UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO DA UFSC

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

PAULO VIEIRA CENTENO

Uma análise de segurança de Redes Definidas por Software sobre protocolo OpenFlow

TRABALHO DE CONCLUSÃO DE CURSO

Florianópolis

2016

PAULO VIEIRA CENTENO

Uma análise de segurança de Redes Definidas por Software sobre protocolo OpenFlow

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação pela Universidade Federal de Santa Catarina.

Orientadora: Profa. Dra. Carla Merkle Westphall

Florianópolis
2016

Catalogação na fonte elaborada pela biblioteca da Universidade Federal de Santa Catarina

A ficha catalográfica é confeccionada pela Biblioteca Central.
Tamanho: 7cm x 12 cm
Fonte: Times New Roman 9,5
Maiores informações em:
http://www.bu.ufsc.br/design/Catalogacao.html

Paulo Vieira Centeno

UMA ANÁLISE DE SEGURANÇA DE REDES DEFINIDAS POR SOFTWARE SOBRE PROTOCOLO OPENFLOW

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientadora: Profª. Dra. Carla Merkle Westphall

Banca Examinadora:

Prof. Dr. Carlos Becker Westphall
Universidade Federal de Santa Catarina

Prof. Dr. Mário Antônio Ribeiro Dantas
Universidade Federal de Santa Catarina

Resumo

O rápido crescimento no uso de computação em nuvem em substituição à computação convencional está levando a infra-estrutura atual de redes a seu limite, criando desafios relacionados a sua escalabilidade e administração. Neste cenário, surgiu a abordagem conhecida como Redes Definidas por Software (Software Defined Network - SDN), que torna possível redefinir a topologia e tomar decisões de roteamento dinamicamente, com uma rede programável. É viabilizada uma nova gama de aplicações e serviços, sem a preocupação com os antigos protocolos de comunicação em cima de enlaces de comunicação com altas taxas de transmissão.

Entretanto, a flexibilidade que SDN incorpora às arquiteturas de rede veio acompanhada de riscos na segurança das redes. Atualmente, as aplicações desenvolvidas e implementadas consideram paradigmas de protocolos obsoletos de comunicação. Não foram projetadas para esse ambiente de redes inteligentes, com potenciais falhas de segurança decorrentes desse fato. O ambiente é híbrido, com aplicações convencionais, desenvolvidas para redes convencionais, operando sobre Redes Definidas por Software.

O presente trabalho apresenta uma análise descritiva de Redes Definidas por Software sobre protocolo OpenFlow. Em adição, demonstra empíricamente problemas de segurança decorrentes da adoção do protocolo OpenFlow.

Palavras-chave: Redes Definidas por Software, Software Defined Network, SDN, OpenFlow, segurança, ataques.

Abstract

The rapid growth in the use of cloud computing as a substitute for conventional computing is taking the current network infrastructure to its limits, creating challenges related to its scalability and administration. In this scenario, the approach known as Software Defined Network (SDN) has emerged, which makes it possible to redefine the topology and make routing decisions dynamically, with a programmable network. A new range of applications and services is possible without the concern of old communication protocols over high-rate communication links.

However, the flexibility that SDN incorporates into network architectures has been accompanied by network security risks. Currently, the developed and implemented applications consider paradigms of obsolete communication protocols. They were not designed for this intelligent networking environment, with potential security breaches arising from this. The environment is hybrid, with conventional applications, developed for conventional networks, operating on Software Defined Networks.

This work presents a descriptive analysis of Software Defined Networks over OpenFlow protocol. In addition, it demonstrates empirically security issues arising from the adoption of the OpenFlow protocol.

Keywords: Software Defined Networks, SDN, OpenFlow, security, attacks.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Acontece uma rápida expansão da computação em nuvem em substituição à computação convencional. Esse crescimento está levando a infra-estrutura atual de redes a seu limite. As ferramentas atualmente existentes para administração dos recursos de infra-estrutura apresentam rigidez na política de mudanças e um alto nível de complexidade em sua gerência, tornando necessário investigar novas maneiras de administrar seus recursos (FEAMSTER; REXFORD; ZEGURA, 2014; KIM; FEAMSTER, 2013).

Em meio a esse cenário, está emergindo um novo tipo de arquitetura de redes, chamado de Rede Definida por Software. Em uma Rede Definida por Software, uma máquina logicamente centralizada, chamada de controlador, administra um conjunto distribuído de comutadores. Essa máquina é um computador comum de uso geral, capaz de computações arbitrárias que permitem redefinir a topologia da rede e tomar decisões de roteamento. Redes Definidas por Software permitem a implementação de uma nova gama de aplicações e serviços nas redes de computadores (AMIN, 2016).

Espera-se que o uso de Redes Definidas por Software mude a forma como centros de dados são projetados. A rede pode se ajustar dinamicamente ao tráfego, adaptando-se automaticamente a políticas de balanceamento de carga de forma transparente aos usuários. As políticas também podem estar baseadas na otimização do consumo de energia dos elementos de rede. Os recursos de hardware são facilmente compartilhados, em comparação com a computação convencional (HU; HAO; BAO, 2014).

O protocolo OpenFlow, o mais comumente adotado para implementação de Redes Definidas por Software, precisa ser analisado em termos de segurança, ou pode levar a riscos de segurança severos (BRANDT et al., 2014; BENTON; CAMP; SMALL, 2013). Há uma complexidade associada a essa flexibilidade. Um resultado não desejado dessa nova forma de administrar redes é a exposição de falhas de segurança com potencial de exploração por atacantes maliciosos. Tais falhas são decorrentes de características da arquitetura de SDN. Criam novos desafios, conseqüência da abstração de alto nível para compartilhamento de recursos de hardware que SDN oferece (MATTOS; DUARTE, 2014).

OpenFlow é o padrão aberto mais amplamente aceito e implantado para SDN. Ele fornece uma especificação comum para implementação de dispositivos habilitados para OpenFlow, e para o canal de comunicação entre dispositivos de plano de dados e de plano de controle, como por exemplo comutadores e controladores (KREUTZ et al.,

2015). O modelo protocolo OpenFlow é descrito pelo presente trabalho. Uma análise é feita com especial atenção ao ponto de vista da segurança, sugerindo pontos que facilitem ataques baseados na arquitetura desse tipo de rede.

É proposta uma demonstração empírica de desafios de segurança em Redes Definidas por Software usando o protocolo OpenFlow. A plataforma de execução dessa demonstração é o emulador de redes Mininet. Mininet é um sistema de prototipagem rápida. O principal objetivo do sistema é o desenvolvimento rápido de aplicações do controlador OpenFlow. Mininet emula redes baseado em técnicas de virtualização, proporcionando novas possibilidades para a estrutura de rede e simulação (ANTONENKO; SMELYANSKIY, 2013). Uma configuração que permita executar tal demonstração é desenvolvida.

1.1 Problema

Conforme Kim e Feamster (2013), a operação e manutenção de uma rede de computadores é uma tarefa difícil. Para expressar as políticas de rede de alto nível, os operadores de rede precisam configurar cada dispositivo de rede, separadamente - a partir de um conjunto heterogêneo de comutadores, roteadores, etc. - usando comandos de baixo nível e específicos de cada fornecedor. Além da complexidade de configuração, as redes são dinâmicas. Os operadores têm poucos mecanismos para responder automaticamente a eventos de rede. É difícil aplicar as políticas necessárias nesse ambiente de constante mudança.

A separação entre plano de controle e plano de dados constitui a base para as Redes Definidas por Software. Comutadores de rede tornam-se dispositivos de encaminhamento simples. A lógica de controle é implementada e centralizada em um controlador, que é a entidade que regula o comportamento da rede. Essa centralização traz benefícios. É mais simples e menos propenso a erros modificar as políticas de rede através de software do que através de configurações de dispositivos de baixo nível (AMIN, 2016).

Um programa de controle pode reagir automaticamente a alterações ilegítimas do estado da rede e, assim, manter as políticas de alto nível em seu devido lugar. A centralização da lógica de controle em um controlador com conhecimento global do estado da rede simplifica o desenvolvimento de funções de rede mais sofisticadas. Esta capacidade de programar a rede, a fim de controlar o plano de dados subjacente é a proposta de valor das Redes Definidas por Software (HU; HAO; BAO, 2014).

As principais causas de preocupação encontram-se justamente nos principais benefícios das Redes Definidas por Software: a programação da rede e a centralização

da lógica de controle. Esses recursos introduzem novas falhas e ataques aos planos, abrindo as portas para novas ameaças que não existiam antes ou eram mais difíceis de explorar. Redes tradicionais têm “proteções naturais” contra o que seriam vulnerabilidades comuns em sistemas de TI convencionais. Ou seja, a natureza fechada, proprietária, dos dispositivos de rede, o seu projeto bastante estático, a heterogeneidade do software, bem como a natureza descentralizada do plano de controle representam defesas contra ameaças comuns. Por exemplo, um ataque explorando uma vulnerabilidade peculiar de um conjunto específico de dispositivos de um único fornecedor, potencialmente prejudica apenas uma parte da rede. Esta diversidade é comparativamente menor em SDNs. Um padrão comum entre os fornecedores e os clientes, como o OpenFlow, também pode aumentar o risco e a eventual introdução de falhas comuns em implementações desses protocolos e do software de plano de controle (KREUTZ; RAMOS; VERISSIMO, 2013).

1.2 Justificativa

A implementação de Redes Definidas por Software através do uso do protocolo OpenFlow traz um desafio: uma evolução extremamente promissora de arquiteturas de rede, contra um aumento perigoso no potencial das ameaças de ataques. Devemos preservar os benefícios dessa evolução, neutralizando os perigos potencializados por essa mesma evolução (KREUTZ; RAMOS; VERISSIMO, 2013).

1.3 Objetivos

O principal objetivo deste trabalho é indicar vulnerabilidades da arquitetura SDN sobre protocolo OpenFlow, simulando um ou mais ataques que explorem tais vulnerabilidades.

Os objetivos específicos são:

a) Simular um ambiente de Redes Definidas por Software virtual.

b) Gerar tráfego na simulação.

c) Testar e validar a implementação do ambiente de teste.

d) Simular ataques explorando características da arquitetura de Redes Definidas por Software no ambiente de teste.

e) Documentar e apresentar os resultados.

1.4 Estrutura dos capítulos

Este Trabalho de Conclusão está organizado da seguinte forma: o capítulo 2 faz uma breve revisão bibliográfica sobre Redes Definidas por Software. Além disso, traz conceitos relativos ao protocolo OpenFlow e o simulador de redes Mininet, depois descrevendo suscintamente o comutador virtual OpenvSwitch e alguns controladores OpenFlow disponíveis.

O capítulo 3 sugere vulnerabilidades expostas por SDN implementadas com protocolo OpenFlow,

Já o capítulo 4 descreve o ambiente de simulação utilizado, assim como as experiências realizadas, e seus resultados.

Finlmente, o capítulo 5 traz as conclusões e sugere trabalhos futuros relacionados com o presente Trabalho de Conclusão.

2 Redes Definidas por Software (SDN)

Redes de computadores têm sido construídas como um conjunto de dispositivos de hardware com propósitos distintos entre si, processando informações sobre políticas, monitoramento de tráfego e roteamento, entre muitos outros serviços. Essas tarefas são configuradas através de interfaces de configuração bastante heterogêneas entre si, que permitem configurar separadamente, por exemplo, firewalls, roteadores, comutadores, listas de acesso e balanceamento de carga.
Rede Definida por Software (SDN), é um novo paradigma em configuração de redes de computadores.

> O paradigma de Redes Definidas por Software se baseia na separação das funções de controle, no plano de controle, das funções de encaminhamento de quadros, no plano de encaminhamento. A ideia chave da separação é prover maior flexibilidade às funções de controle enquanto o hardware especializado para comutar quadros a alta velocidade permanece inalterado. Logo, a Rede Definida por Software oferece uma alta programabilidade das funções de controle da rede em um comutador com alto desempenho de encaminhamento de quadros. O operador pode definir de maneira simples os fluxos e as ações sobre os fluxos através de uma interface de programação de aplicação (MATTOS; DUARTE, 2014).

De acordo com Foster et al. (2013), Rede Definida por Software é uma rede onde um controlador centralizado administra uma coleção distribuída de comutadores, permitindo ao programador controlar o comportamento de toda a rede sem precisar acessar diretamante diretamente cada comutador; em vez disso, o controlador implementa as regras de encaminhamento de pacotes nos comutadores que administra. O resultado é uma ilusão de controle centralizado do comportamento da rede.

Kreutz, Ramos e Verissimo (2013) afirmam que com a separação do plano de controle do plano de dados que prepara o caminho para o paradigma de redes definidas or software, comutadores de rede tornam-se dispositivos simples de encaminhamento e a lógica de controle é implementada em um controlador logicamente centralizado - embora, a princípio, fisicamente distribuído. Em SDN, o controlador é a entidade que regula o comportamento da rede. A centralização da lógica de controle em um módulo de software que é executado em um servidor padrão - o sistema operacional de rede - oferece vários benefícios. Primeiro, é mais simples e menos propenso a erros modificar as políticas de rede através de software, do que através de configurações de dispositivos de baixo nível. Em segundo lugar, um programa de controle pode reagir

automaticamente a alterações espúrias do estado da rede e, assim, manter as políticas de alto nível no lugar. Em terceiro lugar, a centralização da lógica de controle em um controlador com conhecimento global do estado da rede simplifica o desenvolvimento de funções de rede mais sofisticados. Esta capacidade de programar a rede, a fim de controlar o plano de dados subjacente é, por conseguinte, a proposta de valor crucial da Rede Definida por Software.

Na visão de Hu, Hao e Bao (2014), o plano de controle pode enviar comandos para para os planos de dados do hardware, composto por roteadores e comutadores. Este paradigma fornece uma visão de toda a rede e ajuda a fazer alterações a nível global sem uma configuração de dispositivos específica de cada unidade de equipamento físico.

Em Redes Definidas por Software, a complexidade da rede se desloca para o controlador e traz simplicidade e abstração para o operador de rede. Este se afasta da configuração manual de cada dispositivo, aproximando-se da aplicação automatizada de políticas e regras de rede. Uma Rede Definida por Software desacopla o plano de controle do plano de dados e migra aquele para um controlador de rede logicamente centralizado baseado em software. Aplicações de controle de rede mais complexas podem ser implementadas no controlador e explorar o fato de que eles estão cientes de toda a rede devido à natureza centralizada do plano de controle (LARA; RAMAMURTHY, 2014).

Conforme Dürr (2012), SDN oferece um alto grau de flexibilidade, pois uma mudança de funcionalidade consiste em reimplementar parte do software do controlador. Novos algoritmos de roteamento são facilmente implementados, sem mudanças nos switches. A mudança de algoritmo é toda feita no controlador. A implementação de controladores é feita em linguagens de alto nível, como Java ou C++, permitindo o rápido desenvolvimento de novas versões, facilitando mudanças.

Segundo Lantz, Heller e McKeown (2010), a principal conseqüência de SDN é que a funcionalidade da rede é definida depois de sua implantação, ficando a definição a cargo do operador da rede. A evolução da rede pode ocorrer à velocidade de desenvolvimento de software, permitindo novos usos para cabeçalhos de pacotes quanto a seu processamento por camadas específicas.

2.1 O Protocolo Openflow

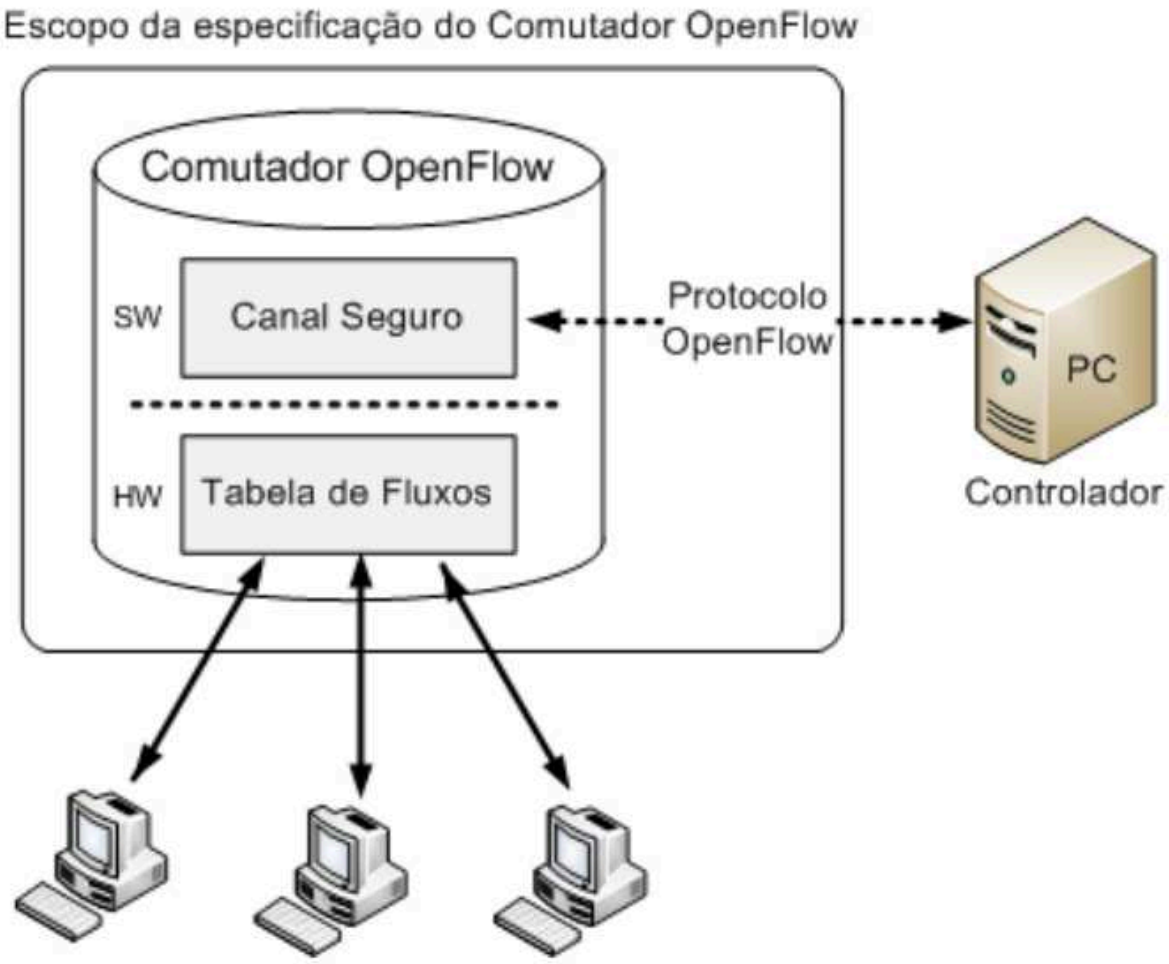
O protocolo OpenFlow é um dos protocolos existentes usados para implementação de Redes Definidas por Software. Para Benton, Camp e Small (2013), OpenFlow e outros protocolos de SDN têm gerado interesse devido ao grande de controle que eles oferecem aos desenvolvedores de software de controle de rede. Ao

criar uma interface normalizada, acessível pela rede para controlar o plano de dados de equipamentos de rede, a lógica do plano de controle pode ser movida de dispositivos de rede individuais para um controlador centralizado ou um grupo de controladores. Ao implementar a lógica de controle no controlador, mudanças de protocolo de rede e requisitos complexos de engenharia de tráfego podem ser ajustados reconfigurando, atualizando ou trocando o controlador ao invés de atualizar ou substituir o equipamento de rede.

O protocolo OpenFlow foi proposto inicialmente como uma alternativa para facilitar a inovação em ambientes de rede universitários. O OpenFlow separa a rede em dois planos: o plano de controle, responsável pela execução dos algoritmos de controle da rede, e o plano de dados, responsável pelo encaminhamento e tratamento dos pacotes de rede em si. O OpenFlow se baseia em encaminhamento por fluxos, se aproveitando do fato de que grande parte dos fabricantes de comutadores e roteadores atuais implementam uma tabela de fluxos e coleta de estatísticas diretamente nos equipamentos (COSTA et al., 2014).

De acordo com Benton, Camp e Small (2013), OpenFlow é a definição de um protocolo para comunicação com um equipamento de rede, e controle do funcionamento do seu plano de dados.

Figura 1 – Arquitetura OpenFlow



(Adi Nascimento Marcondes, 2016)

O plano de dados é controlado com o fornecimento de regras, referidas como fluxos, para os dispositivos de rede, referidos como comutadores. Cada fluxo especifica uma condição de combinação com pacotes de entrada e uma instrução a ser aplicada aos pacotes combinados. A especificação do protocolo OpenFlow tem evoluido, de forma que as operações disponíveis e critérios de correspondência tornaram-se mais complicados (por exemplo, fila de controle de qualidade de serviço e de correspondência sobre várias tabelas de fluxo), mas a idéia básica de combinar uma regra e executar uma ação continua a mesma (FEAMSTER; REXFORD; ZEGURA, 2014; KREUTZ et al., 2015).

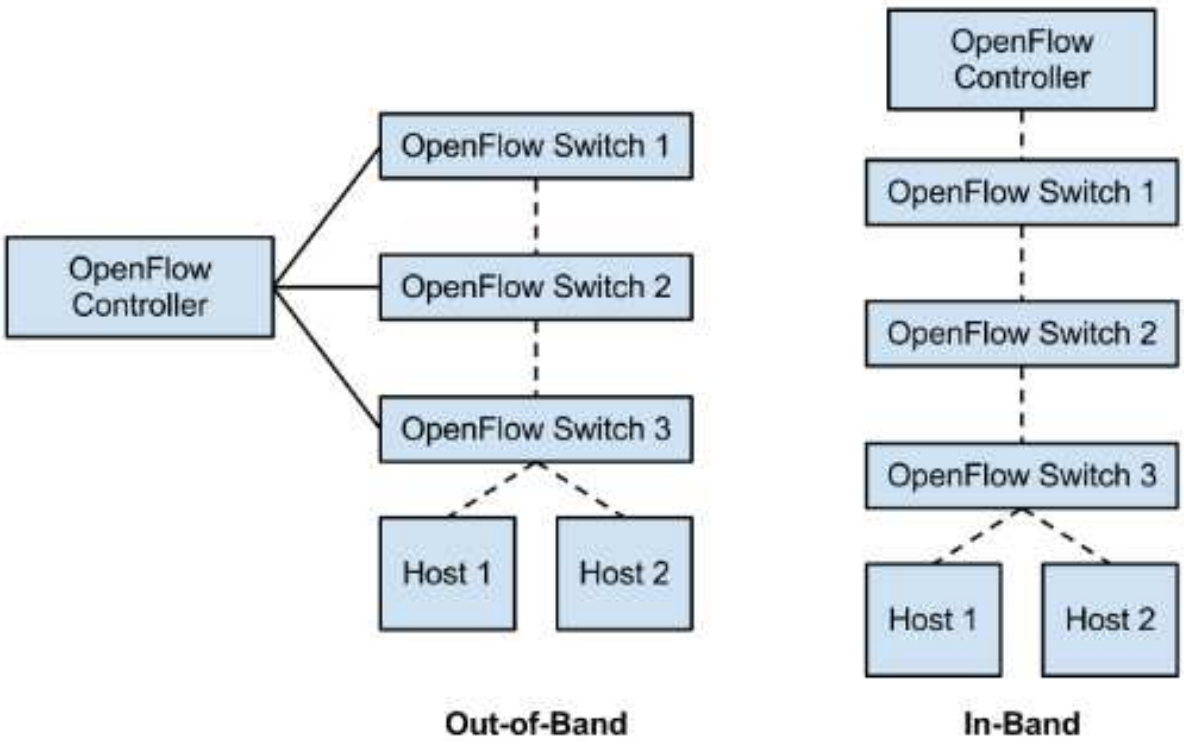
Há dois estilos gerais de criação de regra em redes OpenFlow: proativo, no qual regras são inseridas em comutadores antes que sejam necessárias; e reativo, no qual regras são inseridas em comutadores em resposta a pacotes observados pelo controlador, através de mensagens *de chegada de pacotes*. Em redes reativas, o comutador gera uma mensagem de chegada de pacotes em resposta a um pacote que

não corresponde a uma regra, a qual encapsula o pacote e envia para o controlador para que tome uma decisão. O controlador pode então reconhecer a mensagem e ignorar o pacote, ou responder com uma ação para aplicar ao pacote, juntamente com uma regra opcional para instalar na tabela de fluxos do comutador para combinar futuros pacotes (HU; HAO; BAO, 2014).

Quando a especificação é cumprida rigorosamente, as regras de fluxo para um comutador só podem ser instaladas por um controlador OpenFlow através de uma conexão TCP iniciada pelo comutador. Entretanto, alguns comutadores suportam um “modo de escuta” adicional, em que aceita conexões através de uma porta TCP configurada a partir de qualquer fonte de rede. Estas conexões iniciadas externamente também podem ser usadas para escrever regras para comutadores e ler informações a partir deles. Este modo de operação permite a fácil depuração e verificação do estado das regras, sem adição de carga ou complexidade aos controladores. No entanto, introduz uma importante vulnerabilidade porque não possui métodos de autenticação ou controle de acesso embutidos (SCOTT-HAYWARD; NATARAJAN; SEZER, 2015).

As comunicações entre o controlador e os comutadores são realizadas através de uma conexão TCP que pode, opcionalmente, ser protegida por TLS com certificados mutuamente autenticados. A conexão TCP é iniciada pelo comutador (exceto no modo de escuta não padrão mencionado acima) com uma configuração definida pelo operador que especifica o endereço IP e porta TCP do controlador (JAVID; RIAZ; RASHEED, 2014).
Conforme Hu, Hao e Bao (2014), existem dois métodos gerais de tratamento de tráfego OpenFlow entre os comutadores e o controlador, que são referidos como controle *in-band* e *out-of-band*.

Figura 2 – Controle OpenFlow Out-of-band comparado com In-Band



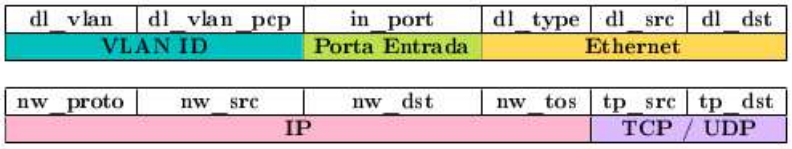
(da Costa, Victor Torres et al., 2014)

(As linhas pontilhadas representam ligações controladas por OpenFlow)

Ainda segundo Hu, Hao e Bao (2014), nas implantações de controle *out-of-band* os comutadores têm caminhos de rede para se comunicar com o controlador que não são afetados pela operação de OpenFlow. Isto requer a configuração de VLANs ou interfaces físicas distintas que não têm regras OpenFlow aplicadas a elas. Por outro lado, em configurações de controle in-band, os comutadores utilizam a rede OpenFlow tanto para transportar dados quanto para se comunicar com o controlador. A Figura 2 ilustra a diferença entre os dois métodos de controle.

As entradas na tabela de fluxos no OpenFlow são compostas por campos de cabeçalho, contadores e ações. Os campos de cabeçalho são atributos que descrevem os tipos de pacotes pertencentes àquele fluxo, ou seja, a descrição de um fluxo. A descrição de um fluxo é feita baseada em um conjunto de campos disponíveis no OpenFlow, que reúne características de várias camadas de protocolo, como pode ser observado na figura 3.

Figura 3 – Exemplo de entrada de fluxo OpenFlow



(da Costa, Victor Torres et al., 2014)

Uma descrição mais detalhada de cada campo, e sua utilização no cabeçalho OpenFlow, é representada na figura 4.

Figura 4 – Descrição detalhada de atributos de fluxo OpenFlow

Campo	Tamanho (*Bits*)	Aplicável em	Notas
Porta de entrada (in_port)	(Dependente da implementação)	Todos os pacotes	Representação numérica da porta de entrada, começando em 1
Endereço Ethernet de origem (dl_src)	48	Todos os pacotes	
Endereço Ethernet de destino (dl_dst)	48	Todos os pacotes	
Tipo Ethernet (dl_type)	16	Todos os pacotes	Número que define o tipo de cabeçalho Ethernet do pacote. Esse número pode indicar, por exemplo, a presença da etiqueta VLAN (padrão 802.1Q)
Identificador VLAN (dl_vlan)	12	Pacotes com tipo Ethernet 0x8100	
Prioridade VLAN (dl_vlan_pcp)	3	Pacotes com tipo Ethernet 0x8100	Campo VLAN PCP
Endereço IP de origem (nw_src)	32	Pacotes IP e ARP	Máscara de sub-rede IP pode ser utilizada
Endereço IP de destino (nw_dst)	32	Pacotes IP e ARP	Máscara de sub-rede IP pode ser utilizada
Protocolo IP (nw_proto)	8	Pacotes IP, IP sobre Ethernet e ARP	
Bits IP ToS (nw_tos)	6	Pacotes IP	
Porta de Transporte de origem (tp_src)	16	Pacotes TCP, UDP e ICMP	Para o caso de pacotes ICMP, é interpretado como o tipo ICMP
Porta de Transporte de destino (tp_dst)	16	Pacotes TCP, UDP e ICMP	Para o caso de pacotes ICMP, é interpretado como o código ICMP

(da Costa, Victor Torres et al., 2014)

Assim, um fluxo poderia ser descrito como, por exemplo, todos os pacotes vindos da porta 1 do comutador e com IP de origem 146.164.69.2. Existe também o chamado valor curinga, que quando aplicado a um campo de cabeçalho, define que aquele campo assume todos os valores possíveis para aquele campo simultaneamente (COSTA et al., 2014).

Cada fluxo também 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. Os contadores utilizados no OpenFlow são detalhados na figura 5.

Figura 5 – Contadores estatísticos de fluxo OpenFlow

Contador	Tamanho (*Bits*)
Por Tabela	
Entradas Ativas	32
Buscas Efetuadas	64
Buscas bem-sucedidas	64
Por Fluxo	
Pacotes Recebidos	64
Bytes Recebidos	64
Duração (s)	32
Duração (ns)	32
Por Porta	
Pacotes Recebidos	64
Pacotes Transmitidos	64
Bytes Recebidos	64
Bytes Transmitidos	64
Perdas de Recepção	64
Perdas de Transmissão	64
Erros de Recepção	64
Erros de Transmissão	64
Erros de Alinhamento de Quadro na Recepção	64
Erros de *Overrun* na Recepção	64
Colisões	64
Por Fila	
Pacotes para Transmissão	64
Bytes para Transmissão	64
Erros de *Overrun* na Transmissão	64

(da Costa, Victor Torres et al., 2014)

Além dos contadores, cada fluxo também possui um conjunto de ações definidas pelo controlador para serem aplicadas aos pacotes daquele fluxo. Dentre os diferentes tipos de ações, existem ações para fazer o encaminhamento de um pacote em uma determinada porta de saída, descartar o pacote ou modificar um determinado campo do cabeçalho do pacote. Exemplos de ações OpenFlow disponíveis podem ser vistas na figura 6.

Figura 6 – Ações OpenFlow

Ação	Dados Associados
Encaminhar(x)	x (depende da implementação): valor indicando em qual porta o pacote deve ser encaminhado
Enfileirar(x)	x (depende da implementação): valor indicando para qual fila o pacote deve ser encaminhado
Decartar	-
Mudar VLAN ID	12 bits: Valor a substituir o VLAN ID atual
Mudar prioridade VLAN	3 bits: Valor a substituir o VLAN PCP atual
Retirar cabeçalho VLAN	-
Mudar endereço Ethernet de origem	48 bits: Valor a substituir o endereço Ethernet de origem atual
Mudar endereço Ethernet de destino	48 bits: Valor a substituir o endereço Ethernet de destino atual
Mudar endereço IP de origem	32 bits: Valor a substituir o endereço IP de origem atual
Mudar endereço IP de destino	32 bits: Valor a substituir o endereço IP de destino atual
Mudar bits IP ToS	6 bits: Valor a substituir o IP ToS atual
Mudar porta de Transporte de origem	16 bits: Valor a substituir a porta TCP ou UDP de origem atual
Mudar porta de Transporte de destino	16 bits: Valor a substituir a porta TCP ou UDP de destino atual

(da Costa, Victor Torres et al., 2014)

Um exemplo de tabela de ações do OpenFlow é apresentado na figura 7.

Figura 7 – Tabela de Ações OpenFlow

Características	Ações (*Actions*)
Pacotes TCP; IP de Origem 192.168.1.1	Encaminhar na Porta 1
MAC de Origem 11 : 22 : 33 : 44 : 55 : 66; ARP	Descartar
Etiqueta VLAN ID 100	Mudar Etiqueta VLAN ID para 200
Porta de Entrada 2; Todos os IPs de Destino (curinga)	Enviar para o Controlador

(da Costa, Victor Torres et al., 2014)

2.2 O Simulador de Redes Mininet

Mininet é um simulador de Redes que foi desenvolvido por Bob Lantz e Brian O'Connor como uma bancada de testes de rede. De acordo com Mininet (2016), Mininet é um sistema de prototipagem rápida, cujo principal objetivo é o desenvolvimento rápido de aplicações de controladores OpenFlow. É um emulador de

rede que cria uma rede de estações de trabalho virtuais, comutadores, controladores e conexões. Estações de trabalho virtuais Mininet executam software de rede padrão do Linux, e seus comutadores suportam OpenFlow para o encaminhamento personalizado altamente flexível disponibilizado por SDN. Mininet dá suporte à virtualização de uma rede experimental completa em um computador pessoal.

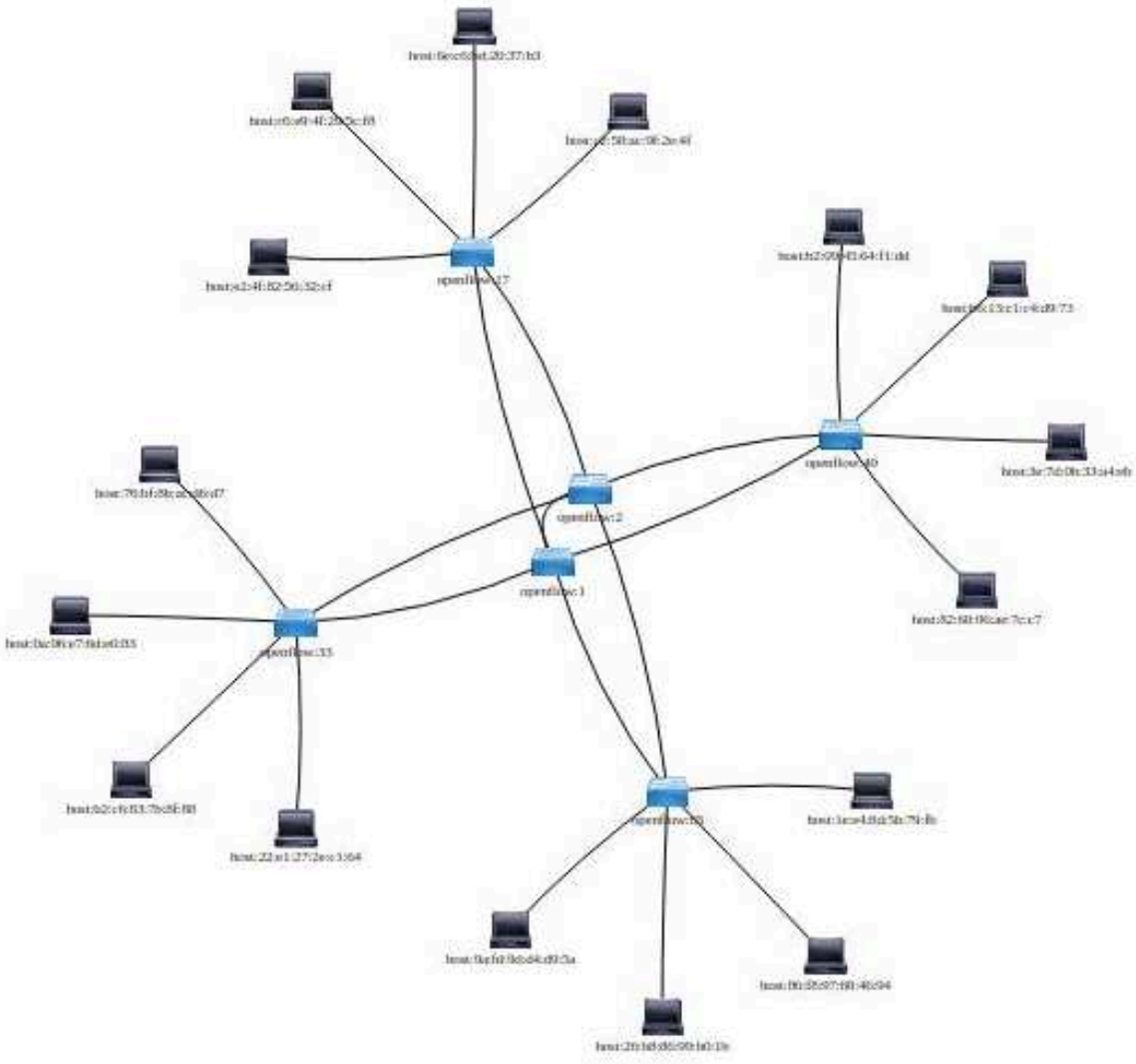
Mininet permite testes com topologias complexas, sem a necessidade de conexão a uma rede física. Inclui uma interface de linha de comando com suporte a múltiplas topologias e a OpenFlow, para testes de depuração ou execução em qualquer ponto da rede. Suporta topologias personalizadas arbitrárias, e inclui um conjunto básico de topologias parametrizadas. É utilizável imediatamente, sem programação, mas também fornece uma interface de programação simples e extensível em linguagem Python para a criação de redes e experimentação. Mininet fornece uma maneira de obter o comportamento correto do sistema (e, na medida suportada pelo hardware, o desempenho) e de experimentar com topologias. Redes Mininet executam código real, incluindo aplicações de rede Unix/Linux padrão, bem como o kernel e pilha de rede Linux real, incluindo quaisquer extensões de kernel disponíveis (OLIVEIRA et al., 2014).

O código desenvolvido e testado em Mininet, para um controlador OpenFlow, comutador modificado, ou estação de trabalho, pode passar para um sistema do mundo real com alterações mínimas, para o teste real, avaliação de desempenho e implementação. Isto significa que um projeto que trabalha em Mininet geralmente pode ir diretamente para os comutadores físicos (MININET, 2016; ANTONENKO; SMELYANSKIY, 2013).

A maioria dos sistemas operacionais virtualiza recursos de computação usando abstração de processos. Mininet usa virtualização baseada em processos para executar muitas estações de trabalho e comutadores em um único kernel do sistema operacional. Mininet pode criar comutadores OpenFlow de kernel ou de espaço do usuário, controladores para controlar os comutadores, e as estações de trabalho para se comunicar através da rede simulada. Mininet conecta comutadores e estações de trabalho usando pares ethernet. (MININET, 2016).

Mininet é capaz de emular redes complexas com várias estações de trabalho e comutadores em um computador pessoal, como na figura abaixo:

Figura 8 – Topologia emulada com uso de Mininet

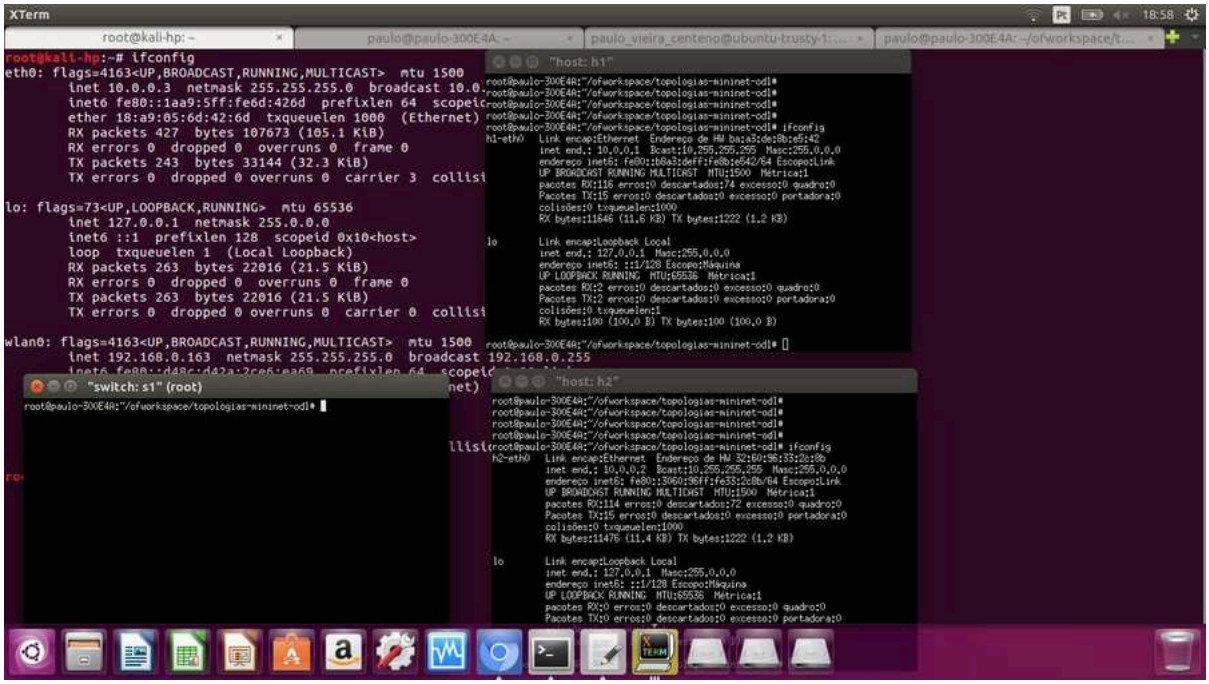


Fonte: Elaborada pelo autor

O Apêndice A apresenta o script que gera essa topologia.

Cada nó da rede emulada com Mininet pode ser acessado usando um terminal de linha de comando, assim como cada comutador. Dessa forma, é possível iniciar serviços e aplicativos em qualquer das estações de trabalho que compõem uma rede Mininet:

Figura 9 – Mininet exibindo Terminais de linha de comando



Fonte: Elaborada pelo autor

2.3 Comutador OpenFlow Open vSwitch

Open vSwitch é um comutador virtual com suporte a OpenFlow. É projetado para permitir a automatização de grandes redes através da extensão programática, suportando ainda interfaces e protocolos de gerenciamento padrão como, por exemplo, NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP e 802.1ag. Além disso, pode suportar a distribuição através de múltiplos servidores físicos (FOUNDATION, 2016a).

2.4 Controladores OpenFlow

Há diversas opções de controlador OpenFlow disponíveis, a maioria deles de uso gratuito. Segue abaixo a descrição sucinta das características de alguns dele.

2.4.1 FloodLight

Floodlight é um controlador SDN capaz de interoperar com comutadores físicos, virtuais e pontos de acesso com suporte ao protocolo OpenFlow. Oferece um sistema de carregamento de módulos extensível por linguagem java. Possúi dependências mínimas. Pode lidar com redes mistas OpenFlow e não OpenFlow, sendo capaz também de gerenciar várias sub-redes de comutadores físicos OpenFlow (Big Switch Networks, 2016).

2.4.2 OpenDayLight

OpenDaylight é um controlador SDN de código-fonte aberto. Foi criado por uma associação de empresas da indústria de equipamentos e programas de rede. Sua proposta é entregar redes programáveis interoperáveis para os provedores de serviços, empresas, universidades e uma variedade de organizações (FOUNDATION, 2016c).

2.4.3 Open vSwitch Controller

O controlador Open vSwitch é um controlador virtual com suporte ao protocolo OpenFlow. Ele é projetado para permitir a automatização de redes através da extensão programática de suas funcionalidades (FOUNDATION, 2016b).

2.4.4 POX

POX é um controlador OpenFlow escrito em Python, característica que permite a inclusão de módulos registrados como ações executadas em determinados eventos. Exemplos desses eventos são 'Início de conexão', que é acionado quando um comutador OpenFlow se conecta ao controlador, e 'Chegada de pacote', que é acionado quando o comutador recebe uma mensagem de pacote de entrada. Na instalação padrão do POX já existem alguns módulos disponíveis, incluindo módulos que implementam o comportamento dos comutadores Ethernet tradicionais, ou mesmo hubs. Usando as bibliotecas Python POX é possível criar novos módulos, ou modificar os já existentes, para que o comutador OpenFlow se comporte da maneira desejada (NOXRepo Community, 2016).

2.4.5 Ryu

O controlador SDN Ryu é um controlador de código fonte aberto, destinado a aumentar a agilidade da rede, tornando mais fácil de gerir e adaptar como o tráfego é tratado. Ele oferece componentes com interfaces de programação de aplicações bem definidas, para os desenvolvedores criarem novas aplicações de gestão de rede e controle (COMUNNITY, 2016).

2.5 Considerações

Existe um ecossistema em torno de SDN que proporciona a possibilidade de simular uma rede completa, desde estações de trabalho a comutadores e controladores OpenFlow.

Também há disponíveis várias implementações de controlador OpenFlow, escritas em diferentes linguagens de programação, como C++, Java e Python. Algumas são mais sofisticadas; outras, mais enxutas. Em geral, há módulos prontos

com comportamento básico, equivalente a comutadores ethernet padrão, que podem ser alterados para se comportar da maneira desejada pelo desenvolvedor.

Essas facilidades viabilizam a simulação e prototipação de redes com topologias das mais diversas, a um custo computacional baixo, podendo ser executada em qualquer computador pessoal convencional. Portanto, oferece um ambiente e infraestrutura propícios sos testes propostos neste trabalho.

3 Vulnerabilidades em Redes Definidas por Software baseadas em protocolo OpenFlow

Segundo Mattos e Duarte (2014), Souza e Nobre (2016), Benton, Camp e Small (2013) a adoção do protocolo OpenFlow como forma de implementar Redes Definidas por Software trouxe um nível maior de abstração na administração de redes. Porém, um efeito colateral dessas facilidades foi a potencialização no risco de ataques maliciosos a essas redes usando técnicas de ataques bastante conhecidas, tirando proveito de características oriundas de SDN. A seguir são descritas algumas das vulnerabilidades apresentadas pelo protocolo OpenFlow.

3.1 Falta de autenticação de origem

O protocolo OpenFlow, apesar de largamente utilizado em SDN, não possui um mecanismo nativo para autenticação segura da origem. Por padrão, as estações finais se autenticam a uma rede OpenFlow através da validação de seus endereços MAC (Media Access Control) ou IP (Internet Protocol), com base na lógica de comutação de pacotes definida pelo administrador da topologia (SOUZA; NOBRE, 2016).

3.2 Canal de comunicação inseguro entre controlador e comutadores

O canal de comunicação entre o controlador e seus comutadores, na especificação inicial do OpenFlow, previa o uso obrigatório de proteção TLS (HELLER, 2009). Porém, em versões posteriores, esse requisito passou a ser opcional (FOUNDATION, 2012). A falta de suporte a TLS cria oportunidades para atacantes se infiltrarem em redes OpenFlow sem serem detectados.

Embora isso possa ser inviável em algumas redes fisicamente seguras, como centros de dados, onde o acesso aos comutadores é difícil para os atacantes, torna-se uma grande preocupação em campus e implantações de escritórios remotos nos quais comutadores são implantados em locais com pouco ou nenhum monitoramento.

Também torna e mais fácil o acesso sem detecção, por exemplo, de armários, escritórios, caixas de conexão ao ar livre, etc (BENTON; CAMP; SMALL, 2013).

3.3 Vulnerabilidade de componentes

Ausência de confiança entre componentes da rede compromete uma Rede Definida por Software, pois as aplicações executadas sobre o controlador podem ter

comportamentos maliciosos. Nesse caso, o controlador deve ser capaz de identificar quais são as aplicações confiáveis e quais são maliciosas (MATTOS; DUARTE, 2014).

Vulnerabilidade de componentes não é um desafio de segurança exclusivo desse novo paradigma de rede, mas torna-se mais crítico, pois uma vulnerabilidade em um nó controlador torna toda a rede vulnerável. Existem três possıveis fontes de vulnerabilidade de componentes: comutadores, controlador e estações de gerenciamento. Uma vulnerabilidade em um comutador pode permitir que um atacante que obtenha acesso a um comutador execute um ataque contra o plano de controle, a exemplo da falsificação de mensagens de outros comutadores para esgotar os recursos do controlador. Uma vulnerabilidade no controlador pode permitir que um atacante altere o plano de controle ou até mesmo execute uma nova aplicação de controle da rede. Uma vulnerabilidade em uma estação de gerenciamento permite que o atacante faça configurações no plano de controle diferentes das corretas (MATTOS; DUARTE, 2014).

Os riscos decorrentes de um ataque homem-no-meio bem sucedido em uma rede OpenFlow gerenciada com tráfego de controle junto com tráfego de dados são indiscutivelmente piores do que uma rede convencional, devido à capacidade do atacante reconfigurar imediatamente todas as opções de comutadores sob controle do controlador dessa rede. Em uma rede normal, o invasor terá que esperar até que o operador efetue a autenticação na interface de gerenciamento de cada comutador usando um protocolo inseguro, como por exemplo Telnet ou SNMPv2, para capturar as credenciais. No entanto, devido à conectividade constante e falta de autenticação no canal de controle TCP OpenFlow em texto simples, um atacante pode imediatamente tomar o controle completo de quaisquer comutadores e executar ataques de espionagem numa granulação muito fina de forma que seriam difíceis de detectar (BENTON; CAMP; SMALL, 2013).

3.4 Risco de negação de serviço

Esse novo paradigma apresenta algumas limitações quanto à segurança da rede, pois um componente com comportamento malicioso pode comprometer o funcionamento de toda a rede realizando um ataque de negação de serviço no controlador (MATTOS; DUARTE, 2014).

A inundação de pacotes é um dos ataques mais comuns que negam serviços de rede. O ataque de inundação efetua muitas transmissões de pacotes levando ao esgotamento dos recursos da rede. Este ataque é de difícil detecção em redes OpenFlow pela falta da relação entre a autenticação dos dispositivos e a tabela de fluxo presente nos comutadores (MARCONDES, 2016).

De acordo com Mattos e Duarte (2014), Scott-Hayward, Natarajan e Sezer (2015) a negação de serviço pode ocorrer tanto no plano de dados quanto no plano de controle. No plano de dados, uma estação maliciosa que gere fluxos falsos pode esgotar tanto os recursos de banda, quanto os recursos de memória, ou tabela de fluxos dos comutadores da rede. A negação de serviço no plano de controle pode ser causada em dois pontos distintos da rede: no controlador e na comunicação do controlador com os comutadores. É possıvel esgotar a capacidade de processamento do controlador de rede enviando uma grande quantidade de pacotes com diferentes cabeçalhos. Todo pacote é analisado e um pacote com cabeçalho que não corresponde a nenhum fluxo já definido deve ser enviado ao controlador de rede.

Assim, em um cenário em que um comutador envia uma quantidade atıpica de novos cabeçalhos de pacotes para o controlador, este pode ter seus recursos de processamento exauridos e não ser capaz de responder a pedidos de novos fluxos em tempo hábil. Da mesma forma, a negação de serviço pode ser alcançada quando o enlace de conexão entre o controlador e os comutadores na rede é intencionalmente congestionado. Caso não haja redundância ou banda suficiente no enlace que conecta os comutadores ao controlador, um comutador malicioso pode gerar tráfego suficiente para sobrecarregar esse enlace e, em consequencia, impedir a comunicação do controlador com os demais comutadores (MATTOS; DUARTE, 2014).

3.5 Considerações

Apesar de ser uma nova arquitetura de redes, SDN sofre com riscos de segurança clássicos, expostos justamente pelas facilidades oferecidas pelo novo desenho proposto por seus projetistas.

Tanto o plano de controle quanto o plano de dados estão sujeitos a vulnerabilidades expostas por SDN e o protocolo OpenFlow.

A seguir, são enumeradas as vulnerabilidades sugeridas e os planos que estão sujetos às mesmas:

Tabela 1 – Vulnerabilidades /plano exposto

Vulnerabilidade	Plano de Controle	Plano de Dados
Falta de autenticação de origem	Sim	Sim

Vulnerabilidade	Plano de Controle	Plano de Dados
Canal de comunicação inseguro	Sim	
Vulnerabilidade de componentes	Sim	
Negação de Serviço	Sim	Sim

Fonte: Elaborada pelo autor

4 Ambiente e resultados experimentais

As redes de computadores atuais apresentam restrições quanto à escalabilidade e administração que estão levando a infraestrutura existente a seu limite. Rede definida por software (SDN) surge como um novo paradigma de implantação e administração de redes, oferecendo uma flexibilidade não disponível até então.

Este trabalho se propõe a fazer uma análise e demonstrar problemas de segurança apresentados pela implementação de Redes Definidas por Software com o uso de protocolo OpenFlow, a partir de experiências práticas. Para viabilizar tal demonstração, usa um ambiente de simulação realístico, o Mininet, que permite emular componentes típicos de arquiteturas SDN. Segundo Mininet (2016) Mininet é uma excelente maneira de desenvolver, compartilhar experimentar Sistemas SDN e OpenFlow.

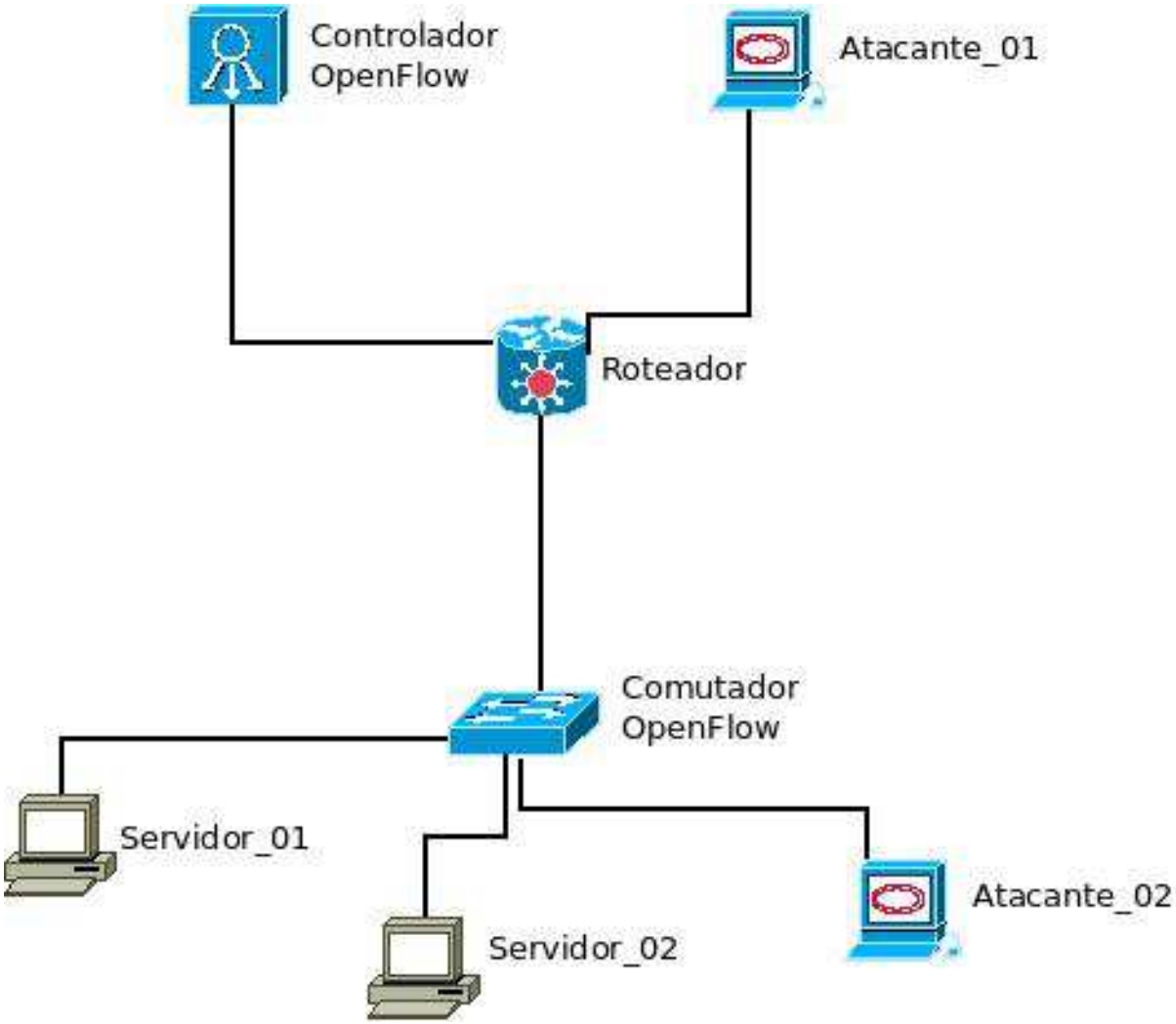
Nesse ambiente de simulação, é criada uma configuração que permite executar essa demonstração empírica.

4.1 Topologia de rede

A topologia de rede adotada nos testes é modelada de maneira bem simplificada. Os planos de controle e de dados ficam separados em duas subredes distintas, e em cada uma dessas subredes há uma máquina maliciosa, permitindo a simulação de ataques. O uso do Mininet permitiria criar redes complexas, com múitas estações de trabalho, comutadores e controladores, mas foge ao objetivo do trabalho. A idéia foi adotar um desenho de rede simples para evidenciar as situações nas simulações de ataque propostas.

As instalações e ajustes variam em torno da seguinte infraestrutura geral:

Figura 10 – Topologia da rede adotada no ambiente de simulação



Fonte: Elaborada pelo autor

Máquina	IP
Comutador OpenFlow	192.168.10.1
Controlador	192.168.10.2
Atacante_01	192.168.10.3
Servidor_01	10.0.0.1
Servidor_02	10.0.0.2
Atacante_02	10.0.0.3

A topologia consiste de:

- Duas subredes: a subrede de controle OpenFlow 192.168.10.0/24 e a rede de clientes SDN 10.0.0.0/24 - controle *out-of-band*
- Rede comutada: implementação de comutador Open vSwitch, executada virtualmente pelo emulador Mininet.

- Um controlador OpenFlow, usado para administrar o comutador OpenFlow. A máquina usada foi uma instância no serviço de nuvem do Google
- Estação de trabalho Atacante_01: máquina física que age como uma máquina atacante na subrede de controle OpenFlow. Esta é uma máquina física com duas interfaces de rede:, uma é conectada à rede virtual simulada pelo Mininet, e a outra é usada para administração e execução das ferramentas.
- Estação de trabalho Servidor_01: atua como servidor legítimo na rede de clientes SDN. Instânciada virtualmente pelo emulador Mininet
- Estação de trabalho Servidor_02: atua como servidor legítimo na rede de clientes SDN. Instânciada virtualmente pelo emulador Mininet
- Estação de trabalho Atacante_02: máquina física que age como uma máquina atacante na subrede de clientes OpenFlow. Esta é uma máquina física com duas interfaces de rede:, uma é conectada à rede virtual simulada pelo Mininet, e a outra é usada para administração e execução das ferramentas.

O Controlador recebeu Ubuntu Linux LTS 14.04. As máquinas Servidor_01 e Servidor_02 usaram Ubuntu Linux LTS 16.04. As máquinas Atacante_01 e Atacante_02 receberam Kali Linux versão 2016.2. Kali Linux fornece todas as ferramentas necessárias em uma plataforma otimizada para testes de penetração.

Um script Python com extensões Mininet se encarrega de instanciar a topologia usada (apêndice B). Uma interface real da estação de trabalho que executa o Mininet é associada à rede emulada, permitindo conectar outra máquna real a esta rede.

4.2 Configuração do ambiente misto - virtual/físico

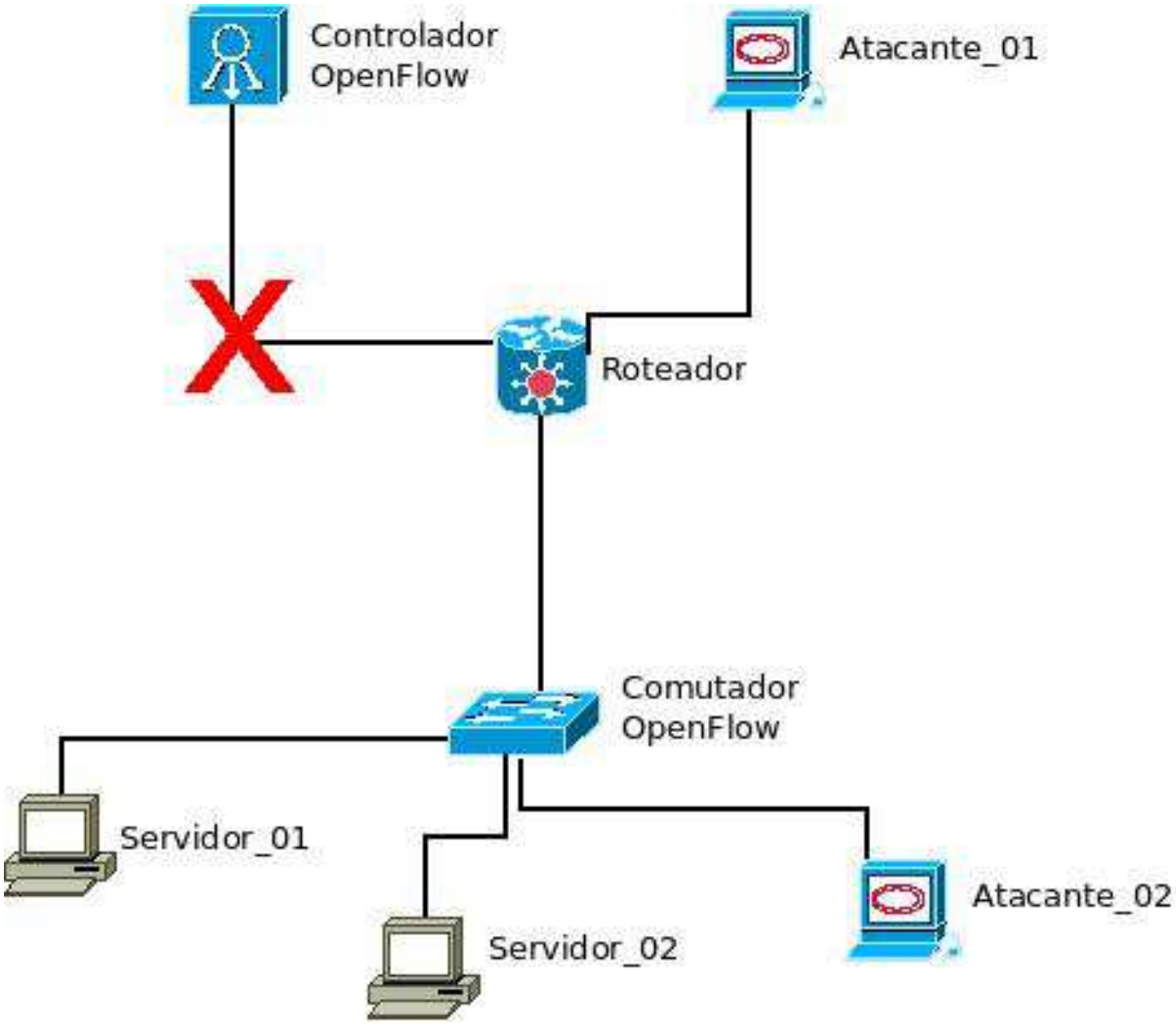
Uma configuração mista virtual/física foi criada, para permitir a execução de testes sem um comutador OpenFlow físico. Como alguns comutadores OpenFlow comerciais executam o comutador Open vSwitch, que é um comutador virtual com recursos de OpenFlow, usado nesta configuração, os resultados obtidos nos testes executados na configuração virtualizada podem aplicar-se a essas opções físicas.

A configuração de testes inclui um notebook com processador i5 2.6Ghz e 4GB de memória RAM para execução do simulador de redes Mininet; outros dois notebooks com processador Atom 1.5Ghz e 2MB de memória foram usados para o papel de estações de trabalho maliciosas, as máquinas Atacante_01 e Atacante 02; e uma instância virtual de Ubuntu 14.04LTS na nuvem do Google, para a execução dos diferentes Controladores OpenFlow. As máquinas Servidor_01 e Servidor_02 foram instanciadas no Mininet, assim como o comutador Open vSwitch versão 2.6.0.

4.3 Interrupção do canal de comunicações

O canal de comunicação entre o comutador eo controlador é um dos pontos mais importantes na infraestrutura OpenFlow. O objetivo deste cenário é observar como a implementação age quando a conexão entre o comutador eo controlador é interrompida.

Figura 11 – Interrupção do canal de comunicações controlador/comutador



Fonte: Elaborada pelo autor

De acordo com as especificação do protocolo OpenFlow, quando a conexão entre o controlador e o comutador é interrompida, este deve entrar imediatamente em ‘modo de falha seguro’ ou ‘modo de falha independente’. No modo seguro, o comutador dispensa quaisquer pacotes que sejam destinados para o controlador e quaisquer fluxos que já estão na tabela de fluxo continuam a expirar de acordo com seus tempos de espera. No modo de falha autônomo, o comutador OpenFlow adota o comportamento de um comutador legado Ethernet ou roteador.

Para estes testes foram utilizados os seguintes implementações de Controlador openFlow: OpenDayLIght, POX e FloodLight.

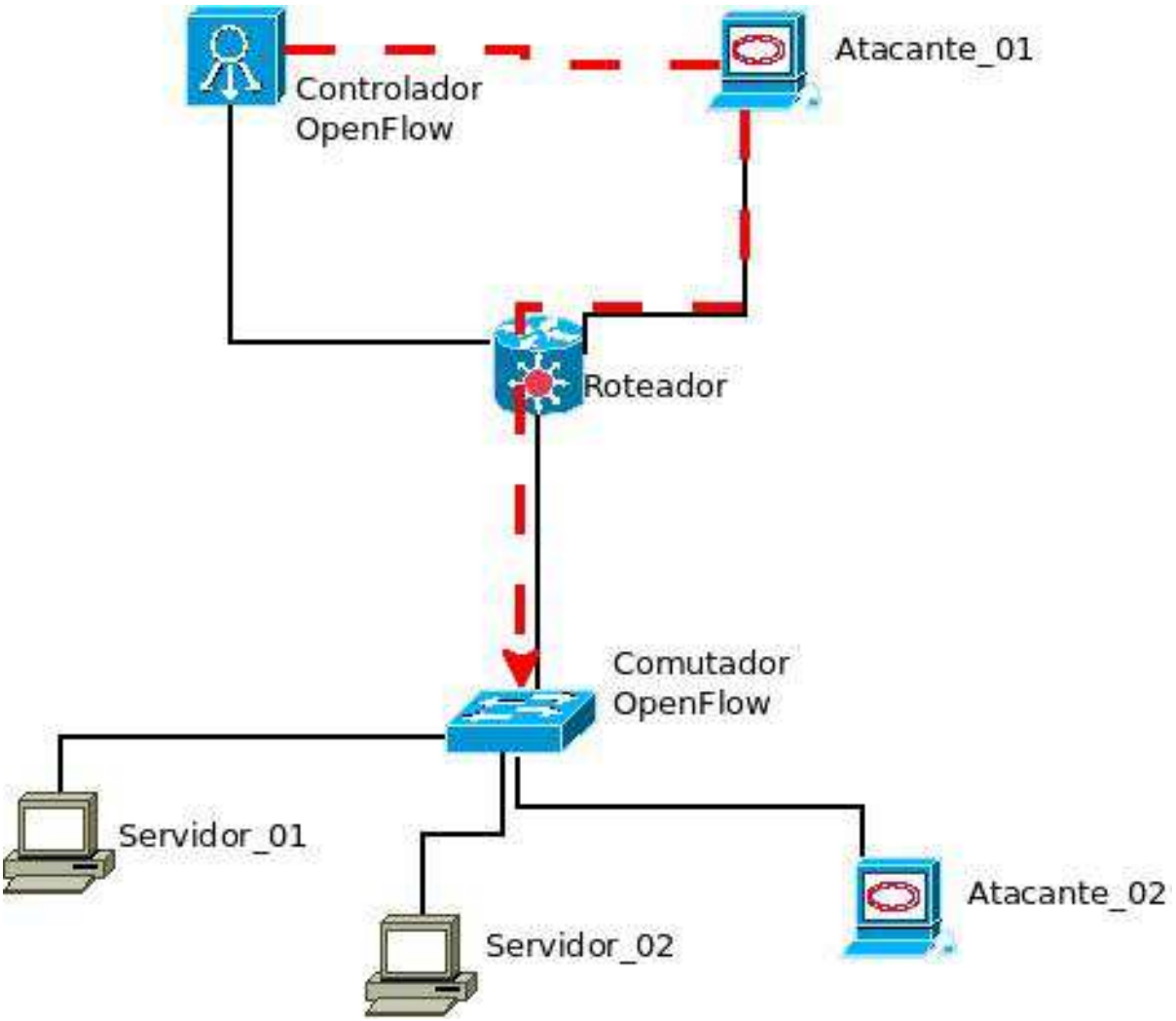
Foi utilizado o comutador virtual OpenFlow Open vSwitch.

4.3.1 Ataque arpspoof

No cenário de interrupção do canal de controle entre controlador e comutador, a rede atacada é a subrede de controle OpenFlow. A ferramenta usada para o ataque foi o arpspoof, pertencente ao pacote dsniff. Arpspoof é capaz de redirecionar pacotes de uma estação de destino (ou todas as estações) na rede local destinados a outra estação na rede local forjando respostas ARP (IRONGEEK, 2016).

A ferramenta arpspoof foi executada na Máquina Atacante_01 para fazer efetivamente um ataque homem-no-meio por envenenamento da tabela ARP do controlador e do comutador OpenFlow.

Figura 12 – Ataque arpspoof



O comando executado foi:

arpspoof -i eth0 -t 192.168.10.1 192.168.10.2

O IP forwarding foi desativado na máquina Atacante_01 para que todo o tráfego termine lá.

O comando usado com essa finalidade foi:

echo 0 > proc/sys/net/ipv4/ip_forward

O comportamento padrão do Open vSwitch é entrar no *'modo de falha independente'*. Para destacar algum comportamento indesejado quando o comutador entra neste modo, como resultado de configuração pobre, o módulo padrão *l2_learning* que vem com POX foi manipulado para descartar os pacotes vindos de Atacante_02 (apêndice C). Desta forma, foram criadas duas redes logicamente separadas. Assumiu-se que as máquinas Servidor_01 e Servidor_02 pertencem a uma rede, e a máquina Atacante_02 pertence a uma rede distinta, que não é capaz de comunicar com a primeira rede. Foi iniciado um ping de Atacante_02 ao Servidor_01 e do Servidor_02 ao Servidor_01 e depois de algum tempo a conexão entre o controlador e o comutador OpenFlow foi interrompida.

PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.3 icmp_seq=30 Destination Host Unreachable
. . .
From 10.0.0.3 icmp_seq=119 Destination Host Unreachable
64 bytes from 10.0.0.1: icmp_req=120 ttl=64 time=2061 ms
64 bytes from 10.0.0.1: icmp_req=121 ttl=64 time=650 ms
64 bytes from 10.0.0.1: icmp_req=122 ttl=64 time=77.5 ms
64 bytes from 10.0.0.1: icmp_req=123 ttl=64 time=0.132 ms
64 bytes from 10.0.0.1: icmp_req=124 ttl=64 time=0.217 ms

Pode-se afirmar com segurança que neste cenário a rede de produção pode ser afetada pela comunicação indesejada.

O modo de falha padrão do Open vSwitch foi alterado para *'modo de falha seguro'* de forma a verificar a sua implementação. Os resultados obtidos ficaram de acordo com a especificação do protocolo OpenFlow. Os pacotes pararam de ser enviados para o controlador e os fluxos existentes na tabela de fluxo expiraram de acordo com seus tempos limite de expiração.

You et al. (2014) prevêem ataques semelhantes ao descrito aqui, dedicando atenção a formas de mitigá-los.

4.4 Negação de serviço da tabela de fluxos do comutador

A execução do ataque de negação de serviço pode ser feita preenchendo completamente a tabela de fluxos do comutador. Teoricamente, se o atacante preencher a tabela de fluxo com fluxos que não correspondam ao tráfego real da rede, então qualquer novo cliente que queira usar a rede não será capaz de fazer isso. O controlador não conseguirá instalar quaisquer novos fluxos e o cliente terá efetivamente negado o uso da rede.

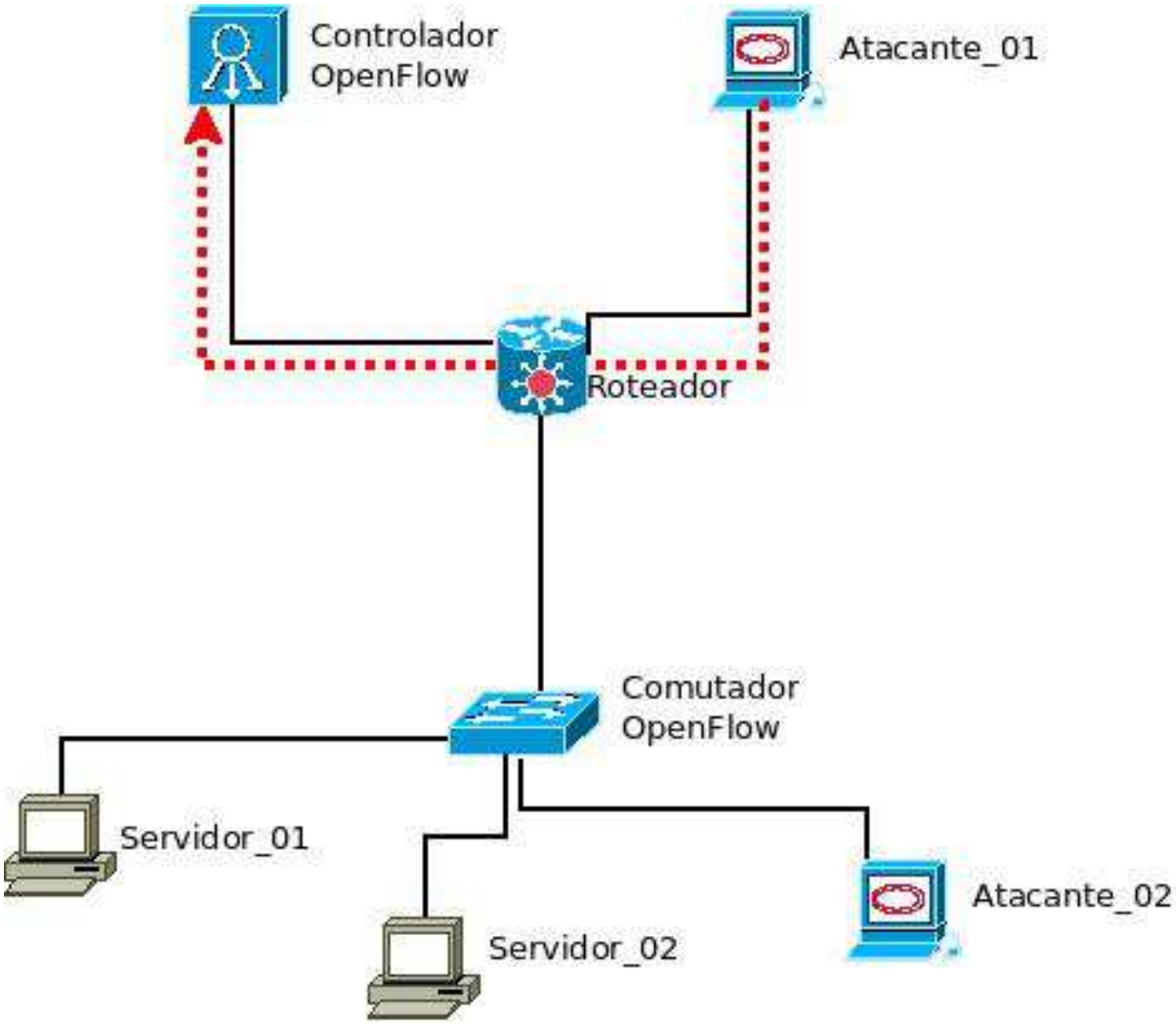
O sucesso deste ataque depende de dois parâmetros da Rede Definida por Software. Um é o tempo de validade dos fluxos que são instalados nos comutadores, configurado pelo administrador da rede. Se o controlador está configurado para instalar fluxos sem tempo limite de expiração (fluxos permanentes), então o ataque é fácilmente executado e o comutador terá uma tabela de fluxo cheia de fluxos falsos, não importando quanto tempo for necessário para o ataque.

Porém, se o controlador instala no comutador fluxos com um tempo limite curto, então o ataque fica mais difícil, pois é preciso preencher a tabela de fluxos antes que os fluxos comecem a expirar. Este é o segundo parâmetro que influencia no sucesso do ataque: a taxa de estabelecimento de fluxos. No caso dos fluxos permanentes, não há preocupação com essa taxa nem com o tempo tempo necessário para encher a tabela de fluxos. Mas quando há um tempo limite definido e os fluxos expiram após 60 ou 30 segundos, por exemplo, é preciso uma taxa bastante alta de inserção de novos fluxos. Nessa situação, mesmo que o comutador descarte os fluxos que tenham expirado, os fluxos recém inseridos vão compensar essa perda.

4.4.1 Ataque partindo de usuário malicioso no plano de controle SDN

Este ataque teve como alvo o plano de controle SDN, que conecta o controlador OpenFlow com o comutador OpenFlow.

Figura 13 – Negação de serviço - rede de controle



Fonte: Elaborada pelo autor

Esse cenário simula uma vulnerabilidade no controlador, permitindo que um atacante altere o plano de controle fazendo configurações diferentes das corretas. Foi implementado através da criação de um módulo para o controlador POX (apêndice D). Assim que uma conexão com o comutador é estabelecida, POX gera e instala modificações de fluxo para o comutador, a fim de preencher a tabela de fluxos. Este módulo registra uma função que é executada em um evento de início de conexão. A função registrada gera endereços MAC, de origem e de destino e endereços IP, novamente de origem e de destino assim como portas, que serão utilizados para criar um pacote de modificação de fluxos completo. A adição de fluxos se deu na mesma velocidade que o controlador foi capaz de gerá-los.

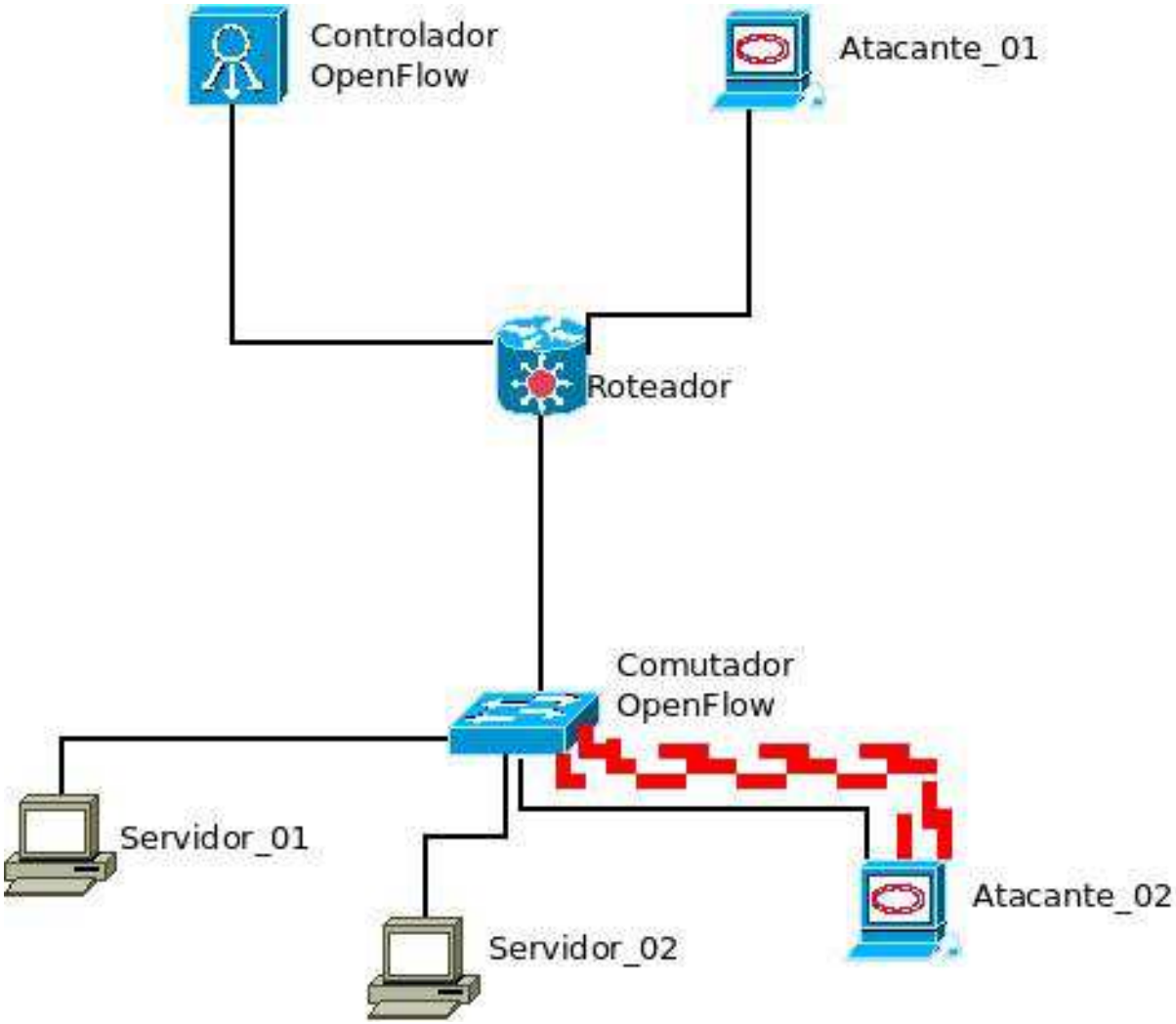
Essa simulação de ataque partindo da rede de controle OpenFlow confirma Mattos e Duarte (2014) , que o descreveu como uma ameaça de segurança em Redes Definidas por Software.

Foi possível esgotar os recursos do comutador, até que se tornasse incapaz de adicionar novos fluxos em sua tabela de fluxos, por falta de memória.

4.4.2 Ataque partindo de usuário malicioso no plano de dados SDN

O ataque aqui descrito acontece no plano de dados SDN. Foi usada a ferramenta Scapy. Conforme Bionde (2016) Scapy é um poderoso programa de manipulação interativa de pacotes, capaz de forjar ou decodificar pacotes de um grande número de protocolos, enviá-los, capturá-los, podendo lidar com varredura de portas, sondagem, testes de unidade, ataques e descoberta de redes. Usando esta ferramenta na máquina Atacante_02, foi possível enviar uma grande quantidade de pacotes ICMP com endereços IP diferentes, obrigando o controlador a instalar novos fluxos a cada solicitação de eco diferente recebida, pois os pacotes de entrada não correspondem a nenhum dos fluxos existentes no comutador. Foi possível preencher a tabela de fluxos do comutador após certo tempo executando o ataque. Seus recursos foram esgotados. O mesmo ficou incapaz de incluir novos fluxos na tabela de fluxos, por falta de memória.

Figura 14 – Negação de serviço - plano de dados



Fonte: Elaborada pelo autor

4.4.3 Resultados dos ataques de negação de serviço

Tanto na rede de controle quanto na rede cliente SDN, os resultados são semelhantes em relação à quantidade possível de fluxos inseridos e o comportamento após a tabela de fluxo ficar cheia.

Foi usado um script monitorando a memória livre da máquina que executou a simulação, assim como a contagem de fluxos instalados no comutador da Rede Definida por Software (Apêndice E). O monitoramento da quantidade de fluxos no comutador e memória livre da máquina que executa a simulação deixa claro que o Open vSwitch não considera as limitações de memória da máquina e não tem um número máximo de fluxos que pode armazenar. O comutador aceita fluxos enquanto houver memória livre na máquina. O número de fluxos instalados variou de acordo com a quantidade de aplicações em execução na máquina no momento dos testes, em alguns casos chegando a 450.000 fluxos, outros cerca de 600.000 fluxos. Quando a quantidade de

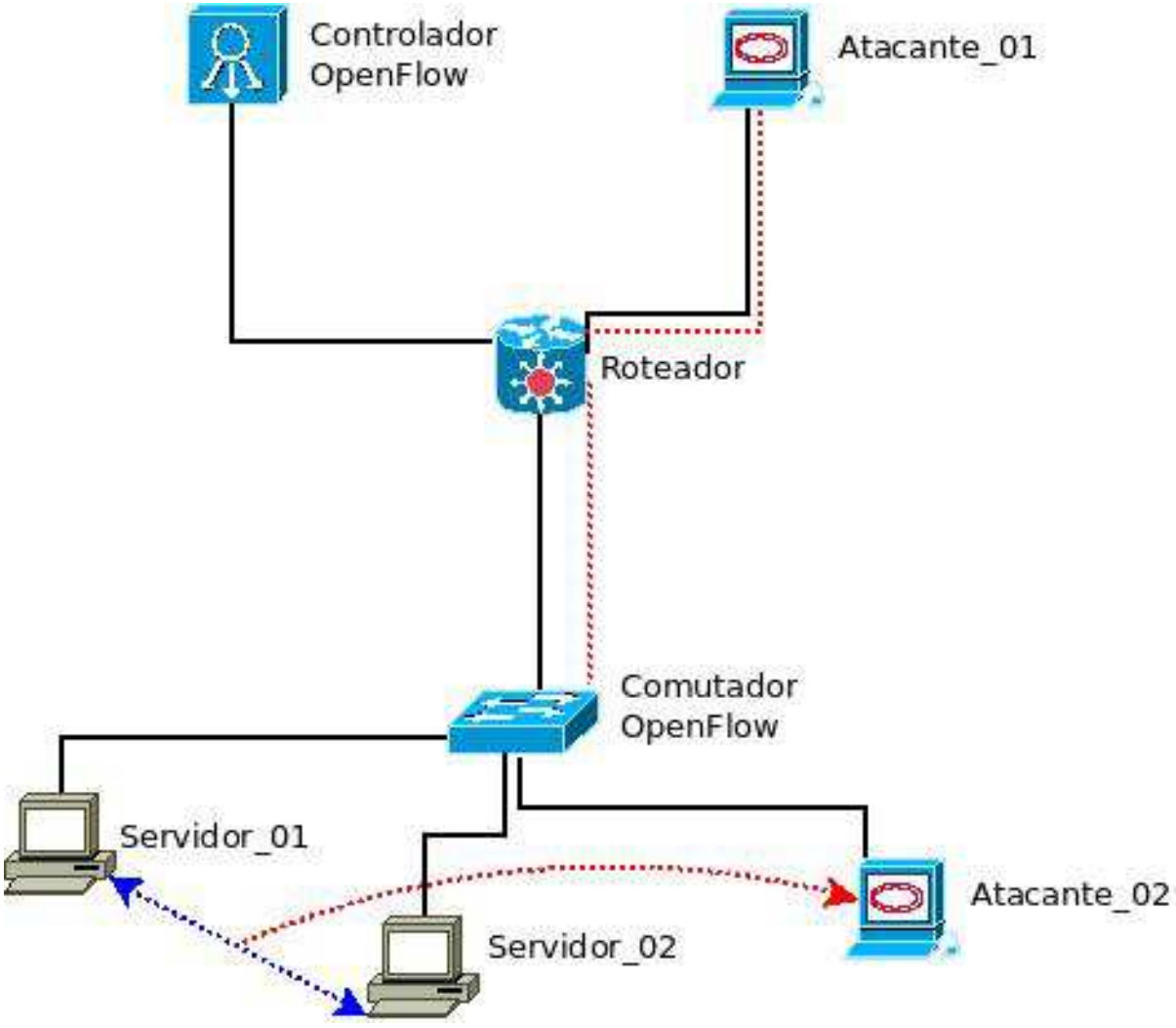
memória livre ficou muito baixa, abaixo de 10 megabytes livres, o sistema operacional encerrou o processo Open vSwitch tentando evitar uma negação de serviço completa. A consequência desse encerramento é que todas as comunicações de rede com base no comutador pararam tornando o ataque de negação de serviços bem sucedido.

Os resultados observados nesse teste confirmam as afirmativas de Scott-Hayward, Natarajan e Sezer (2015) quanto ao risco de ataques partindo de um usuário malicioso na rede de clientes SDN.

4.5 Ataque homem-no-meio

Outra possibilidade que um intruso tem na rede do controlador OpenFlow é a criação de portas espelhadas nos comutadores OpenFlow, inserindo ações extras quando ocorrem modificações de fluxo.

Figura 15 – Ataque homem-no-meio



Ettercap é uma ferramenta para ataques homem no meio capaz monitorar

conexões ao vivo e filtrar conteúdo em tempo real, suportando dissecção ativa e passiva de muitos protocolos (ETTERCAP, 2016). A ferramenta *Ettercap* foi utilizada para analisar pacotes capturados com comandos do controlador OpenFlow. A partir dessa análise, foi possível injetar pacortes de controle que criam novas regras de envio de pacotes, espelhando o tráfego entre estações de trabalho legítimas na porta de uma estação de trabalho maliciosa na rede OpenFlow.

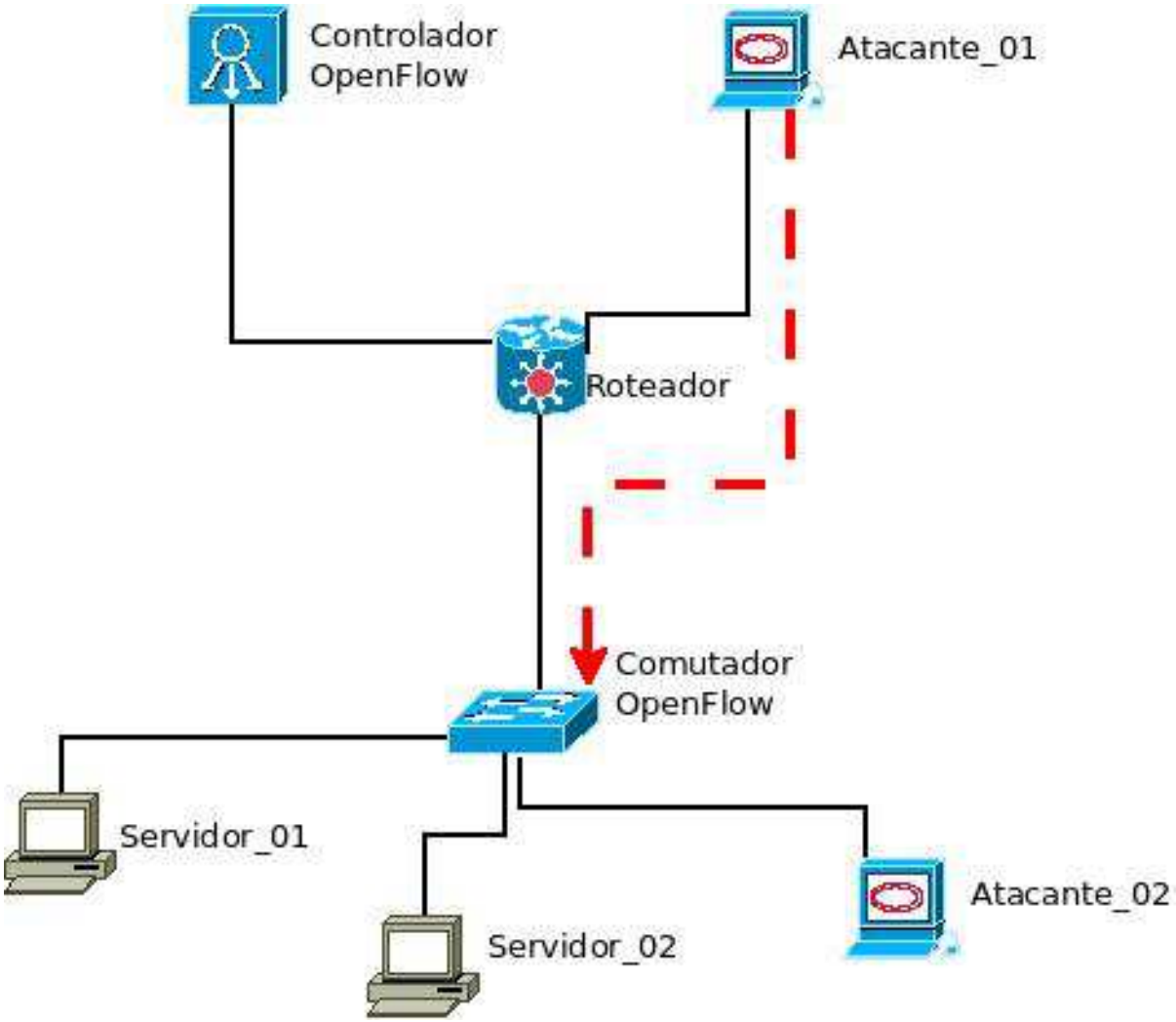
Foi necessário estender a ferramenta *Ettercap*, a fim de torná–la capaz de analisar os pacotes de entrada filtrando comandos do controlador e manipular os pacotes para injeção de nova regra que provoca o espelhamento (Apêndice F). A interface de rede da máquina Atacante_02 foi configurada em modo promíscuo, permitindo capturar o tráfego das máquinas Servidor_01 e Servidor_02 que foi espelhado na interface da máquins Atacante_02.

A ausência da obrigatoriedade do uso de TLS, conforme Benton, Camp e Small (2013) prevê, torna mais fácil a tarefa de executar esse ataque no ambiente de Redes Definidas por Software.

4.6 Acesso à interface de depuração

Alguns comutadores OpenFlow disponibilizam uma interface de depuração, muitas vezes chamado de modo de escuta. É uma conexão TCP em texto plano sem formatação que dá acesso ao plano de dados do comutador e destina-se apenas para fins de depuração. Ela fornece ao usuário a capacidade de criar, modificar e apagar os caminhos de dados de comutação junto com a manipulação de fluxos na tabela de fluxo.

Figura 16 – Acesso à interface de depuração do comutador



Fonte: Elaborada pelo autor

O comutador testado, Open vSwitch, não possui esse recurso. Por esse motivo, não foi possível simular qualquer ataque explorando essa característica do protocolo OpenFlow. Porém, qualquer conexão TCP em texto plano está sujeita a ataques clássicos, como homem-no-meio e espionagem. Poertanto, equipamentos onde esse recurso esteja disponível e fique habilitado, seja por descuido ou por uso de configurações padrão de fábrica, estão sujeitos a tais ataques, facilmente executados.

4.7 Testes com comutador físico

Durante o desenvolvimento do presente trabalho, houve a possibilidade de usar um comutador doméstico que possibilita a substituição do sistema operacional para suportar o protocolo OpenFlow. O equipamento é um TP-LINK modelo WR842ND. É um roteador sem fio que possúi cinco portas ethernet, quatro delas para rede local e uma para conexão com a internet.

O sistema operacional original dele foi substituido pelo sistema OpenWRT versão 15.05. OpenWrt é uma distribuição Linux para dispositivos embarcados que oferece um sistema de arquivos com gerenciamento de pacotes, permitindo a personalização o dispositivo através do uso de pacotes para atender diferentes aplicação. O uso desse sistema permite várias adequações por parte do usuário, entre elas a instalação do comutador Open vSwitch (OpenWRT team, 2016).

A ideia é fazer com que as portas ethernet físicas do roteador sejam associadas ao Open vSwitch para que este administre-as usando o protocolo OpenFlow. Para isso, cada porta física é isolada em uma VLAN, e o Open vSwitch se encarrega de associar essas VLANs logicamente, permitindo a troca de pacotes de rede entre elas. O comutador Open vSwitch recebe as informações de conexão com o controlador de Rede Definida por Software, que assume então o controle das portas físicas do comutador.

Durante os experimentos, não houve troca de pacotes entre as portas físicas do comutador. A configuração seguiu a documentação disponível. Ficou a suspeita de pacotes de instalação incompatíveis com o equipamento usado. Tentando superar essa dificuldade, o sistema operacional OpenWRT foi recompilado diversas vezes, em diferentes configurações, não alcançando sucesso. Isso inviabilizou qualquer experimento proposto pelo presente trabalho.

4.8 Considerações

As simulações permitiram mostrar que fragilidades expostas pelo protocolo OpenFlow podem ser exploradas através de técnicas de ataques clássicas, como homem-no-meio e inundação de pacotes. A ausência de obrigatoriedade do uso de TLS no canal de comunicações entre controlador e comutador exibe um desleixo com a segurança na especificação do protocolo. Pode-se supor que seja intencional, talvez para baixar custo de fabricação de equipamentos com suporte a OpenFlow sem a necessidade de processadores mais poderosos.

Foi possível injetar pacotes de controle, comandando o comutador OpenFlow para que copiasse de tráfego de estações de trabalho legítimas em uma interface de rede de um cliente malicioso.

Além disso, a solução adotada em alguns comutadores OpenFlow físicos é executar internamente o comutador virtual Open vSwitch. Isso pode significar que as situações simuladas neste trabalho são transferidas para esses equipamentos praticamente sem nenhuma diferennça em relação ao ambiente simulado.

5 Conclusão e trabalhos futuros

5.1 Conclusão

Neste trabalho foi proposta uma análise de possíveis ameaças de segurança em conseqüência da implantação do paradigma de Redes Definidas por Software através da adoção do protocolo OpenFlow para controle dos equipamentos comutadores de pacotes de rede. Essa análise foi feita com o uso de simuladores de rede capazes de emular estações de trabalho, comutadores de controladores, além de máquinas físicas conectadas a esses ambientes simulados.

Observa-se que as facilidades oferecidas pelo paradigma de Redes Definidas por Software aos administradores de rede expõem falhas de segurança decorrentes de vulnerabilidades oriundas de características da especificação do protocolo OpenFlow adotado pelo mercado para implementação de Redes Definidas por Software.

Os experimentos conduzidos ilustram algumas dessas vulnerabilidades que afetam redes OpenFlow baseadas na especificação adotada de forma generalizada por fornecedores de equipamentos de rede. As redes dependem fortemente de mensagens de chegada de pacotes, sendo expostas a ataques de negação de serviço contra comutadores e controladores. Pode-se observar também que a segurança do canal de comunicações entre controlador e comutadores tem sido amplamente ignorada, tornando OpenFlow vulnerável a ataques homem-no-meio.

As simulações realizadas produziram resultados coerentes com o esperado quanto aos ataques de negação de serviço e homem-no-meio. Ficou evidenciado que um usuário malicioso pode prejudicar a operação de uma Rede Definida por Software e manipular seu conteúdo com técnicas amplamente conhecidas.

5.2 Trabalhos futuros

Considerando que o presente trabalho descreveu testes sobre ambientes simulados virtualmente, como possíveis trabalhos futuros pode-se sugerir:

- Uso de comutadores físicos de mercado para execução de simulações de ataques semelhantes às propostas aqui;
- Estudo comparativo de resultados obtidos em ambientes simulados e ambientes físicos;
- Estudo comparativo entre equipamentos comerciais quanto aos aspectos de segurança enumerados neste trabalho;

- Ferramentas disponíveis para oferecer segurança a Redes Definidas por Software;
- Análise de outros protocolos adotados para a implementação de Redes Definidas por Software, do ponto de vista de segurança de redes.

Referências

AMIN, Aniruddh. Vulnerabilities of Openflow Network and Network Security for SDN Network. *Global Journal For Research Analysis*, v. 5, n. 5, 2016.

ANTONENKO, Vitaly.; SMELYANSKIY, Ruslan. Global network modelling based on mininet approach. In: ACM. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. [S.l.], 2013. p. 145 – 146.

BENTON, Kevin.; CAMP, L Jean.; SMALL, Chris. Openflow vulnerability assessment. In: ACM. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. [S.l.], 2013. p. 151 – 152.

Big Switch Networks. *Floodlight OpenFlow Controller*. 2016. Disponível em: <http://www.projectfloodlight.org/floodlight/>. Acesso em: 25/09/2016.

BIONDE, Phillipe. *Scapy*. 2016. Disponível em: <http://www.secdev.org/projects/scapy>. Acesso em: 25/10/2016.

BRANDT, Markus. et al. Security Analysis of Software Defined Networking ProtocolsâOpenFlow, OF-Config and OVSDB. In: *The 2014 IEEE Fifth International Conference on Communications and Electronics (ICCE 2014), DA NANG, Vietnam*. [S.l.: s.n.], 2014.

COMUNNITY, SDN Framework. *Ryu SDN Framework*. 2016. Disponível em: <https://osrg.github.io/ryu/>. Acesso em: 18/10/2016.

COSTA, Victor Torres da. et al. Controle e Isolamento de Recursos em Ambientes de Redes Virtuais OpenFlow. 2014.

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

ETTERCAP. *Ettercap Home Page*. 2016. Disponível em: <https://ettercap.github.io/ettercap/>. Acesso em: 25/10/2016.

FEAMSTER, Nick.; REXFORD, Jennifer.; ZEGURA, Ellen. The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, ACM, v. 44, n. 2, p. 87 – 98, 2014.

FOSTER, Nate. et al. Languages for software-defined networks. *Communications Magazine, IEEE*, IEEE, v. 51, n. 2, p. 128 – 134, 2013.

FOUNDATION, Linux. *Open vSwitch*. 2016. Disponível em: <http://openvswitch.org/>. Acesso em: 18/10/2016.

FOUNDATION, Linux. *Open vSwitch*. 2016. Disponível em: <http://openvswitch.org/>. Acesso em: 18/10/2016.

FOUNDATION, Linux. *The OpenDayLight Platform*. 2016. Disponível em: <https://www.opendaylight.org/>. Acesso em: 18/10/2016.

FOUNDATION, Open Networking. *OpenFlow Switch Specification Version 1.3.0*. 2012. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf>. Acesso em: 29/09/2016.

HELLER, Brandon. *OpenFlow Switch Specification Version 1.0.0*. 2009. Disponível em: <http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf>. Acesso em: 29/09/2016.

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.

IRONGEEK. *Manual Page - arpspoof*. 2016. Disponível em: <http://www.irongeek.com/i.php?page=backtrack-3-man/arpspoof>. Acesso em: 25/10/2016.

JAVID, Tariq.; RIAZ, Tehseen.; RASHEED, Asad. A layer 2 firewall for software defined network. In: IEEE. *Information Assurance and Cyber Security (CIACS), 2014 Conference on*. [S.l.], 2014. p. 39 – 42.

KIM, Hyojoon.; FEAMSTER, Nick. Improving network management with software defined networking. *IEEE Communications Magazine*, IEEE, v. 51, n. 2, p. 114 – 119, 2013.

KREUTZ, Diego.; RAMOS, Fernando.; VERISSIMO, Paulo. Towards secure and dependable software-defined networks. In: ACM. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. [S.l.], 2013. p. 55 – 60.

KREUTZ, Diego. et al. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, IEEE, v. 103, n. 1, p. 14 – 76, 2015.

LANTZ, Bob.; HELLER, Brandon.; MCKEOWN, Nick. A network in a laptop: rapid prototyping for software-defined networks. In: ACM. *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. [S.l.], 2010.

LARA, Adrian.; RAMAMURTHY, Byrav. Opensec: A framework for implementing security policies using openflow. In: IEEE. *2014 IEEE Global Communications Conference*. [S.l.], 2014. p. 781 – 786.

MARCONDES, Adi Nascimento. *Um Mecanismo para Gerência de Segurança da Autenticação e Controle de Acesso em redes SDN*. 2016. Dissertação (Mestrado).

MATTOS, Diogo Menezes Ferrazani.; DUARTE, Otto Carlos Muniz Bandeira. AuthFlow: Um Mecanismo de Autenticação e Controle de Acesso para Redes Definidas por Software. 2014.

MININET. *Mininet - An Instant Virtual Network on your Laptop (or other PC)*. 2016. Disponível em: <http://mininet.org/>. Acesso em: 25/09/2016.

NOXRepo Community. *NOXRepo*. 2016. Disponível em: <http://www.noxrepo.org/>. Acesso em: 18/10/2016.

OLIVEIRA, Rogério Leão Santos de. et al. Using mininet for emulation and prototyping software-defined networks. In: IEEE. *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*. [S.l.], 2014. p. 1 – 6.

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

SCOTT-HAYWARD, Sandra.; NATARAJAN, Sriram.; SEZER, Sakir. A survey of security in software defined networks. *IEEE Communications Surveys & Tutorials*, IEEE, v. 18, n. 1, p. 623 – 654, 2015.

SOUZA, Osiel Oliveira.; NOBRE, Jeferson Campos. Autenticação de Estações de Trabalho em Redes Definidas por Software com Utilização de Certificados Auto-Assinados. 2016.

YOU, Wanqing. et al. OpenFlow security threat detection and defense services. *International Journal of Advanced Networking and Applications*, Eswar Publications, v. 6, n. 3, 2014.

Apêndices

APÊNDICE A – Script de topologia de rede complexa

O arquivo Python “centroDeDadosLinkRedundante.py” instancia a topologia exibida na figura 8. Observa-se as possibilidades de simulação permitidas com o uso do Mininet.

```
"""
Script para criar topologia de centro de dados no Mininet
"""

from mininet.topo import Topo
from mininet.util import irange

class TopologiaCentroDeDadosLinkRedundante( Topo ):
      "Topologia de Centro de Dados de Link Redundante com
   comutadores de alta disponibilidade"

      def build( self, numPrateleiras=4, numHostsPorPrateleira
   =4, numComutadores=2 ):
            # Essa configuracao suporta somente 15 comutadores
   raiz ou menos
            if numComutadores >= 16:
                  raise Exception( "Usar menos de 16
   comutadores de alta disponibilidade" )

            self.racks = []
            comutadoresRaiz = []
            ultimoComutadorRaiz = None

            # Cria e conecta todos os comutadores raiz
            for i in irange( 1, numComutadores ):
                  comutadorRaiz = self.addSwitch( 's%s' % i )
                  comutadoresRaiz.append( comutadorRaiz )

                  # Se inicializamos pelo menos 2 comutadores,
   nos certificamos de
                  # conecta-los. s1 -> s2 -> ... -> sN
                  if ultimoComutadorRaiz:
                        self.addLink( ultimoComutadorRaiz,
   comutadorRaiz )

                  ultimoComutadorRaiz = comutadorRaiz

            # faz a conexao final do ultimo comutador com o
   primeiro comutador
            if numComutadores > 1:
                  self.addLink( ultimoComutadorRaiz,
   comutadoresRaiz[0] )

            for i in irange( 1, numPrateleiras ):
```

```
                rack = self.instanciaPrateleira( i,
    numHostsPorPrateleira=numHostsPorPrateleira )
                self.racks.append( rack )
                for comutador in rack:
                    for comutadorRaiz in comutadoresRaiz:
                        self.addLink( comutadorRaiz,
    comutador )

    def instanciaPrateleira( self, loc, numHostsPorPrateleira
    ):
        "Instancia um Prateleira de hosts com um comutador
    de topo de Prateleiras"

        dpid = ( loc * 16 ) + 1
        comutador = self.addSwitch( 's1r%s' % loc, dpid='%x
    ' % dpid )

        for n in irange( 1, numHostsPorPrateleira ):
            host = self.addHost( 'h%sr%s' % ( n, loc ) )
            self.addLink( comutador, host )

        # Retorna lista de comutadores de topo de
    Prateleira para esta Prateleira
        return [comutador]

# Arquivo pode ser importado usando 'mn --custom <arquivo> --
    topo cdraizaltadisponibilidade'
topos = {
    'cdraizaltadisponibilidade':
    TopologiaCentroDeDadosLinkRedundante
}
```

APÊNDICE B – Script da topologia usada nos experimentos

O arquivo Python “topologia_simulacao_tcc.py” instancia a topologia exibida na figura 10 através do Mininet.

```
#!/usr/bin/python

"""
Um script de topologia mista virtual/real para Mininet.

[Introduction to Mininet]: https://github.com/mininet/mininet/
   wiki/Introduction-to-Mininet#apilevels

"""

from mininet.cli import CLI
from mininet.log import setLogLevel, info, error
from mininet.net import Mininet
from mininet.topo import Topo
from mininet.node import RemoteController, OVSSwitch
from mininet.link import Intf

class TopologiaSimulacaoTCC( Topo ):
      "Topologia minima com um comutador, dois hosts virtuais e
    um host real"

      def build( self ):
            # Cria dois hosts.
            h1 = self.addHost( 'h1' )
            h2 = self.addHost( 'h2' )

            # Cria um comutador
            s1 = self.addSwitch( 's1' )

            # Adiciona links entre o comutador e cada host
            self.addLink( s1, h1 )
            self.addLink( s1, h2 )

def executarTopologiaSimulacaoTCC():
      "Inicializa uma rede Mininet usando a topologia de
   simulaca    o  do TCC"

      intfName = 'enp2s0'
     # Cria uma instancia da topologia
      topo = TopologiaSimulacaoTCC()

      # Cria uma rede baseada na topologia usando Open VSwitch
   e controlador remoto
      net = Mininet(
            topo=topo,
```

```
            controller=lambda name: RemoteController( name, ip
   ='104.198.219.198' ),
            switch=OVSSwitch,
            autoSetMacs=True )
      switch = net.switches[ 0 ]
#     info ( '*** Adding hardware interface', intfName, 'to
   switch',
#               switch .name, '\n' )
      _intf = Intf( intfName, node=switch )

#     info ( '*** Note: you may need to reconfigure the
   interfaces for '
#               'the Mininet hosts:\n', net.hosts, '\n' )

      # inicia a rede
      net.start()

      # abre uma Interface de linha de comandos para o usuario
   executar comandos
      CLI( net )

      # Ao sair da linha de comandos, finaliza a rede
      net.stop()

if __name__ == '__main__':
      # Executa quando este arquivo eh chamado diretamente
      setLogLevel( 'info' )
      executarTopologiaSimulacaoTCC()

# Permite importar este arquivo usando 'mn --custom <filename>
   --topo topologiaTCC'
topos = {
      'topologiaSimulacaoTCC': TopologiaSimulacaoTCC
}
```

APÊNDICE C – Módulo Controlador Pox - duas redes de dados

O arquivo Python “l2_learning.py” é o módulo padrão do Controlador Pox que simula o comportamento de um comutador ethernet padrão. Foi modificado para dividir suas portas em duas redes distintas, uma onde estão as estações de trabalho autênticas, e outra, a estação de trabalho maliciosa.

```
# Copyright 2011-2012 James McCauley
#
# 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.

"""
An L2 learning switch.

It is derived from one written live for an SDN crash course.
It is somwhat similar to NOX's pyswitch in that it installs
exact-match rules for each flow.
"""

from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
import time

log = core.getLogger()

# We don't want to flood immediately when a switch connects.
# Can be overriden on commandline.
_flood_delay = 0

class LearningSwitch (object):
  """
```

```
  The learning switch "brain" associated with a single OpenFlow
      switch.

  When we see a packet, we'd like to output it on a port which
     will
  eventually lead to the destination.  To accomplish this, we
     build a
  table that maps addresses to ports.

  We populate the table by observing traffic.  When we see a
     packet
  from some source coming from some port, we know that source
     is out
  that port.

  When we want to forward traffic, we look up the desintation
     in our
  table.  If we don't know the port, we simply send the message
      out
  all ports except the one it came in on.  (In the presence of
     loops,
  this is bad!).

  In short, our algorithm looks like this:

  For each packet from the switch:
  1) Use source address and switch port to update address/port
     table
  2) Is transparent = False and either Ethertype is LLDP or the
      packet's
     destination address is a Bridge Filtered address?
     Yes:
        2a) Drop packet -- don't forward link-local traffic (
           LLDP, 802.1x)
              DONE
  3) Is destination multicast?
     Yes:
        3a) Flood the packet
            DONE
  4) Port for destination address in our address/port table?
     No:
        4a) Flood the packet
            DONE
  5) Is output port the same as input port?
     Yes:
        5a) Drop packet and similar ones for a while
  6) Install flow table entry in the switch so that this
     flow goes out the appopriate port
     6a) Send the packet out appropriate port
  """
  def __init__ (self, connection, transparent):
    # Switch we'll be adding L2 learning switch capabilities to
    self.connection = connection
    self.transparent = transparent
```

```
    # Our table
    self.macToPort = {}

    # We want to hear PacketIn messages, so we listen
    # to the connection
    connection.addListeners(self)

    # We just use this to know when to log a helpful message
    self.hold_down_expired = _flood_delay == 0

    #log.debug("Initializing LearningSwitch, transparent=%s",
    #          str(self.transparent))

  def _handle_PacketIn (self, event):
    """
    Handle packet in messages from the switch to implement
       above algorithm.
    """

    packet = event.parsed

    def flood (message = None):
      """ Floods the packet """
      msg = of.ofp_packet_out()
      if time.time() - self.connection.connect_time >=
         _flood_delay:
        # Only flood if we've been connected for a little while
           ...

        if self.hold_down_expired is False:
          # Oh yes it is!
          self.hold_down_expired = True
          log.info("%s: Flood hold-down expired -- flooding",
              dpid_to_str(event.dpid))

        if message is not None: log.debug(message)
        #log.debug("%i: flood %s -> %s", event.dpid,packet.src,
           packet.dst)
        # OFPP_FLOOD is optional; on some switches you may need
            to change
        # this to OFPP_ALL.
        msg.actions.append(of.ofp_action_output(port = of.
           OFPP_FLOOD))
      else:
        pass
        #log.info("Holding down flood for %s", dpid_to_str(
           event.dpid))
      msg.data = event.ofp
      msg.in_port = event.port
      self.connection.send(msg)

    def drop (duration = None):
      """
```

```
      Drops this packet and optionally installs a flow to
         continue
      dropping similar ones for a while
      """
      if duration is not None:
        if not isinstance(duration, tuple):
          duration = (duration,duration)
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet)
        msg.idle_timeout = duration[0]
        msg.hard_timeout = duration[1]
        msg.buffer_id = event.ofp.buffer_id
        self.connection.send(msg)
      elif event.ofp.buffer_id is not None:
        msg = of.ofp_packet_out()
        msg.buffer_id = event.ofp.buffer_id
        msg.in_port = event.port
        self.connection.send(msg)

    self.macToPort[packet.src] = event.port # 1

    if not self.transparent: # 2
      if packet.type == packet.LLDP_TYPE or packet.dst.
         isBridgeFiltered():
        drop() # 2a
        return

    if packet.dst.is_multicast:
      flood() # 3a
    else:
      if packet.dst not in self.macToPort: # 4
        flood("Port for %s unknown -- flooding" % (packet.dst,)
           ) # 4a
      else:
        port = self.macToPort[packet.dst]
        if port == event.port: # 5
          # 5a
          log.warning("Same port for packet from %s -> %s on %s
             .%s.  Drop."
              % (packet.src, packet.dst, dpid_to_str(event.dpid
                 ), port))
          drop(10)
          return
        # 6
        log.debug("installing flow for %s.%i -> %s.%i" %
                  (packet.src, event.port, packet.dst, port))
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet, event.port
           )
##########################################################################

##########################################################################

#        msg.idle_timeout = 10
```

```
#        msg.hard_timeout = 30
        msg.idle_timeout = 2
        msg.hard_timeout = 4
        if (event.port != 3):
          print "Encaminhando - porta origem: " + str(port) + "
              porta destino: " + str(event.port)
          msg.actions.append(of.ofp_action_output(port = port))
        else:
          print "Descartando  - porta origem: " + str(port) + "
              porta destino: " + str(event.port)
##########################################################################

##########################################################################

        msg.data = event.ofp # 6a
        self.connection.send(msg)

class l2_learning (object):
  """
  Waits for OpenFlow switches to connect and makes them
     learning switches.
  """
  def __init__ (self, transparent):
    core.openflow.addListeners(self)
    self.transparent = transparent

  def _handle_ConnectionUp (self, event):
    log.debug("Connection %s" % (event.connection,))
    LearningSwitch(event.connection, self.transparent)

def launch (transparent=False, hold_down=_flood_delay):
  """
  Starts an L2 learning switch.
  """
  try:
    global _flood_delay
    _flood_delay = int(str(hold_down), 10)
    assert _flood_delay >= 0
  except:
    raise RuntimeError("Expected hold-down to be a number")

  core.registerNew(l2_learning, str_to_bool(transparent))
```

APÊNDICE D – Módulo do Controlador Pox - encher tabela de fluxos

O arquivo Python “enche_tabela_fluxos.py” é um módulo responsável por encher a tabela de fluxos para causar negação de serviço por falta de confiança no Controlador.

```
#!/usr/bin/python
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr
from pox.lib.addresses import IPAddr, EthAddr
from random import randint
import time
import sys

hr_inicio = 0
nr_fluxos = 0
executar = 1

#Gera porta de comutador aleatoria
def gerar_porta_comutador(porta = None):
    porta_aleatoria = randint(1,3)
    if porta != None and porta_aleatoria == porta:
        porta_aleatoria = ((porta_aleatoria + 1) % 3) + 1
    return porta_aleatoria

#Gera IP aleatorio
def gerar_ip():
    return ".".join([str(randint(0,255)),str(randint(0,255)),str
    (randint(0,255)),str(randint(0,255))])

#Gera numero MAC aleatorio
def gerar_nr_mac():
    nr_mac = [ randint(0x00, 0xff), randint(0x00, 0xff), randint
    (0x00, 0xff), randint(0x00, 0xff), randint(0x00, 0xff),
    randint(0x00, 0xff) ]
    return ':'.join(map(lambda x: "%02x" % x, nr_mac))

def _manipuladorInicioConexao(evento):
    print "Comutador " + dpidToStr(evento.dpid) + " inicializou"
    global hr_inicio
    hr_inicio = time.time()
    global executar
    global nr_fluxos
    while executar == 0:
        ip_origem = gerar_ip()
        ip_destino = gerar_ip()
        mac_origem = gerar_nr_mac()
        mac_destino = gerar_nr_mac()
        porta_origem_comutador = gerar_porta_comutador()
        porta_destino_comutador = gerar_porta_comutador(
    porta_origem_comutador)
```

```
        mensagem = of.ofp_flow_mod()
        mensagem.match.nw_src = IPAddr(ip_origem)
        mensagem.match.nw_dst = IPAddr(ip_destino)
        mensagem.match.dl_src = EthAddr(mac_origem)
        mensagem.match.dl_dst = EthAddr(mac_destino)
        mensagem.match.in_port = porta_origem_comutador
        mensagem.actions.append(of.ofp_action_output(port =
    porta_destino_comutador))
        mensagem.match.dl_type = 0x800
        mensagem.match.dl_vlan = 1
        mensagem.match.dl_vlan_pcp = 0
        mensagem.match.nw_tos = 0x10
        mensagem.match.nw_proto = 6
        mensagem.match.tp_src = 80
        mensagem.match.tp_dst = 80
        event.connection.send(mensagem)

        nr_fluxos = nr_fluxos + 1
        print str(nr_fluxos) + " - ip_orig: " + ip_origem + "
    mac_orig: " + mac_origem + " porta_dest_comutador: " + str(
    porta_destino_comutador) + " ip_dest: " + ip_destino + "
    mac_dest: " + mac_destino + " porta_orig_comutador: " + str(
    porta_origem_comutador)
        print "Desconectou de " + dpidToStr(event.dpid)
        print str(nr_fluxos) + " fluxos incluidos em " + str(time
    .time() - hr_inicio) + " seg."

def _manipuladorFimConexao (event):
    global executar
    executar = 1

def launch ():
    core.openflow.addListenerByName("ConnectionUp",
    _manipuladorInicioConexao)
    core.openflow.addListenerByName("ConnectionDown",
    _manipuladorFimConexao)
```

APÊNDICE E – Script gerador de estatísticas

O script bash “estatisticas_01.sh” gera estatísticas de memória e número de fluxos instalados no comutador OpenFlow.

```
#!/bin/bash
inicio=‘date +%s‘
while true; do
        i =‘expr $i + 1‘
        cpu =‘top –b –n 3 –d 0.1 | grep %Cpu | awk ’{ print $4 }’
     | sort –nr | head –1‘
       mem =‘top –b –n 1 | head –4 | tail –1 | awk ’{ print $7
    }’‘
         fluxos  =‘ovs–ofctl dump–flows br0 | wc –l‘

                  tempo=‘date +%s‘
                  tempo=‘echo "$tempo – $inicio" | bc‘
                  echo "Uso de CPU: $cpu Mem ria livre: $mem KB
    N   de fluxos: $fluxos tempo: $tempo"
        sleep   1
done
```

APÊNDICE F – - Extensão da ferramenta Ettercap

O arquivo “etterfilter.tbl” define os atributos de pacotes de controle OpenFlow, necessários para filtragem e modificação dos mesmos pela feramenta Ettercap.

```
################
# etterfiler.tbl
################
#
# Openflow Header
[ofp][5]
  version  :1 = 0
  type  :1 = 1
  length  :2 = 2
  transid  :4 = 4
#
#
#Openflow Flowmod Header
[ofpfm][5]
  version  :1 = 0
  type  :1 = 1
  length  :2 = 2
  transid  :4 = 4
  matchwildcard  :4 = 8
  matchinport  :2 = 12
  matchdlsrc  :6 = 14
  matchdldst  :6 = 20
  matchdlvlan  :2 = 26
  matchvlanpcp  :1 = 28
  matchpad  :1 = 29
  matchethertype  :2 = 30
  matchnwtos  :1 = 32
  matchnwproto  :1 = 33
  matchpad2  :2 = 34
  matchnwsrc  :4 = 36
  matchnwdst  :4 = 40
  matchtpsrc  :2 = 44
  matchtpdst  :2 = 46
  cookie  :8 = 48
 command :2 = 56
  idletime  :2 = 58
  maxtime  :2 = 60
  priority  :2 = 62
  buffid  :4 = 64
  outport  :2 = 68
  flags  :2 = 70
  actiontype  :2 = 72
  actionlen  :2 = 74
  actionpad  :6 = 76
#Extra optional actions:
  # pactionoutport:2 = 78
```

```
# pactionmaxbytes:2 = 80
# actiontype:2 = 82
# actionlen:2 = 84
# aoutport:2 = 86
# maxbytes:2 = 88
# nactiontype:2 = 90
# nactionlen:2 = 92
# noutport:2 = 94
```

APÊNDICE G – Artigo em formato SBC

Uma análise de segurança de Redes Definidas por Software sobre protocolo OpenFlow

Paulo Vieira Centeno [1]

[1] Departamento de Informática e Estatística – UFSC

paulo.vieira.centeno@gmail.com

Abstract. *The growth in the use of cloud computing takes network infrastructure to its limits, creating scalability and management challenges. In this scenario, Software Defined Network (SDN) has emerged, which makes it possible to redefine the topology and make routing decisions dynamically, with a programmable network.*

The flexibility of SDN has brought security risks. Current applications consider paradigms of obsolete communication protocols. They were not designed for this intelligent networking environment, with security breaches arising from this. The environment is hybrid, with conventional applications, developed for conventional networks, operating over SDN.

The present work presents a descriptive analysis of SDN on OpenFlow protocol. It also empirically demonstrates security issues arising from the adoption of the OpenFlow protocol.

Resumo. *O crescimento no uso de computação em nuvem leva a infraestrutura de redes a seu limite, criando desafios de escalabilidade e administração. Neste cenário, surgiram as Redes Definidas por Software (Software Defined Network - SDN), que possibilitam redefinir a topologia e tomar decisões de roteamento dinamicamente, com uma rede programável.*

A flexibilidade de SDN trouxe riscos de segurança. As aplicações atuais consideram paradigmas de protocolos de comunicação obsoletos. Não foram projetadas para esse ambiente de redes inteligentes, com falhas de segurança decorrentes disso. O ambiente é híbrido, com aplicações convencionais, desenvolvidas para redes convencionais, operando sobre SDN.

O presente trabalho apresenta uma análise descritiva de SDN sobre protocolo OpenFlow. Também demonstra empíricamente problemas de segurança decorrentes da adoção do protocolo OpenFlow.

1. Introdução

A computação em nuvem se expande, enquanto as ferramentas atuais de administração dos recursos de infra-estrutura apresentam rigidez na política de mudanças e complexidade em sua gerência, tornando necessário investigar novas maneiras de administrar os recursos. A operação de uma rede de computadores é uma tarefa difícil. Para expressar as políticas de rede de alto nível, operadores de rede precisam configurar cada dispositivo de rede, separadamente - a partir de um conjunto heterogêneo de comutadores, roteadores, etc. - usando comandos de baixo nível e específicos de cada fornecedor. Além disso, as redes são dinâmicas. Há poucos mecanismos para responder automaticamente a eventos de rede [Feamster *et al.*, 2014; Kim e Feamster, 2013].

Um novo tipo de arquitetura de redes surgiu como solução: as Redes Definida por Software (SDN). Em SDN, uma máquina logicamente centralizada, o controlador, administra um conjunto distribuído de comutadores. Essa máquina é capaz de computações arbitrárias que permitem redefinir a topologia da rede e tomar decisões de roteamento. SDN possibilita a implementação de uma nova gama de aplicações nas redes de computadores. Os recursos de hardware são facilmente compartilhados. A separação entre planos de controle e de dados constitui a base para SDN. Comutadores de rede tornam-se dispositivos de encaminhamento simples. A lógica de controle é implementada e centralizada em um controlador, que regula o comportamento da rede [Amin, 2016].

Um resultado indesejado do uso de SDN é a exposição de falhas de segurança com potencial de exploração por atacantes maliciosos. São decorrentes de características de SDN. A programação da rede e a centralização da lógica de controle introduzem falhas e ataques aos planos, abrindo as portas para ameaças que não existiam ou eram mais difíceis de explorar [Mattos e Duarte, 2014; Kreutz *et al.*, 2013].

O protocolo mais comumente adotado para implementação de redes SDN é o OpenFlow. Ele precisa ser analisado em termos de segurança, ou pode levar a riscos de segurança severos [Brandt *et al.*, 2014; Benton *et al.*, 2013].

A implementação de SDN traz um desafio: a evolução de arquiteturas de rede, contra um aumento no potencial das ameaças de ataques. Os benefícios dessa evolução devem ser preservados, neutralizando os perigos por ela potencializados [Kreutz *et al.*, 2013]. O objetivo deste trabalho é apontar vulnerabilidades de SDN sobre protocolo OpenFlow e simular ataques explorando-as. Um ambiente SDN é simulado. Após, é gerado tráfego validando a implementação do ambiente. Então, são tentados ataques explorando características de SDN. Ao final, são documentados e apresentados os resultados.

2. Redes Definidas por Software (SDN)

Redes de computadores têm sido construídas como um conjunto de dispositivos de hardware com propósitos distintos entre si, processando informações sobre políticas, monitoramento de tráfego e roteamento, entre muitos outros serviços. Essas tarefas são configuradas através de interfaces de configuração heterogêneas entre si, que permitem configurar separadamente, por exemplo, firewalls, roteadores, comutadores, listas de acesso e balanceamento de carga.

Rede Definida por Software (SDN) é um novo paradigma em configuração de redes de computadores, que se baseia na separação das funções de controle, no plano de controle, das funções de encaminhamento de quadros, no plano de encaminhamento. A separação busca prover maior flexibilidade às funções de controle enquanto o hardware especializado para comutar quadros a alta velocidade permanece inalterado. SDN oferece uma alta programabilidade das funções de controle da rede em um comutador com alto desempenho de encaminhamento de quadros. O operador pode definir de maneira simples os fluxos e as ações sobre os fluxos através de uma interface de programação de aplicação [Mattos e Duarte, 2014].

De acordo com Foster *et al.* [2013], SDN é uma rede onde um controlador centralizado administra uma coleção distribuída de comutadores, permitindo controlar o comportamento de toda a rede sem precisar acessar diretamante diretamente cada comutador; em vez disso, o controlador implementa as regras de encaminhamento de pacotes nos comutadores que administra. O resultado é uma ilusão de controle centralizado do comportamento da rede.

Kreutz *et al.* [2013] afirmam que é mais simples e menos propenso a erros modificar as políticas de rede através de software, do que através de configurações de dispositivos de baixo nível. Um programa de controle pode reagir automaticamente a alterações espúrias do estado da rede e, assim, manter as políticas de alto nível no lugar. A centralização da lógica de controle em um controlador com conhecimento global do estado da rede simplifica o desenvolvimento de funções de rede mais sofisticados. Esta capacidade de programar a rede, a fim de controlar o plano de dados subjacente é a proposta de valor crucial de SDN.

2.1. O protocolo OpenFlow

O protocolo OpenFlow é um dos protocolos existentes para implementação de SDN. Para Benton *et al.* [2013], OpenFlow e outros protocolos de SDN têm gerado interesse devido ao grande controle oferecido aos desenvolvedores de software de controle de rede. Criando uma interface normalizada, acessível pela rede para controlar o plano de

dados de equipamentos de rede, a lógica do plano de controle pode ser movida de dispositivos de rede individuais para um controlador ou grupo de controladores centralizado. Implementando a lógica de controle no controlador, mudanças de protocolo de rede e requisitos complexos de engenharia de tráfego podem ser ajustados reconfigurando, atualizando ou trocando o controlador ao invés de atualizar ou substituir o equipamento de rede.

O protocolo OpenFlow separa a rede em dois planos: o plano de controle, responsável pela execução dos algoritmos de controle da rede, e o plano de dados, responsável pelo encaminhamento e tratamento dos pacotes de rede em si. É baseado em encaminhamento por fluxos, se aproveitando do fato de que grande parte dos fabricantes de comutadores e roteadores atuais implementam uma tabela de fluxos e coleta de estatísticas diretamente nos equipamentos [da Costa *et al.*, 2014].

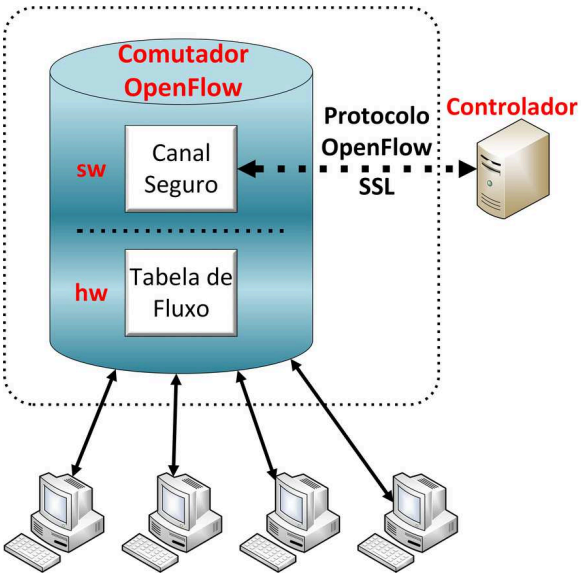


Figura 1. Arquitetura OpenFlow

O plano de dados é controlado com o fornecimento de regras, referidas como fluxos, para os dispositivos de rede, referidos como comutadores. Cada fluxo especifica uma condição de combinação com pacotes de entrada e uma instrução a ser aplicada aos pacotes combinados. A idéia básica é combinar uma regra com a execução de uma ação [Feamster *et al.*, 2014; Kreutz *et al.*, 2015].

Há dois estilos de criação de regra em redes OpenFlow: proativo, onde as regras são inseridas em comutadores antes que sejam necessárias; e reativo, no qual regras são inseridas em comutadores em resposta a pacotes observados pelo controlador, através de mensagens Packet-In. Em redes reativas, o comutador gera uma mensagem Packet-in em resposta a um pacote que não corresponde a uma regra, a qual encapsula o pacote e envia para o controlador para que tome uma decisão. O controlador pode então reconhecer

a mensagem e ignorar o pacote, ou responder com uma ação para aplicar ao pacote, juntamente com uma regra opcional para instalar na tabela de fluxos do comutador para combinar futuros pacotes [Hu *et al.*, 2014].

De acordo com a especificação de OpenFlow, as regras de fluxo para um comutador só podem ser instaladas por um controlador através de uma conexão TCP iniciada pelo comutador. Porém, alguns comutadores suportam um "modo de escuta" adicional, em que aceitam conexões através de uma porta TCP. Estas conexões também podem ser usadas para escrever regras para comutadores e ler informações a partir deles, introduzindo uma importante vulnerabilidade, por não possuir métodos de autenticação ou controle de acesso embutidos [Scott-Hayward *et al.*, 2015].

As comunicações entre o controlador e os comutadores são realizadas através de uma conexão TCP que pode, opcionalmente, ser protegida por TLS com certificados mutuamente autenticados. A conexão TCP é iniciada pelo comutador com uma configuração definida pelo operador que especifica o endereço IP e porta TCP do controlador [Javid *et al.*, 2014].

As entradas na tabela de fluxos no OpenFlow são compostas por campos de cabeçalho, contadores e ações. Os campos de cabeçalho descrevem os tipos de pacotes pertencentes àquele fluxo. A descrição de um fluxo é feita baseada em um conjunto de campos disponíveis no OpenFlow, que reúne características de várias camadas de protocolo. Assim, um fluxo poderia ser descrito como, por exemplo, todos os pacotes vindos da porta 1 do comutador e com IP de origem 146.164.69.2 [da Costa *et al.*, 2014]. Cada fluxo possui contadores associados, e cada contador registra dados do fluxo descrito, como quantidade de dados transmitidos, duração e quantidade de pacotes transmitidos. Além dos contadores, cada fluxo também possui um conjunto de ações definidas pelo controlador para serem aplicadas aos pacotes do fluxo. Dentre os diferentes tipos de ações, existem ações para fazer o encaminhamento de um pacote em uma determinada porta de saída, descartar o pacote ou modificar um determinado campo do cabeçalho do pacote.

2.2. O Simulador de Redes Mininet

Mininet é um simulador de Redes que foi desenvolvido por Bob Lantz e Brian O'Connor como uma bancada de testes de rede. De acordo com Mininet [2016], Mininet é um sistema de prototipagem rápida, cujo objetivo é o desenvolvimento de aplicações de controladores OpenFlow. É capaz de criar uma rede de estações de trabalho virtuais, comutadores, controladores e conexões. Estações de trabalho virtuais executam software de rede padrão do Linux, e seus comutadores suportam OpenFlow. Mininet dá suporte à virtualização de uma rede experimental.

Mininet permite testes com topologias complexas, sem a necessidade de conexão a

uma rede física. Inclui uma interface de linha de comando, para testes de depuração ou execução em qualquer ponto da rede. Suporta topologias personalizadas arbitrárias, e inclui um conjunto básico de topologias parametrizadas. Fornece uma interface de programação simples e extensível em linguagem Python [de Oliveira *et al.*, 2014].

2.3. Comutador OpenFlow Open vSwitch

Open vSwitch é um comutador virtual com suporte a OpenFlow. É projetado para permitir a automatização de grandes redes através da extensão programática [Foundation, 2016].

2.4. Controladores OpenFlow

Há diversas opções de controlador OpenFlow disponíveis, a maioria deles de uso gratuito, como OpenDayLight, Open vSwitch e POX. Em geral, são extensíveis através da implementação de módulos em linguagens de alto nível como C++, Java e Python .

3. Vulnerabilidades em Redes Definidas por Software baseadas em protocolo OpenFlow

Segundo Mattos e Duarte [2014]; Souza e Nobre [2016]; Benton *et al.* [2013] a adoção do protocolo OpenFlow como forma de implementar SDN trouxe um nível maior de abstração na administração de redes. Porém, potencializou o risco de ataques maliciosos a essas redes usando técnicas de ataques conhecidas, tirando proveito de características oriundas de SDN. Segue a descrição de algumas vulnerabilidades apresentadas pelo protocolo OpenFlow:

1) Falta de autenticação de origem: o protocolo OpenFlow não possui um mecanismo nativo para autenticação segura da origem. As estações finais se autenticam a uma rede OpenFlow através da validação de seus endereços MAC (Media Access Control) ou IP (Internet Protocol) [Souza e Nobre, 2016].
2) Canal de comunicação inseguro entre controlador e comutadores: a especificação inicial do OpenFlow previa o uso obrigatório de proteção TLS [Heller, 2009]. Porém, em versões posteriores, esse requisito passou a ser opcional [Foundation, 2012]. A falta de suporte a TLS deixa espaço para atacantes se infiltrarem em redes OpenFlow sem detecção.
3) Vulnerabilidade de componentes: a ausência de confiança entre componentes compromete uma Rede SDN, pois as aplicações executadas sobre o controlador podem ter comportamentos maliciosos [Mattos e Duarte, 2014]. Não é um desafio de segurança exclusivo desse paradigma. Existem três possıveis fontes de vulnerabilidade de componentes: comutadores, controlador e estações de gerenciamento. Uma vulnerabilidade em um comutador pode permitir que um atacante que obtenha acesso

a um comutador execute um ataque contra o plano de controle, como a falsificação de mensagens de controle para esgotar os recursos do controlador. Uma vulnerabilidade no controlador pode permitir que um atacante altere o plano de controle ou até mesmo execute uma nova aplicação de controle. Uma vulnerabilidade em uma estação de gerenciamento permite que o atacante faça configurações no plano de controle diferentes das corretas [Mattos e Duarte, 2014].

4) Risco de negação de serviço: um componente malicioso pode comprometer o funcionamento de toda a rede realizando um ataque de negação de serviço no controlador [Mattos e Duarte, 2014]. A inundação de pacotes é um dos ataques mais comuns que negam serviços de rede. O ataque de inundação efetua muitas transmissões de pacotes levando ao esgotamento dos recursos da rede. Este ataque é de difícil detecção em redes OpenFlow pela falta da relação entre a autenticação dos dispositivos e a tabela de fluxo presente nos comutadores [Marcondes, 2016]. De acordo com Mattos e Duarte [2014]; Scott-Hayward *et al.* [2015] a negação de serviço pode ocorrer tanto no plano de dados quanto no plano de controle. No plano de dados, uma estação maliciosa gerando fluxos falsos pode esgotar os recursos de banda, de memória, ou tabela de fluxos dos comutadores da rede. A negação de serviço no plano de controle pode ser causada em dois pontos distintos da rede: no controlador e na comunicação do controlador com os comutadores. É possıvel esgotar a capacidade de processamento do controlador enviando uma grande quantidade de pacotes com diferentes cabeçalhos. Todo pacote é analisado e um pacote com cabeçalho que não corresponde a nenhum fluxo já definido deve ser enviado ao controlador de rede. Assim, em um cenário em que um comutador envia uma quantidade atıpica de novos cabeçalhos de pacotes para o controlador, este pode ter seus recursos de processamento exauridos e não ser capaz de responder a pedidos de novos fluxos em tempo hábil. Igualmente, a negação de serviço pode ser alcançada quando o enlace de conexão entre o controlador e os comutadores é intencionalmente congestionado. Caso não haja redundância ou banda suficiente no mesmo, um comutador malicioso pode gerar tráfego suficiente para sobrecarregá-lo e impedir a comunicação do controlador com os comutadores.

A tabela abaixo enumera as vulnerabilidades sugeridas e os planos que estão sujeitos às mesmas:

Tabela 1. Tabela de vulnerabilidades /plano exposto

Vulnerabilidade	Plano de Controle	Plano de dados
Falta de autenticação de origem	Sim	Sim
Canal de comunicação inseguro	Sim	
Vulnerabilidade de componentes	Sim	
Negação de Serviço	Sim	Sim

4. Ambiente e resultados experimentais

Este trabalho se propõe a fazer uma análise e demonstrar problemas de segurança apresentados pela implementação de SDN com o uso de protocolo OpenFlow, a partir de experiências práticas.

Para viabilizar tal demonstração, usa um ambiente de simulação realístico, o Mininet, que permite emular componentes típicos de arquiteturas SDN. Segundo [Mininet, 2016] Mininet é uma excelente maneira de desenvolver, compartilhar e experimentar Sistemas SDN e OpenFlow.

Nesse ambiente de simulação, é criada uma configuração que permite executar essa demonstração empírica.

4.1. Topologia de rede

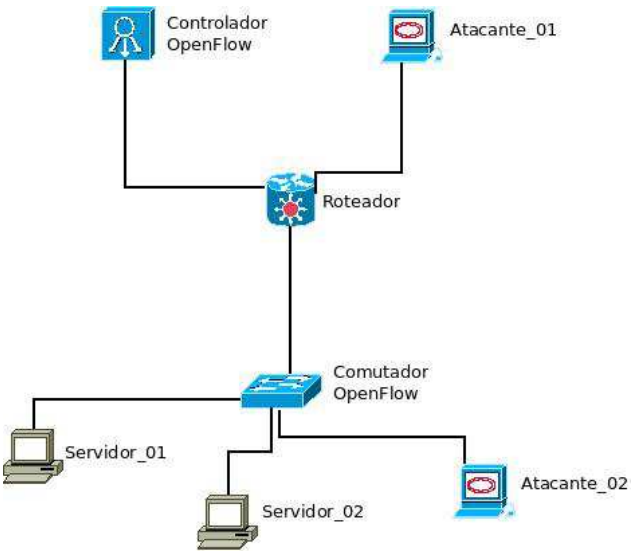


Figura 2. Topologia da rede adotada no ambiente de simulação

Tabela 2. Topologia usada nos testes

Máquina	IP	Tipo	S.O.	Usuário
Comutador	192.168.10.1	Virtual	Ubuntu Linux (Mininet)	Legítimo
Controlador	192.168.10.2	Real	Ubuntu Linux	Legítimo
Atacante_01	192.168.10.3	Real	Kali Linux	Malicioso
Servidor_01	10.0.0.1	Virtual	Ubuntu Linux (Mininet)	Legítimo
Servidor_02	10.0.0.2	Virtual	Ubuntu Linux (Mininet)	Legítimo
Atacante_02	10.0.0.3	Real	Kali Linux	Malicioso

Rede de controle SDN/OpenFlow: 192.168.10.0/24

Rede de dados SDN/OpenFlow: 10.0.0.0/24

Rede comutada: implementação de comutador Open vSwitch, executada virtualmente pelo emulador Mininet.

Os planos de controle e de dados ficam separados em duas subredes distintas, e em cada uma dessas subredes há uma máquina maliciosa, permitindo a simulação de ataques. Foi adotado um desenho de rede simples para evidenciar as situações nas simulações de ataque propostas.

Um script Python com extensões Mininet se encarrega de instanciar a topologia usada. Uma interface física da estação de trabalho que executa o Mininet é associada à rede emulada, permitindo conectar outra máquna real a esta rede. Como alguns comutadores OpenFlow comerciais executam internamente o comutador Open vSwitch, comutador virtual usado nesta configuração, os resultados obtidos nos testes executados na configuração virtualizada podem aplicar-se aos mesmos.

4.2. Interrupção do canal de comunicação comutador/controlador

O canal de comunicação entre o comutador e o controlador é um dos pontos mais importantes na infraestrutura OpenFlow. O objetivo deste cenário é observar como a implementação age quando a conexão entre o comutador eo controlador é interrompida.

Para estes testes foram utilizados os controladores OpenDayLIght, POX e FloodLight, e o comutador virtual OpenFlow Open vSwitch.

No cenário de interrupção do canal de controle entre controlador e comutador, a rede atacada é a subrede de controle OpenFlow.

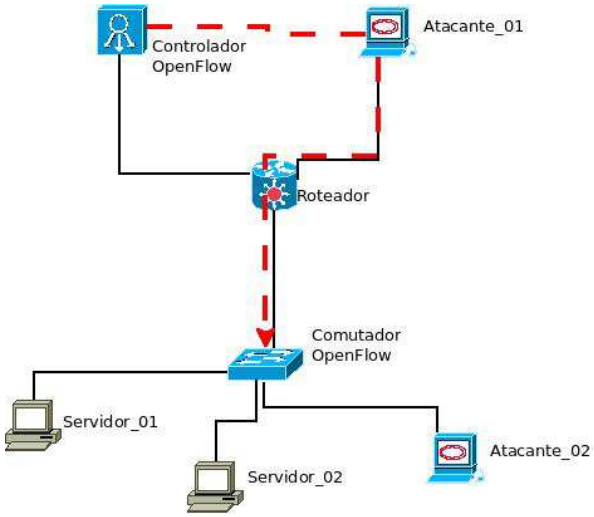


Figura 3. Ataque arpspoof

A ferramenta usada para o ataque foi o arpspoof, pertencente ao pacote dsniff. Arpspoof é capaz de redirecionar pacotes de uma estação de destino (ou todas as estações) na rede local destinados a outra estação na rede local forjando respostas ARP [IronGeek, 2016].

A ferramenta arpspoof foi executada na Máquina Atacante_01 para fazer efetivamente um ataque homem-no-meio por envenenamento da tabela ARP do controlador e do comutador OpenFlow. O IP forwarding foi desativado na máquina Atacante_01 para que todo o tráfego termine lá. Pode-se afirmar com segurança que neste cenário a rede de produção pode ser afetada pela comunicação indesejada.

4.3. Negação de serviço da tabela de fluxos do comutador

A execução do ataque de negação de serviço pode ser feita preenchendo completamente a tabela de fluxos do comutador. Se o atacante preencher a tabela de fluxo com fluxos que não correspondam ao tráfego real da rede, qualquer novo cliente que queira usar a rede não será capaz de fazer isso. O controlador não conseguirá instalar quaisquer novos fluxos e o cliente terá efetivamente negado o uso da rede.

O sucesso deste ataque depende de dois parâmetros da rede SDN. Um é o tempo de validade dos fluxos que são instalados nos comutadores, configurado pelo administrador da rede. Se o controlador está configurado para instalar fluxos sem tempo limite de expiração, o ataque é fácilmente executado e o comutador terá uma tabela de fluxo cheia de fluxos falsos, independente do tempo necessário para o ataque. O segundo parâmetro que influencia no sucesso do ataque é a taxa de estabelecimento de fluxos. Se o controlador instala no comutador fluxos com um tempo limite curto, então o ataque fica mais difícil, pois é preciso preencher a tabela de fluxos antes que os fluxos comecem a expirar. Quando

há um tempo limite definido e os fluxos expiram após 60 ou 30 segundos, por exemplo, é preciso uma taxa bastante alta de inserção de novos fluxos.

4.3.1. Ataque partindo de usuário malicioso no plano de controle SDN

Este ataque teve como alvo o plano de controle da rede SDN, que conecta o controlador OpenFlow com o comutador OpenFlow.

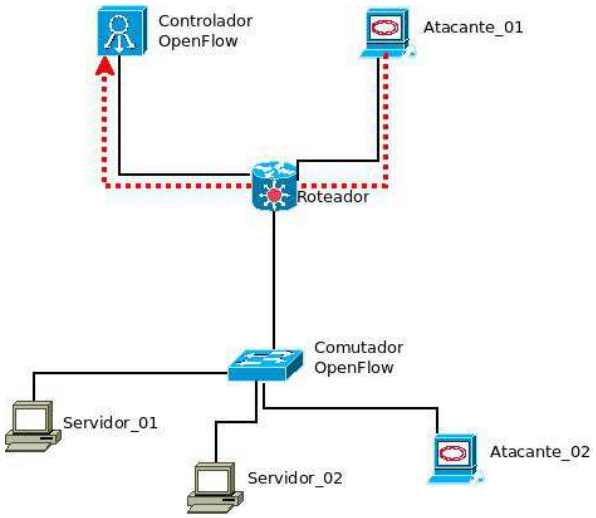


Figura 4. Negação de serviço - plano de controle

Esse cenário simula uma vulnerabilidade no controlador, permitindo que um atacante altere o plano de controle fazendo configurações diferentes das corretas. Foi implementado através da criação de um módulo para o controlador POX. Assim que uma conexão com o comutador é estabelecida, POX gera e instala modificações de fluxo para o comutador, a fim de preencher a tabela de fluxos. Este módulo registra uma função que é executada em um evento de início de conexão. A função registrada gera aleatoriamente endereços MAC, endereços IP e portas de origem e destino, que serão utilizados para criar um pacote de modificação de fluxos completo. A adição de fluxos se deu na mesma velocidade que o controlador foi capaz de gerá-los.

O resultado do ataque foi o esgotamento dos recursos do comutador, que se tornou incapaz de adicionar novos fluxos em sua tabela de fluxos por falta de memória.

4.3.2. Ataque partindo de usuário malicioso no plano de dados SDN

O ataque aqui descrito acontece no plano de dados de SDN.

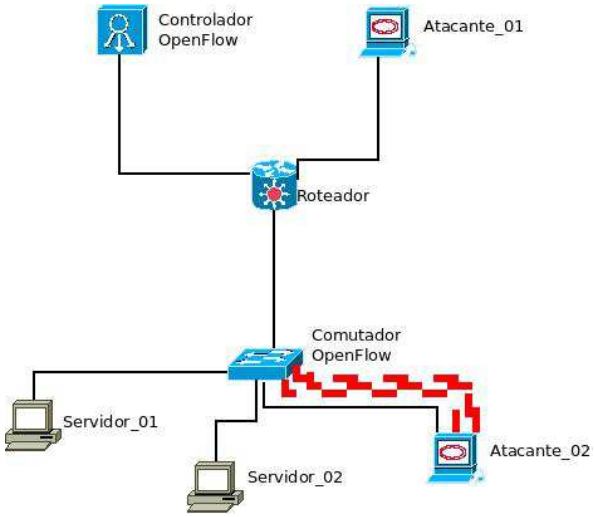


Figura 5. Negação de serviço - plano de dados

Foi usada a ferramenta Scapy. Conforme Bionde [2016] Scapy é um programa de manipulação interativa de pacotes, capaz de forjar ou decodificar pacotes de um grande número de protocolos, enviá-los e capturá-los. Usando esta ferramenta na máquina Atacante_02, foi possível enviar uma grande quantidade de pacotes ICMP com endereços IP diferentes, obrigando o controlador a instalar novos fluxos a cada solicitação de eco diferente recebida, pois os novos pacotes não correspondiam a nenhum dos fluxos existentes no comutador. Foi possível preencher a tabela de fluxos do comutador após certo tempo executando o ataque. Seus recursos foram esgotados.

4.3.3. Resultados dos ataques de negação de serviço

Tanto na rede de controle quanto na rede cliente SDN, os resultados são semelhantes em relação à quantidade possível de fluxos inseridos e o comportamento após a tabela de fluxo ficar cheia.

Um script monitorou a memória livre da máquina que executou a simulação, assim como a contagem de fluxos instalados no comutador SDN. Ficou claro que o Open vSwitch não considera as limitações de memória da máquina e não tem um número máximo de fluxos que pode armazenar. O comutador aceita fluxos enquanto houver memória livre. O número de fluxos instalados variou de acordo com a quantidade de aplicações em execução na máquina no momento dos testes, chegando a cerca de 600.000 fluxos. Quando a quantidade de memória livre ficou muito baixa, menos de 10 megabytes livres, o sistema operacional encerrou o processo Open vSwitch tornando o ataque de negação de serviços bem sucedido.

4.4. Ataque homem-no-meio

Outra possibilidade que um intruso tem na rede do controlador OpenFlow é a criação de portas espelhadas nos comutadores OpenFlow, inserindo ações extras quando ocorrem modificações de fluxo.

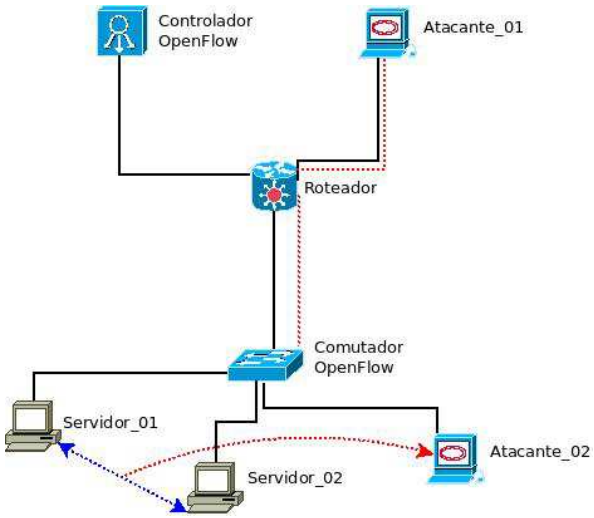


Figura 6. Ataque homem-no-meio

Ettercap é uma ferramenta para ataques homem no meio capaz monitorar conexões e filtrar conteúdo em tempo real, suportando dissecção ativa e passiva de muitos protocolos [Ettercap, 2016]. A ferramenta Ettercap foi utilizada para analisar pacotes capturados com comandos do controlador OpenFlow. A partir dessa análise, foi possível injetar pacortes de controle criando novas regras de envio de pacotes, espelhando o tráfego entre estações de trabalho legítimas na porta de uma estação de trabalho maliciosa.

4.5. Acesso à interface de depuração

Alguns comutadores OpenFlow disponibilizam uma interface de depuração. É uma conexão TCP em texto plano sem formatação que dá acesso ao plano de dados do comutador. Ela fornece ao usuário a capacidade de criar, modificar e apagar os caminhos de dados de comutação junto com a manipulação de fluxos na tabela de fluxo.

O comutador testado, Open vSwitch, não possui esse recurso. Portanto, não foi possível simular ataques explorando essa característica. Porém, qualquer conexão TCP em texto plano está sujeita a ataques clássicos, como homem-no-meio e espionagem. Assim, equipamentos onde esse recurso esteja disponível e habilitado estão sujeitos a tais ataques.

4.6. Testes com comutador real

Durante o desenvolvimento do presente trabalho, houve a tentativa de usar um comutador doméstico com sistema operacional modificado para suportar o protocolo

OpenFlow. O equipamento é um TP-LINK modelo WR842ND, um roteador sem fio que com cinco portas ethernet, quatro delas para rede local e uma para conexão com a internet.

O sistema operacional original foi substituido pelo sistema OpenWRT versão 15.05. OpenWrt é uma distribuição Linux para dispositivos embarcados com suporte à instalação do comutador Open vSwitch [OpenWRT team, 2016]. As portas ethernet físicas do roteador são associadas ao Open vSwitch para que este administre-as usando o protocolo OpenFlow, permitindo a troca de pacotes de rede entre elas. O comutador Open vSwitch recebe as informações de conexão com o controlador SDN, que assume então o controle das portas físicas do comutador.

Durante os experimentos, não houve troca de pacotes entre as portas físicas do comutador. A configuração seguiu a documentação disponível. Isso inviabilizou qualquer experimento proposto pelo presente trabalho.

4.7. Considerações

As simulações permitiram mostrar que fragilidades expostas pelo protocolo OpenFlow podem ser exploradas através de técnicas de ataque conhecidas, como homem-no-meio e inundação de pacotes. A não obrigatoriedade do uso de TLS no canal de comunicações entre controlador e comutador exibe um desleixo com a segurança na especificação do protocolo. Pode-se supor que seja intencional, talvez para baixar custo de fabricação de equipamentos com suporte a OpenFlow sem a necessidade de processadores mais poderosos.

Foi possível injetar pacotes de controle, comandando o comutador OpenFlow para que copiasse de tráfego de estações de trabalho legítimas em uma interface de rede de um cliente malicioso.

Além disso, a solução adotada em alguns comutadores OpenFlow físicos é executar internamente o comutador virtual Open vSwitch. Isso pode significar que as situações simuladas neste trabalho são transferidas para esses equipamentos praticamente sem nenhuma diferença em relação ao ambiente simulado.

5. Conclusão e trabalhos futuros

Neste trabalho foi proposta uma análise de possíveis ameaças de segurança em conseqüência da adoção do paradigma de redes SDN com protocolo OpenFlow. Foi usado um simulador de rede capaz de emular estações de trabalho, comutadores de controladores, além de máquinas reais conectadas a esse ambiente simulado.

Observou-se que as facilidades oferecidas pelo paradigma de SDN aos administradores de rede expõem vulnerabilidades oriundas de características da especificação do protocolo OpenFlow adotado pelo mercado na implementação dessas redes.

Os experimentos ilustram algumas dessas vulnerabilidades. As redes dependem fortemente de mensagens de packet-in, sendo expostas a ataques de negação de serviço contra comutadores e controladores. Pode se observar também que a segurança do canal de comunicações entre controlador e comutadores tem sido ignorada, tornando OpenFlow vulnerável a ataques homem-no-meio.

As simulações realizadas produziram resultados coerentes com o esperado quanto aos ataques de negação de serviço e homem-no-meio. Ficou evidenciado que um usuário malicioso pode prejudicar a operação de uma rede SDN e manipular seu conteúdo com técnicas amplamente conhecidas.

Como possíveis trabalhos futuros pode-se apontar:

- Uso de comutadores físicos de mercado para execução de simulações de ataques semelhantes às propostas neste trabalho
- Estudo comparativo entre resultados obtidos em ambientes simulados e físicos
- Estudo comparativo entre equipamentos comerciais quanto a aspectos de segurança
- Survey de ferramentas disponíveis para oferecer segurança a SDN
- Análise de outros protocolos adotados para a implementação de SDN, do ponto de vista de segurança de redes.

Referências

Amin, A. (2016). Vulnerabilities of Openflow Network and Network Security for SDN Network. *Global Journal For Research Analysis*, **5**(5).

Benton, K., Camp, L. J., e Small, C. (2013). Openflow vulnerability assessment. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 151 – 152. ACM.

Bionde, P. (2016). Scapy.

Brandt, M., Khondoker, R., Marx, R., e Bayarou, K. (2014). Security Analysis of Software Defined Networking ProtocolsâOpenFlow, OF-Config and OVSDB. In *The 2014 IEEE Fifth International Conference on Communications and Electronics (ICCE 2014), DA NANG, Vietnam*.

da Costa, V. T., Costa, L. H. M. K., Duarte, O. C. M. B., e Resende Junior, F. G. V. (2014). Controle e Isolamento de Recursos em Ambientes de Redes Virtuais OpenFlow.

de Oliveira, R. L. S., Schweitzer, C. M., Shinoda, A. A., e Prete, L. R. (2014). Using mininet for emulation and prototyping software-defined networks. In *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, pages 1 – 6. IEEE.

Ettercap (2016). Ettercap Home Page.

Feamster, N., Rexford, J., e Zegura, E. (2014). The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, **44**(2), 87 – 98.

Foster, N., Guha, A., Reitblatt, M., Story, A., Freedman, M. J., Katta, N. P., Monsanto, C., Reich, J., Rexford, J., Schlesinger, C., *et al.* (2013). Languages for software-defined networks. *Communications Magazine, IEEE*, **51**(2), 128 – 134.

Foundation, L. (2016). Open vSwitch.

Foundation, O. N. (2012). OpenFlow Switch Specification Version 1.3.0.

Heller, B. (2009). OpenFlow Switch Specification Version 1.0.0.

Hu, F., Hao, Q., e Bao, K. (2014). A survey on software-defined network and openflow: from concept to implementation. *IEEE Communications Surveys & Tutorials*, **16**(4), 2181 – 2206.

IronGeek (2016). Manual Page - arpspoof.

Javid, T., Riaz, T., e Rasheed, A. (2014). A layer 2 firewall for software defined network. In *Information Assurance and Cyber Security (CIACS), 2014 Conference on*, pages 39 – 42. IEEE.

Kim, H. e Feamster, N. (2013). Improving network management with software defined networking. *IEEE Communications Magazine*, **51**(2), 114 – 119.

Kreutz, D., Ramos, F., e Verissimo, P. (2013). Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 55 – 60. ACM.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., e Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, **103**(1), 14 – 76.

Marcondes, A. N. (2016). *Um Mecanismo para Gerência de Segurança da Autenticação e Controle de Acesso em redes SDN*. Master's thesis.

Mattos, D. M. F. e Duarte, O. C. M. B. (2014). AuthFlow: Um Mecanismo de Autenticação e Controle de Acesso para Redes Definidas por Software.

Mininet (2016). Mininet - An Instant Virtual Network on your Laptop (or other PC).

OpenWRT team (2016). OpenWRT Wireless Freedom.

Scott-Hayward, S., Natarajan, S., e Sezer, S. (2015). A survey of security in software defined networks. *IEEE Communications Surveys & Tutorials*, **18**(1), 623 – 654.

Souza, O. O. e Nobre, J. C. (2016). Autenticação de Estações de Trabalho em Redes Definidas por Software com Utilização de Certificados Auto-Assinados.