na máquina do ATACANTE, no endereço físico | 08:00:27:a2:b7:bd | 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

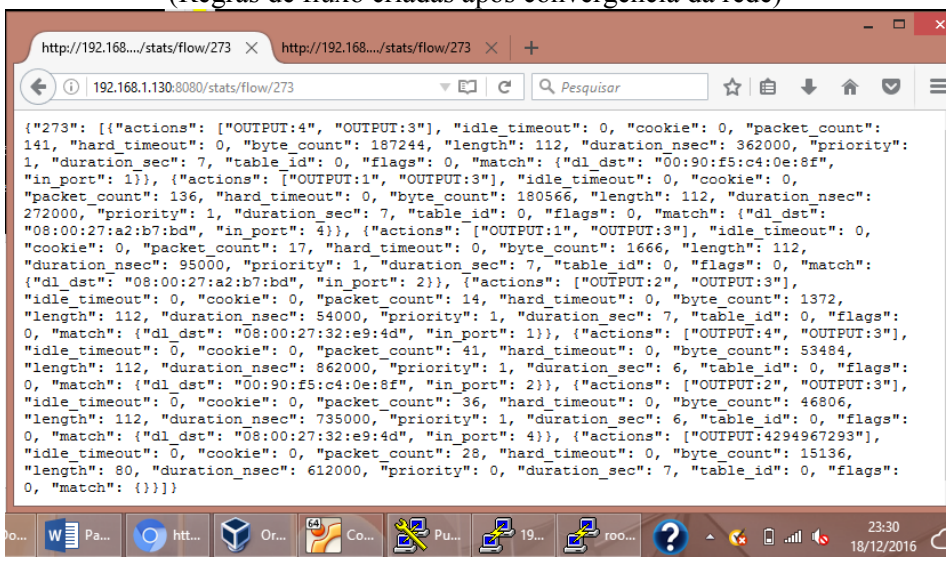
[28] A fase de enumeração é executada por um ATACANTE com o objetivo de identificar se o host está ativo, o sistema operacional e serviços instalados na máquina-alvo a fim de definir o nível de vulnerabilidade do dispositivo e a estratégia de ataque a ser empregada.

recebido. O controlador Ryu, ao receber a mensagem *Packet_event* do Snort, inicia o processo de tratamento do evento de segurança por meio do método *checkmessenger()* no código *mitigacao.py*. Esse método está programado para tratar ataques do tipo **Classe 2** e consiste em enviar uma mensagem para o *Switch OpenFlow* com a ação de reescrever (*REWRITE*) todas as entradas de fluxos que tenham o endereço MAC que originou o ataque para a interface *eth4* do *Switch*, que está conectado a uma máquina devidamente preparada para funcionar como uma *honeypot*.

Nesse caso o método sabe qual foi o endereço MAC do disposito que originou o ataque por meio da mensagem *Packet_event,* que contém todas as informações de ataque. As informações da mensagem *Packet_event* são carregadas como parâmetros no método *modify_tables_flow.py*. Desaa forma é possível redirecionar todo o tráfego para a interface *eth4* do *Switch* e efetivamente implementar uma armadilha para o atacante com o propósito de obter informações sobre o ataque em andamento e, ainda, preservar a máquina-alvo.

O resultado da ação de reescrita pode ser observado pela consulta em REST API, via cliente web, logo depois do tratamento dado pelo código *mitigacao.py*, que identifica as entradas de fluxo no *Switch OpenFlow* após convergência e conexão entre todos os nós da rede (Figura 42).

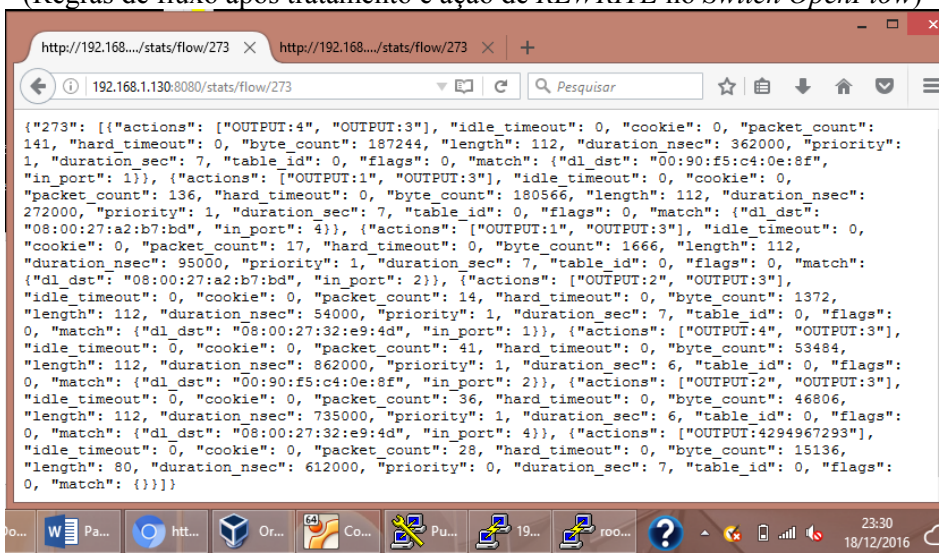
Figura 42 - Resultado da saída do comando REST API em *browser*
(Regras de fluxo criadas após convergência da rede)



A partir da comparação entre as entradas de fluxos antes e depois da execução do comando nmap, pode-se observar que as entradas de fluxo referentes

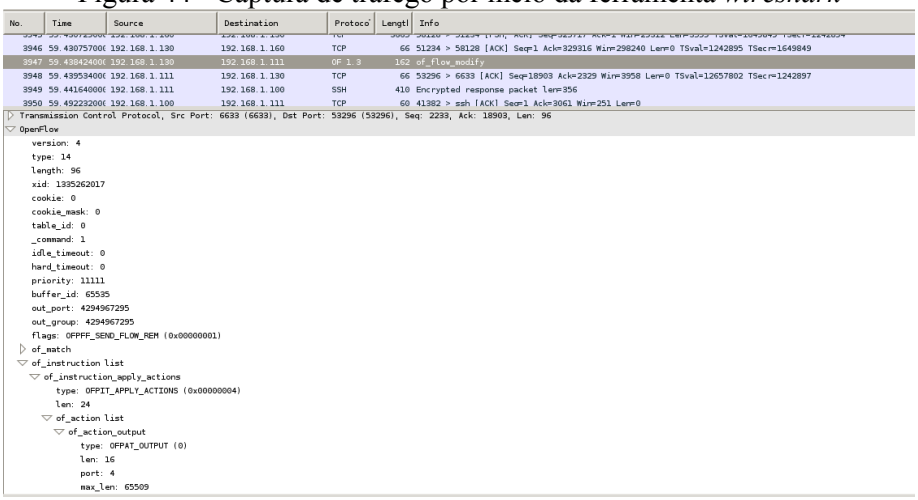
ao endereço MAC do ATACANTE que tinham como porta de saída a interface *eth1,* ligada ao servidor web, agora tem como porta de saída a interface *eth4,* que está interligada a *honeypot*, conforme apresentado na Figura 43.

Figura 43 - Resultado da saída do comando REST API em *browser*
(Regras de fluxo após tratamento e ação de *REWRITE* no *Switch OpenFlow*)



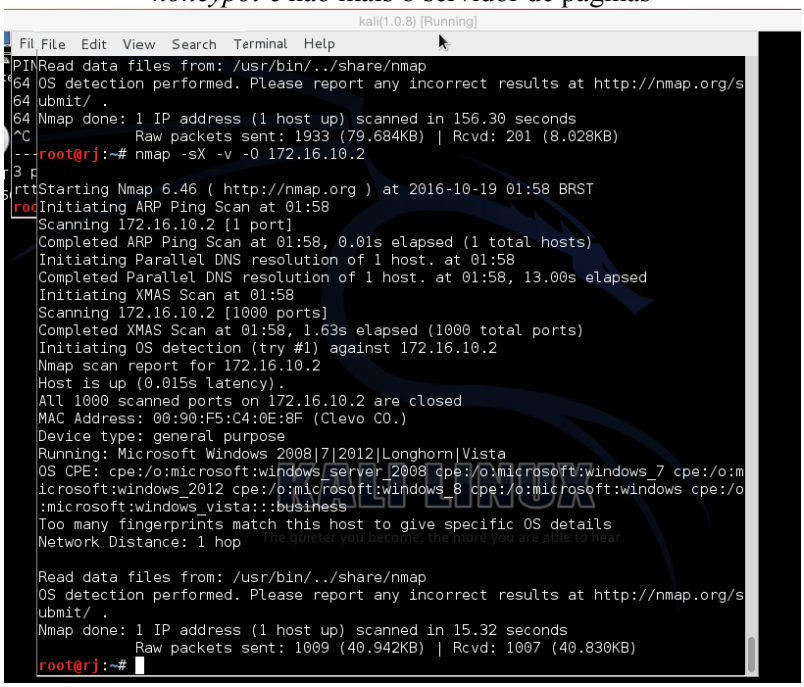
E uma outra forma de se comprovar que o controlador efetivamente enviou um comando de *REWRITE* ao *Switch OpenFlow/SDN* pode ser identificado por meio da análise de tráfego no próprio controlador feita por meio da ferramenta *wireshark* (Figura 44) que comprova que foi encaminhado do controlador (IP 192.168.1.130) o comando *of_flow_modify* ao *Switch* (IP 192.168.1.111) para reescrever todas as entradas de fluxo que tenham o MAC 08:00:27:a2:b7:bd do dispositivo atacante e indicando como porta de saída a *interface 4* (*of_action_output*) do Switch que está interligado a máquina da *honeypot*.

Figura 44 - Captura de tráfego por meio da ferramenta *wireshark*



Esta ação atualmente leva em média de 1 a 2 segundos, sendo suficiente para retornar na saída do comando na máquina do ATACANTE o resultado da enumeração da máquina *Windows XP* (Figura 45) que está sendo emulada no servidor *honeyd* na interface *eth4* do *Switch* propositalmente para enganar o usuário malicioso e dar a ele informações falsas e preservando a máquina-alvo.

Figura 45 - Saída do comando nmap no ATACANTE obtendo resultados da máquina da *honeypot* e não mais o servidor de páginas



Com esta demonstração é possível concluir que o mecanismo de reação foi validado e cumpre os objetivos propostos no trabalho de identificar tipo de ataque Classe 2, com reencaminhamento do tráfego por meio da reescrita de todas as entradas de fluxo no *Switch* que tenham como origem o MAC considerado suspeito, direcionamento o tráfego para uma *honeypot* e enganando o ATACANTE.

6 CONCLUSÃO

O desafio da implantação de um ambiente para detecção e prevenção flexível de ataques em redes *OpenFlow/SDN* é o foco da proposta desta dissertação, que foi avaliada de forma efetiva, por meio de um ambiente de rede real *OpenFlow/SDN* para experimentação na Universidade de Salvador (UNIFACS).

O esforço inicial se resumiu ao entendimento do princípio de funcionamento das redes SDN, por meio do simulador MiniNet, à definição de qual controlador de rede *OpenFlow* deveria ser adotada no contexto do trabalho, assim como a identificar que tipo de ambiente de rede seria empregado para permitir a validação dos mecanismos de proteção propostos baseados no protocolo *OpenFlow*.

Em um segundo momento foi possível identificar a necessidade de agregar novas funcionalidades ao plano de controle visando implementar os mecanismos de proteção definidos nos objetivos da dissertação. Como evolução natural da prática de pesquisa e a maturidade no desenvolvimento do trabalho definiu-se o uso de um ambiente real para validação da proposta, por meio da construção de um ambiente de rede real *OpenFlow/SDN* para experimentação, empregando dispositivos reais e tráfego real, em substituição à prática de simulação. Somado a isso, o entendimento do arcabouço do controlador adotado na dissertação permitiu que o desenvolvimento da proposta seguisse uma abordagem inovadora.

Neste sentido, o trabalho de dissertação apresentou uma solução para extensão das funcionalidades do controlador Ryu, incluindo novas aplicações que permitem a configuração de parâmetros de controle de tráfego nos dispositivos de rede, a fim de prover mecanismos de detecção e reação a ameaças de segurança. A abordagem de segurança utilizou a estratégia centralizada e a programabilidade do controlador como forma de tratar a segurança de rede *OpenFlow/SDN,* empregando um IDS de rede baseado em assinaturas para complementar os métodos convencionais de segurança e disponibilizar recursos ao administrador de rede que possibilite o monitoramento, tratamento e análise dos eventos de segurança. A pesquisa demonstrou que é possível se construir infraestruturas de redes para experimentação, baseadas em redes *OpenFlow,* de maneira rápida e simples.

A abordagem explora uma das principais características do controlador de rede *OpenFlow* de ser extensível e programável e permitiu a implementação flexível dos mecanismos de monitoramento e tratamento de eventos de segurança em tempo real, por meio da categorização dos tipos de ataque, associados a recursos de *whitelist* e *blacklist*.

Sua abordagem de segurança inova principalmente na detecção de intrusão por meio da utilização de um IDS que foi customizado e por meio da programabilidade do controlador de rede, que possibilitou que a estratégia de tratamento flexível dos eventos de segurança adotada na dissertação fosse feita de forma inovadora e não convencional. A inovação consistiu em demonstrar que é possível, por meio da parametrização da ferramenta de detecção de intrusão, permitir ao administrador de rede definir, de acordo com as necessidades da política de segurança aplicada à sua rede, que tipo tratamento seria mais adequado para diferentes classes de ataque.

Esta técnica flexível, agora associada aos mecanismos de controle de rede do ambiente *OpenFlow/SDN*, permite não apenas monitorar e detectar potenciais ataques de rede, como também implementar mecanismos efetivos de "reação" às ameaças de forma controlada e centralizada no núcleo da rede por meio do controlador.

Com efeito, os testes de validação baseado em cenários apresentados e as técnicas empregadas para apresentação dos resultados demonstraram a efetividade dos mecanismos de monitoramento, detecção e reação às ameaças de segurança propostos nesta dissertação.

O grande diferencial do trabalho reside no fato de que os mecanismos de reação foram implementados no próprio controlador, a decisão é centralizada, não dependendo de equipamentos externos e também por não estar limitado apenas a ação de remover entradas de fluxos, mas também ser capaz de reescrever entradas de fluxos, o que é até o presente momento impossível de se realizar nos comutadores convencionais, demonstrando algumas das características de quebra de paradigma das redes *OpenFlow/SDN*.

As contribuições da dissertação são evidenciadas por meio da própria construção do ambiente de rede real *OpenFlow/SDN*, que empregou um dispositivo de prateleira comum de baixo custo, com suporte *OpenFlow*,

linguagem de programação Python, assim como soluções e ferramentas baseadas em software livre, e que pode ser recriado por outros grupos de pesquisas para testes e validação de novas propostas.

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 IDPS, 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.

Com isso, foi possível disponibilizar uma solução de IDPS para identificar tráfego malicioso em uma rede L2, com recursos de admissão por meio de *blacklist* e *whitelist*, antes mesmo do tráfego sair para internet, por exemplo, não ficando limitado ao que ocorre com as redes convencionais de ter que contar com recursos de detecção e prevenção usando sensores que necessariamente devem estar instalados na borda da rede.

Com relação à reescrita na tabela de fluxos dos Switches *OpenFlow/SDN*, destaca-se que foi possível demonstrar o grande potencial deste recurso ao permitir enganar o atacante e desviar seu ataque para uma *honeypot* em um tempo imperceptível, possibilitando ao administrador de rede obter toda a assinatura de ataque empregada sem despertar a atenção.

Trabalhos futuros devem incluir o uso de outros IDS em um ambiente com mais de um *Switch OpenFlow* para fornecer balanceamento de carga entre alternativas potenciais, assim como possibilitar identificar como o controlador irá sincronizar e tratar as diferentes mensagens enviadas por diferentes equipamentos de detecção de intrusão para tratamento no controlador. Diferentes cenários de validação podem ser elaborados com o objetivo de avaliar outros tipos de ataques, como por exemplo, ataques DoS e DDoS.

Também podem ser estudados fatores de escalabilidade da solução em uma rede com uma carga de tráfego similar a uma rede em produção, assim como identificar a precisão da detecção de intrusão por meio do levantamento do número de falsos positivos (identificação de um alerta para uma intrusão que na

realidade é inexistente) e falsos negativos (quando efetivamente ocorre uma intrusão, mas o alerta não foi gerado).

De maneira geral, a potencialidade e diversidade da solução no contexto *OpenFlow/SDN* é programável e, além disso, pode ser dinamicamente ajustada segundo critérios e políticas de gerência da rede ou sistema. 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 *OpenFlow/SDN*.

REFERÊNCIAS

ABNT. **NBR ISO/IEC 27002:2005**: Tecnologia da informação – Técnicas de segurança – Código de prática para a gestão da segurança da informação. Rio de Janeiro, 2005. 120 p

AKBAR, Mehdi S.; JUNAID, Khalid; SAYED, Ali K. Revisiting traffic anomaly detection using software defined networking. In: RECENT advances in intrusion detection. [S.l.]: Springer, 2011. p. 161 e 80.

ANWER M. Bilal, et al. Switchblade: a platform for rapid deployment of network protocols on programmable hardware. **ACM SIGCOMM Comput Commun Rev.,** v.40, n.4, 2010.

AZEVEDO, R. P. **Detecção de ataques de negação de serviço em redes decomputadores através da transformada Wavelet 2D**. 2012. Dissertação (Mestrado). UFSM, Centro de Tecnologia, Programa de Pós-Graduação em Informática, RS, 2012.

BACE, R; MELL, P. **Intrusion detection systems**. [S.l.]: National Institute of Standards and Technology (NIST), 2001.

BAKER, Andrew. **Snort IDS and IPS Toolkit. Jay Beale's Open Source Security**. Rockland: Synress, 2007.

BALLARD, J. R.; RAE, I.; AKELLA, A. Extensible and scalable network monitoring using opensafe. In: INM/WREN, 2010. **Proc….** 2010.

BARBOSA A. **Sistemas de detecção de intrusão**. Disponível em: <http://www.lockabit.coppe.ufrj.br/downloads/academicos/IDS.pdf>. Acesso em: 12 nov. 2013.

BENTON, Kevin.; CAMP, L Jean.; SMALL, Chris. Openflow vulnerability assessment. In: ACM ACM SIGCOMM WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING, 2., 2013. **Proceedings…** 2013. p. 151 – 152.

BOBBA, Rakesh et al. **Redes definidas por software (SDN).** Disponível em: <osetoreletrico.com.br/web/a-revista/edicoes/1786-redes-definidas-por-softwa re-sdn.html>. Acesso em: 12 out. 2015.

CAMPOS, André L. N. **Sistema de segurança da informação:** controlando os riscos. Florianópolis: Visual Books, 2007.

CASADO, M.; FOSTER, N.; GUHA, A. Abstractions for software-defined networks. **Communications of the ACM**, ACM, v. 57, n. 10, p. 86–95, 2014.

CERT - COMPUTER EMERGENCY RESPONSE TEAM. **Denial of service attacks.** Disponível em: <http://www.cert.org/tech_tips/denial_of_service.html>. Acesso em: 19 set. 2015.

CISCO. **Relatório anual de cibersegurança** [Online]. Disponível em: <http:// b2me. cisco.com/pt-br-annual-cybersecurity-report-2017>. Acesso em: 10 fev. 2016.

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: a survey. **ACM Computing Surveys**, v.41, n.3, p.1–58, 2009

CONSORTIUM, O. **Pantou**: OpenFlow 1.0 for OpenWRT, 2012. Disponível em: <http://z6.co.uk/openflow>. Acesso em: 23 jul. 2015.

CONTERATO, Marcelo et al. **Avaliação do suporte à simulação de redes OpenFlow no NS-3**. [S.l.]: [s.n.], 2013.

COSTA, L. R. OpenFlow e o paradigma de redes definidas por software. 2013. Trabalho de Conclusão de Curso (Licenciatura em Computação)– Universidade de Brasília - UNB, Brasília, 2013. 143 f.

COURSERA. **Course on SDN**. [Online]. Disponível em: <https://www. coursera. org/ course/ sdn>. Acesso em: 12 fev. 2016.

CPQD. **OpenFlow 1.3 Software Switch**., 2012. Disponível em: <https://github.com/CPqD/ ofsoftswitch13>. Acesso em: 23 jul. 2015.

DEMIRAY, Sadettin. Improving misuse detection with neural networks. 2005. Dissertação (Mestrado)- Izmir Institute of Technology, Izmir – Turquia, 2005.

DESMEDT, Y. G. Threshold cryptography". **European Transactions on Telecommunications,** v. 5.4, 1994.

DONNER, M. L.; OLIVEIRA, L. R. Análise de Satisfação com a Segurança no Uso de Internet Banking em Relação aos Atuais Recursos Disponíveis no Canal Eletrônico. In: ENCONTRO DA ASSOCIAÇÃO NACIONAL DE PÓS-GRADUAÇÃO E PESQUISA EM ADMINISTRAÇÃO, 2008, Rio de Janeiro. **Anais…** 2008.

EVANGELISTA, S. V. B. **Sistemas de detecção de intrusos e sistemas de prevenção de intrusos**: princípios e aplicações de entropia. [S.l.]: [s.n.], 2008.

DÜRR, Frank. **Towards cloud-assisted software-defined networking**. Stuttgart: University of Stuttgart/Institute of Parallel \& Distributed Systems, 2012.

FARIAS, Fernando N. N. et al. Pesquisa experimental para a internet do futuro: uma proposta utilizando virtualização e o framework openflow. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS, 29., 2011. **Anais…** 2011.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to SDN: an intellectual history of programmable networks. In: ACM SIGCOMM Computer. **Communication Review**, ACM New York, NY, USA, v. 44, p. 87–98, 2014.

FERNANDES, E. L.; ROTHENBERG, C. E.. OpenFlow 1.3 software switch. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES (SBRC), 2014. **Anais…** 2014. p.1021–1028.

FLORES, Alejandro. **Implementando o Barnyard com SNORT**. Recife, 2004. Disponível em: <http://www.triforsec.com.br/novo/doc_copiar.php?id=5>. Acesso em: 5 mar. 2016.

GEORGIEV, M. The most dangerous code in the world: validating SSL certi_cates in non-browser software". In: ACM CCS, 2012. **Anais…** 2012.

GOMES, Verônica S. et al. Isolamento de recursos de rede em ambientes virtuais baseados em Openflow/SDN. In: WORKSHOP DE GERÊNCIA E OPERAÇÃO DE REDES E SERVIÇOS, 18., 2013. **Anais…** 2013.

GUDE, N. et al. NOX: towards na operating system for networks. In: SIGCOMM COMPUTER COMMUNICATION REVIEW, 38., 2008. **Anais…** 2008.

HOLZ, R. X.509 Forensics: Detecting and Localising the SSL/TLS Men-in-the-Middle". In: COMPUTER SECURITY. LNCS. 2012. **Proced.…** 2012.

HU, Fei.; HAO, Qi.; BAO, Ke. A survey on software-defined network and openflow: from concept to implementation. **IEEE Communications Surveys & Tutorials,** IEEE, v. 16, n. 4, p. 2181 – 2206, 2014.

HUGHES, J. **Network simulation introduction. Website OpenXtra**. 2009. Disponível em: <http://www.openxtra.co.uk/articles/network-simulation>. Acesso em: 12 mar. 2016.

INDIGO. **Indigo project Website**. 2013. Disponível em: <http://www. OpenFlowhub.org/display/Indigo/Indigo+-Open+Source+ OpenFlow+ Switches+ First+Generation/>. Acesso em: 12 maio 2016.

ITU-T Y.3300. **Framework of software-defined networking**. Disponível em: <http://www.itu.int>. Acesso em: 14 jul. 2016.

KIZZA, J. M. **Guide to computer network security.** New York, NY: Springer, 2005.

KREUTZ, D et al. Software-defined networking: a comprehensive survey. **Proceedings of the IEEE**, v.103, n.1, p.14–76, 2014.

KRUEGEL, C.; VIGNA, G. Anomaly detection of web-based attacks. In: ACM CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY, 10., New York, NY. **Proceedings…** 2003. p. 251-261.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In: HOTNETS-IX OF THE ACM SIGCOMM WORKSHOP ON HOT TOPICS IN NETWORKS ARTICLE, 19., 2010. **Proceedings…** 2010.

LIBERATO, A. et al. Avaliação de desempenho de Plataformas para Validação de Redes Definidas por Software. In: CSBC 2014 - WPERFORMANCE, 2014. **Proceedings…** 2014.

LINDA, Ondrej; VOLLMER, Todd; MILOS. Manic Neural network based intrusion detection system for critical infrastructures. In: 'IJCNN'09, INT. JOINT INNS-IEEE CONF. ON NEURAL ETWORKS', 2009, Atlanta, Georgia, USA. **Proceedings…** 2009. p. 1827–1834.

LYNCH, D. M. Securing against insider attacks. **Information Systems Security,** v.15, n.5, p.39–47, 2006.

LOPES, Martin A. et al. An Elastic Intrusion Detection System for Software Networks. In: ANNALS OF TELECOMMUNICATIONS, 2016. **Proceedings…** 2016.

LOPES, Y. **Tecnologia SDN promove o fim da escuridão na Rede de Comunicação.** 2015. Disponível em: <http://www3.selinc.com.br/news/?p=1259>. Acesso em: 13 maio 2016.

MACEDO, D. F. Programmable networks-from software-defined radio to software-defined networking. **IEEE Communications Surveys and Tutorials,** v.17, n.2, p.1102–1125, 2015.

MANDUJANO, S. A **Multiagent approach to outbound intrusion detection**. [S.l.]: [s.n.], 2004.

MARCIANO, J. L.; MARQUES, M. L. O enfoque social da segurança da informação. **Ci. Inf.,** Brasília, v. 35, n. 3, p. 89-98, set. /dez. 2006.

MARCHESAN, G.; MEDINA, R. D. Simulando Cenários para Redes Definidas por Sofware. In: SIMPÓSIO DE PESQUISA E DESENVOLVIMENTO EM COMPUTAÇÃO, 2015. **Anais...** 2015.

MARCONDES, Cesar. **Projeto de Desenvolvimento em OpenFlow**. [S.l.]: [s.n.], 2011.

MALLERY, John. **Hardening Linux**. [S.l.]: McGraw-Hill Osborne Media, 2004.

MATTOS, D. et al. OMNI: OpenFlow MaNagement Infrastructure. In: NETWORK OF THE FUTURE (NOF), INTERNATIONAL CONFERENCE, 2011. **Proceedings…** 2011.

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. In: SBSEG, 13., 2013. **Proceedings…** 2013.

MCKEOWN, N. et al. OpenFlow: Enabling innovation in campus networks. SIGCOMM Comput. **Commun. Rev.,** v.38, n.2, p. 69–74, 2008.

MONTORO, Rodrigo. **Introdução ao Snort - Serie Snortando (Parte 1).** São Paulo, 2012. Disponível em: <http://spookerlabs.blogspot.com.br/2012/01/introducao-ao-snort-seriesnortando.html >. Acesso em: 5 jun. 2015.

MOREIRA, Marcelo D. et al. Internet do futuro: um novo horizonte. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES-SBRC, 2009. **Anais...** 2009.

MOREIRA, Nilton Stringasci. **Segurança mínima.** Rio de Janeiro: Axcel Books do Brasil, 2001.

NAGAHAMA, F. Y. et al. **Ipsflow–uma proposta de sistema de prevenção aode intrusão baseado no framework OpenFlow**. [S.l.]: [s.n.], 2012.

NAKAMURA, E. **Segurança em de redes em cooperativos**. São Paulo: Novatec, 2007.

NAOUS, J. et al. Netfpga: Reusable router architecture for experimental research. In: ACM WORKSHOP ON PROGRAMMABLE ROUTERS FOR EXTENSIBLE SERVICES OF TOMORROW, PRESTO 8., 2008. New York, NY, USA. **Proceedings**… New York, NY, USA: ACM, 2008.

NUNO, V. C. Guedes; RODRIGUES L., S. **Plataforma de desenvolvimento e simulação de protocolos.** [S.l.]: [s.n.], 2005.

OLIVEIRA, Rafael Emerick Z de et al. **Parâmetros balizadores para experimentos com comutadores openflow:** avaliaçao experimental baseada em mediçoes de alta precisao. [S.l.]: SBrT, 2012.

ONF. **White Paper**. **OpenFlow Switch Specification Versão 1.4.0**. California, USA. [s.n.], 2013. 205p.

ONF. **Open Networking Foundation.** [Online]. 2014. Disponível em: <http://www.opennetworking.org>. Acesso em: 12 jul. 2015.

ONF. **SDN Architecture**. 2014b. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf >. Acesso em: 12 jul. 2015.

ONSUMMIT. **Open Networking Summit.** 2012. Disponível em: <http://opennetsummit.org/>. Acesso em: 3 jul. 2015.

OPENFLOW. **OpenFlow Specification**. 2012. Disponível em: <http://OpenFlowSwitch.org.>. Acesso em: Jul de 2015.

OPENWRT. **OpenWRT Wireless Freedom**. 2016. Disponível em: <https:// openwrt. org/>. Acesso em: 18/10/2016.

PEGDEN, C. D., SHANNON, R. E.; SADOWSKI,R. P. **Introduction to simulation using SIMAN.** 2. ed. New York: McGraw-Hill, 1995.

PEIXOTO, MÀRIO C. P. **Engenharia social e segurança da informação na gestão corporativa.** Rio de Janeiro: Brasport, 2006.

PFAFF, B. et al. Extending networking into the virtualization layer. In: ACM WORKSHOP ON HOT TOPICS IN NETWORKS-HOTNETS, 2009. **Proceedings**… New York, NY, USA: ACM, 2008.

PIGRELAY. Disponível em: <https://github.com/John-Lin/pigrelay/blob/master/pigrelay.py>, 2014. Acesso em: 4 jul. 2015.

PIMENTEL, J. F. A; FAGUNDES, P. C. **Snort sistemas distribuídos**. Juiz de Fora, 2009. Disponível em: <http://www.jeanpimentel.com.br/67/snort>. Acesso em: 15 set. 2016.

PORRAS, P. et al. A security enforcement kernel for OpenFlow networks. In: WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKS ACM, 5., 2012. **Proceedings**… 2012.

PRITCHETT, Willie L.; SMET, David D. **Kali Linux Cookbook.** [S.l.]: Packt Publishing, 2013.

REZENDE, P. H. A. **Extensões na Arquitetura SDN para o Provisionamento de QoS através do Monitoramento e uso de múltiplos caminhos**. 2016. Dissertação (Mestrado)- UFU, Faculdade de Computação, Programa de Pós-Graduação em Ciência da Computação, São Paulo, 2016.

RODRÍGUEZ, F. L. **Arquitetura e protótipo de uma rede sdn-openflow para provedor de serviço**. 2014. Dissertação (Mestrado)-UNB, Faculdade de Tecnologia, Departamento de Engenharia Elétrica, DF, 2014.

ROTHENBERG, C. E. OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. **Cad. CPqD Tecnologia**, Campinas, 7, n.1, p.65–76, 2010.

RYU, P.T. **Ryu sdn framework using OpenFlow 1.3**. 2014. Disponível em: <http: //osrg.github.io/ryu- book/en/ Ryubook.pdf>. Acesso em: 12 jul. 2015.

RFC 7426. **Software-Defined Networking (SDN)**: layers and architecture terrminology. Disponível em: <http://www.ietf.org/>. Acesso em: 9 set. 2016.

SALVADOR, Gustavo. **Entendendo os fundamentos da segurança da informação**. 2013. Disponível em: <http://www.profissionaisti.com.br/2013/10/entendendo-os-fundamentos-da-seguranca-da-informacao>. Acesso em: 12 nov. 2015.

SCARFONE, Karen; MELL, Peter. **Guide to Intrusion Detection and Prevention Systems (IDPS)**. Recommendations of the National Institute of Standards and Technology. Gaithersburg: National Institute of Standards and Technology, 2007. p. 1-127.

SHENKER, Scott et al. **The future of networking, and the past of protocols**. Open Networking Summit. [S.l.]: [s.n.], 2011.

SHERWOOD, R. et al. Carving research slices out of your production networks with OpenFlow. SIGCOMM Comput. **Commun. Rev.,** v.40, p.129–130, 2010.

SHIN, S. et al. FRESCO: Modular composable security services for software-defined networks. In: NETWORK AND DISTRIBUTED SECURITY SYMPOSIUM, 2013. **Proceedings…** 2013.

SNORT. **Snort.** Disponível em: <http://www.snort.org/>. Acesso em: Set 2016.

UNNIKRISHANAN, D. et al. Scalable network virtualization using fpgas. In: ANNUAL ACM/SIGDA INTERNATIONAL SYMPOSIUM ON FIELD PROGRAMMABLE GATE ARRAYS, 18., 2010. **Proceedings…** 2010.

VAUGHAN-NICHOLS, Steven J. OpenFlow: The Next Generation of the Network? **Computer Magazine**, v. 44, ed. 8, p. 13-15, ago. 2011.

WANG, J. **Computer network security**: theory and practice. [S.l.]: Higher Education Press, 2009

XING, T. et al. Snort-flow: A OpenFlow-based intrusion prevention system in cloud environment. In: RESEARCH AND EDUCATIONAL EXPERIMENT WORKSHOP (GREE), 2., 2013. **Proceedings…** 2013. p. 89–92.

APÊNDICE A - CÓDIGO SWITCH_SNORT.PY

Código switch_snort.py

```
# -*- coding: utf-8 -*-
# Copyright (C) 2013 Nippon Telegraph and Telephone Corporation.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.


from __future__ import print_function
import array
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ipv4
from ryu.lib.packet import icmp
from ryu.lib import snortlib
from ryu.ofproto.ofproto_v1_3 import OFPG_ANY
from mitigacao import *
class SimpleSwitchSnort(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'snortlib': snortlib.SnortLib}
```

```
def __init__(self, *args, **kwargs):
        super(SimpleSwitchSnort, self).__init__(*args, **kwargs)
        self.snort = kwargs['snortlib']
        self.snort_port = 3 # por padrao, todos os pacotes vão para a porta 3
        self.mac_to_port = {}
        self.mac_flood = []  # lista com mac de flood
        self.dictDatapath = {}
        self.mitigacao = mitigacao()
        self.mac_flood = self.mitigacao.loadFile()
        self.mac_white = self.mitigacao.loadWhiteFile()
        socket_config = {'unixsock': False}
        self.snort.set_config(socket_config)
        self.snort.start_socket_server()

def get_package(self, pkt):
        pkt = packet.Packet(array.array('B', pkt))
return pkt

def packet_print(self, pkt):
        pkt = packet.Packet(array.array('B', pkt))
        eth = pkt.get_protocol(ethernet.ethernet)
        _ipv4 = pkt.get_protocol(ipv4.ipv4)
        _icmp = pkt.get_protocol(icmp.icmp)
        if _icmp:
             self.logger.info("%r", _icmp)
        if _ipv4:
             self.logger.info("%r", _ipv4)
        if eth:
             self.logger.info("%r", eth)
        return eth


'''Function : _dump_alert
args: ev - pacote contendo a msg enviada pelo snort
Description: Função que ao receber os alertas encaminha esse pacote para a classe
mitigacao
a qual ira dar o Tratamento necessario, de acordo com o a classe do ataque.
```

```
'''
    @set_ev_cls(snortlib.EventAlert, MAIN_DISPATCHER)
        def _dump_alert(self, ev):
        msg = ev.msg
        self.mac_flood =
self.mitigacao.checkmessenger(ev,self.mac_flood,self.dictDatapath,
                                self.get_package(msg.pkt))
   @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
   def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        msg = ev.msg

 # install table-miss flow entry
#
# We specify NO BUFFER to max_len of the output action due to
# OVS bug. At this moment, if we specify a lesser number, e.g.,
# 128, OVS will send Packet-In with invalid buffer_id and
# truncated packet data. In that case, we cannot output packets
# correctly.
        match = parser.OFPMatch()
        actions =
[parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

   def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=match,
instructions=inst)
        datapath.send_msg(mod)

# Este evento é gerado sempre, fica ocioso esperando receber pacotes
   '''Function : _packet_in_handler
   args: ev - Pacotes vindos de maquinas que querem se conectar a rede
   Description: Função "principal", nela e que todas as maquinas que desejam se conectar a
```

```
rede devem passar
   Para tratamento dos pacotes, antes de enviar para o switch um pedido de criação de nova regra
   e testado se aquele mac que quer se conectar a rede e um mac contido primeiramente na whitelist
   e se ele não esta contido na blacklist
   '''
   @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
   def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath # http://ryu.readthedocs.io/en/latest/ryu_app_api.html#ryu-controller-controller-datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']  #porta que foi recebida a msg
        pkt = packet.Packet(msg.data)  #dado da msg
        eth = pkt.get_protocols(ethernet.ethernet)[0]  #protocols
        dst = eth.dst
        src = eth.src
        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})
''' dictDatapath -  Dicionario no qual eu associo o mac que esta buscando uma conexão
   com seu datapath, table_id e Porta de entrada para que quando o mesmo mac vier a atacar
   o switch eu consigo obter todas essas informações a partir do mac dele'''
        self.dictDatapath[src] = [datapath, msg.table_id, in_port]
        # self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)
        # learn a mac address to avoid FLOOD next time.  />>
        self.mac_to_port[dpid][src] = in_port  #fonte da msg
        if dst in self.mac_to_port[dpid]:  # se dentro dessa lista tem o destinatario da msg
            out_port = self.mac_to_port[dpid][dst]    # pego o destinatario e coloco como porta de saida
        else:
            out_port = ofproto.OFPP_FLOOD  # se não eu coloco essa constante como porta de saida oq ser FPP_FLOOD
        actions = [parser.OFPActionOutput(out_port), parser.OFPActionOutput(self.snort_port)]

        #install a flow to avoid packet_in next time
```

```
        print("#############################################################")
        print("#+++++++++++++++    Packet_in treatment    +++++++++++++++#")
        print("#############################################################")
        print("+++++++++++++++   Registered MAC Address    +++++++++++++++")
        print(self.mac_white)
        print("+++++++++++++++  Enderecos MACs na Blacklist  +++++++++++++++")
        print(self.mac_flood)
        print("+++++++++++++++    MAC Source  +++++++++++++++")
        print(src)
        if src in self.mac_white:
         if src not in self.mac_flood and dst not in self.mac_flood:
                print ('Criando regra de fluxo...')
                 if out_port != ofproto.OFPP_FLOOD:  #All physical ports, except the input port
and those disabled by Spanning Tree Protocol.
                        match = parser.OFPMatch(in_port=in_port, eth_dst=dst)  # oq ser isso
                        self.add_flow(datapath, 1, match, actions)
                        data = None
                        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
                                data = msg.data
                        out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                                in_port=in_port, actions=actions, data=data)
                        datapath.send_msg(out)
                        else:
                                print(' Packet_in nao atendido - MAC Suspeito! ')
        else:
            print(' MAC nao cadastrao - Contate o Administrador! ')
```

APÊNDICE B - CÓDIGO MITIGACAO.PY

```
# -*- coding: utf-8 -*-

from ryu.lib.packet import ethernet

import os

class mitigacao(object):

        ##"""docstring for Mitigacao"""

        def __init__(self):

                super(mitigacao, self).__init__()

'''

Function : remove_table_flows

args: datapath -  dados da messagem que será enviada para o switch

          table_id -  ID da tabela na qual esta contida a regra de fluxo

          match - Elemento vazio neste caso

          instructions - Elemento vazio neste caso

Description: A função gera json( Sequencia de Caracteres pré configuradas), e utilizando o argumento Ofproto.OFPFC_DELETE

Quando enviado para o switch ele deletar a regra com  o mac e port_in correspondentes.

'''

        def remove_table_flows(self, datapath, table_id, match, instructions):

            ##"""Create OFP flow mod message to remove flows from table."""

            ofproto = datapath.ofproto

            flow_mod = datapath.ofproto_parser.OFPFlowMod(datapath, 0, 0 ,table_id ,
ofproto.OFPFC_DELETE,0,
0,11111,ofproto.OFPCML_NO_BUFFER,ofproto.OFPP_ANY,ofproto.OFPG_ANY,0, match,
instructions)

            datapath.send_msg(flow_mod)

'''

Function : modify_table_flows

args: datapath -  dados da messagem que será enviada para o switch

          table_id -  ID da tabela na qual esta contida a regra de fluxo

          match - Contem o mac destino  e a porta de entrada do pacote criador da regra

          instructions - Contem qual tipo de ação deve ser tomada

Description: A função gera json, e utilizando o argumento Ofproto.OFPFC_MODIFY

Quando enviado para o switch ele modifica a tabela com o argumento actions que esta dentro do instructions.

'''

        def modify_table_flows(self, datapath, table_id, match, instructions):


            ##"""Create OFP flow mod message to remove flows from table."""
```

```
                ofproto = datapath.ofproto

                flow_mod = datapath.ofproto_parser.OFPFlowMod(datapath, 0, 0 ,table_id ,
ofproto.OFPFC_MODIFY,0,
0,11111,ofproto.OFPCML_NO_BUFFER,ofproto.OFPP_ANY,ofproto.OFPG_ANY,ofproto.OFP
FF_SEND_FLOW_REM, match, instructions)

                datapath.send_msg(flow_mod)
'''
Function : checkmessenger
args: ev -  dados da messagem que será enviada para o switch
          mac_flood -  ID da tabela na qual esta contida a regra de fluxo
          dictDatapath - Contem o mac destino  e a porta de entrada do pacote criador da
regra
          pkt - Contem qual tipo de ação deve ser tomada
Description: A função gera json, e utilizando o argumento Ofproto.OFPFC_MODIFY
Quando enviado para o switch ele modifica a tabela com o argumento actions que esta
dentro do instructions.
'''
        def checkmessenger(self,ev,mac_flood,dictDatapath,pkt):
            msg = ev.msg
            msgconvert = str(msg.alertmsg[0])
            if msgconvert.find ("Class_3") !=-1:
                lista = open("blacklist.txt","a+")
                escrevendo = str(pkt.get_protocol(ethernet.ethernet).src+"\n")
                l = lista.readlines()
            print("#+++   Packet_Event treatment - Class 3 attack   +++#")
            print("#**  Attempt to suspicous activity on the network **#")
            print (l)
            if escrevendo not in l:
                lista.write(escrevendo)
                mac_flood.append(pkt.get_protocol(ethernet.ethernet).src)
                print("#*** Inserting suspect MAC Address in Blacklist ***#")
                print("alertmsg: %s" % "".join(msg.alertmsg))
                lista.close()
                print(pkt.get_protocol(ethernet.ethernet).src)
                mc = pkt.get_protocol(ethernet.ethernet).src #saving mac address to
remove
                parser = dictDatapath[mc][0].ofproto_parser
                empty_match = parser.OFPMatch(in_port=dictDatapath[mc][2])
                empty_match2 = parser.OFPMatch(eth_dst=mc)
```

```
self.remove_table_flows(dictDatapath[mc][0],dictDatapath[mc][1],empty_match,[])

self.remove_table_flows(dictDatapath[mc][0],dictDatapath[mc][1],empty_match2,[])
                        return mac_flood
            if msgconvert.find ("Class_2") !=-1:
                print("#+++   Packet_Event treatment - Class 2 attack   +++#")
                #print(msgconvert)
                print("#**  Attempt to suspicous activity on the network **#")
                print("alertmsg: %s" % "".join(msg.alertmsg))
                #print(pkt.get_protocol(ethernet.ethernet).src)
                mc = pkt.get_protocol(ethernet.ethernet).src #saving mac address to remove
                parser = dictDatapath[mc][0].ofproto_parser
                dst = pkt.get_protocol(ethernet.ethernet).dst
                empty_match = parser.OFPMatch(in_port=dictDatapath[mc][2],eth_dst=dst)
                actions = [parser.OFPActionOutput(4)]
                instructions = [parser.OFPInstructionActions(dictDatapath[mc][0].ofproto.
                                        OFPIT_APPLY_ACTIONS, actions)]

self.modify_table_flows(dictDatapath[mc][0],dictDatapath[mc][1],empty_match,instructio
ns)
                return mac_flood
'''
Function : loadFile
args:
Description: Função que carrega os mac que estão dentro do arquivo blacklist.txt
'''
            def loadFile(self):
                #leitura do arquivo
                mac_flood=[]
                if os.stat("blacklist.txt").st_size == 0:
                      return mac_flood
                else:
                       arc = open("blacklist.txt","r")
                       for i in arc.readlines():
                              s = i.replace("\n","")
                              #print i
```

```
                                mac_flood.append(s)
                                return mac_flood


'''Function : loadWhiteFile
args:
Description: Função que carrega os mac que estão dentro do arquivo whitelist.txt
'''
        def loadWhiteFile(self):
                mac_flood=[]
                if os.stat("whitelist.txt").st_size == 0:
                        return mac_flood
                else:
                   arc = open("whitelist.txt","r")
                   for i in arc.readlines():
                      s = i.replace("\n","")
                      #print i
                      mac_flood.append(s)
                     return mac_flood
```

APÊNDICE C - PROCEDIMENTO DE COMPILAÇÃO DO SWITCH OPENFLOW (TPLINK)

Digite o seguinte na mesma linha:

```
apt-get install git quilt build-essential binutils flex bison
autoconf gettext   texinfo sharutils subversion libncurses5-dev
ncurses-term zlib1g-dev gawk
exit
mkdir ~/openwrt
cd ~/openwrt
svn co svn://svn.openwrt.org/openwrt/trunk/
cd trunk
./scripts/feeds update -a
./scripts/feeds install -a
make menuconfig
```

Select Target Profile (TP-LINK TL-WR1043N/ND)

```
make prereq
make
cd ~/openwrt
git clone https://github.com/CPqD/OpenFlow-openwrt.git
cd ~/openwrt/trunk/package/
ln -s ~/openwrt/OpenFlow-openwrt/OpenFlow-1.3/
cd ~/openwrt/trunk/

ln -s ~/openwrt/OpenFlow-openwrt/OpenFlow-1.3/files
make menuconfig
```

Select (*) OpenFlow package under network
Select (*) tc package under network
Select kmod-tun under Kernel Modules->Network Support
Save and Exit

```
make kernel_menuconfig
```

Under Networking Support->Networking options->QoS and/or fair queueing select Hierarchical Token Bucket (HTB)
Save and Exit
Dica opcional: se quiser gravar os arquivos de configuração para que eles fiquem na imagem compilada, os grave em `~/openwrt/trunk/files/etc/config`
Agora faça a compilação final:

```
make
```

A imagem estará em ~/openwrt/trunk/bin/ar71xx/openwrt-ar71xx-generic-tl-wr1043nd-v1-jffs2-factory.bin, após a longa compilação. Copie a imagem criada para a sua estação de trabalho e utilizande o WinSCP ou similar. Fazer o upgrade do firmware pela tela de gerência web do TP-link.
Caso o roteador já esteja com o OpenWRT instalado, copie com o WinSCP (modo SCP) o arquivo para o /tmp do TP-Link e utilize o seguinte comando para fazer o upgrade:

```
mtd -r write /tmp/openwrt-ar71xx-generic-tl-wr1043nd-v1-jffs2-
factory.bin firmware
```

Depois do upgrade feito, conecte o seu micro em uma das portas **LAN** do roteador e acesse por telnet (ou Putty) o endereço 192.168.1.1. Primeiramente crie o seguinte vínculo estático. Sem ele, os processos do OpenFlow não "sobem":

```
cd /etc

ln -s /lib/functions.sh

cd /sbin

ln -s /usr/sbin/tc
```

APÊNDICE D - CONFIGURAÇÃO DE REDE DO SWITCH OPENFLOW (TPLINK)

```
root@zeus:/etc/config# cat network

config switch

        option name 'rtl8366rb'

        option reset '1'

        option enable_vlan '1'

        option enable_learning '0'

config switch_vlan

        option device 'rtl8366rb'

        option vlan '1'

        option ports '1 5t'

config switch_vlan

        option device 'rtl8366rb'

        option vlan '2'

        option ports '2 5t'

config switch_vlan

        option device 'rtl8366rb'

        option vlan '3'

        option ports '3 5t'

config switch_vlan

        option device 'rtl8366rb'

        option vlan '4'

        option ports '4 5t'

config switch_vlan

        option device 'rtl8366rb'

        option vlan '5'

        option ports '0 5t'
```

```
config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'
config interface
        option ifname 'eth0.1'
        option proto 'static'
config interface
        option ifname 'eth0.2'
        option proto 'static'
config interface
        option ifname 'eth0.3'
        option proto 'static'
config interface
        option ifname 'eth0.4'
        option proto 'static'
config interface 'wan'
        option ifname 'eth0.5'
        option proto 'static'
        option ipaddr '192.168.1.111'
        option netmask '255.255.255.0'
        option type 'bridge'
config interface 'wwan'
        option proto 'dhcp'
```

APÊNDICE E - PUBLICAÇÕES E PREMIAÇÃO DO TRABALHO

Campos, Maxli; Martins, Joberto. A Security Architecture Proposal for Detection and Response to Threats in SDN Networks, IEEE Andean Council International Conference - ANDESCON, Peru, 2016.

Campos, Maxli; Martins, Joberto. Uma Proposta de Arquitetura de Segurança para Detecção e Reação a Ameaças em Redes SDN, Encontro Unificado de Computação - ENUCOMP, Teresina-PI, 2016.

Campos, Maxli; Martins, Joberto. A SDN-based Flexible System for On-the-Fly Monitoring and Treatment of Security Events, International Workshop on ADVANCES in Infrastructures and Service - ADVANCE'17, Evry-France, 2017.

Tendo em vista o trabalho "Uma Proposta de Arquitetura de Segurança para Detecção e Reação a Ameaças em Redes SDN", apresentado no Encontro Unificado de Computação - ENUCOMP, Teresina-PI, 2016, ter sido avaliado pela banca avaliadora como o 3° melhor trabalho do evento, os autores foram convidados a publicar o artigo na Revista Brasileira de Computação Aplicada (RBAC), conforme email abaixo enviado pelo Editor da Revista.

Gmail — Max Campos <mcampos.2004@gmail.com>

[RBCA] Decisão editorial
1 mensagem

Prof. Carlos Amaral Hölbig <rbca@upf.br> — 19 de dezembro de 2016 13:40
Responder a: Revista Brasileira de Computação Aplicada <rbca@upf.br>
Para: Senhor Maxli Barroso Campos <mcampos.2004@gmail.com>
Cc: Joberto Martins <joberto.martins@gmail.com>

Prezados autores,

Temos a satisfação em informar que seu artigo foi aceito para publicação na Revista Brasileira de Computação Aplicada (RBCA - ISSN 2176-6649) pelo fato de ter sido selecionado entre os melhores artigos do ENUCOMP 2016.

Para dar continuidade ao processo de publicação solicitamos o envio de seu artigo em formato .doc ou .tex (neste caso com todos os arquivos de figuras utilizados no texto), sem falta, até o dia 15/01/17, para o e-mail da revista (rbca@upf.br), com o intuito de procedermos a correção ortográfica e de formato do trabalho, conforme normas e exigências da Editora da UPF.

A versão final do artigo será enviada posteriormente aos autores para concordância e posterior publicação.

Atenciosamente,

Prof. Dr. Carlos Amaral Hölbig
Editor RBCA

Revista Brasileira de Computação Aplicada
E-mail: rbca@upf.br
http://www.upf.br/seer/index.php/rbca