Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Um honeypot-cluster baseado em SDN e Raspberry Pi contra ameaças de segurança

Davi de Menezes Pereira

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Daniel Macêdo Batista

São Paulo
2022

Agradecimentos

Agradeço à minha família e amigos pelo apoio que me deram e pelas experiências que me proporcionaram durante a graduação. Agradeço também ao professor Daniel Batista, pelo apoio em 2021, quando fui seu orientando de iniciação científica, e em 2022 na construção deste projeto.

Resumo

Davi de Menezes Pereira. **Um honeypot-cluster baseado em SDN e Raspberry Pi contra ameaças de segurança**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2022.

O grande volume de dados trafegando em redes de comunicação tem exigido plataformas de computação de alto desempenho para detecção, o mais rápido possível, de ameaças à segurança da informação. Entretanto, o custo dessas plataformas é um dos fatores que limitam a sua utilização. Realizar parte do processamento mais próximo dos dados, com plataformas de baixo custo pode levar a uma boa eficácia de custos. Além disso, a maior dinamicidade exigida pelos sistemas atuais sugere que uma arquitetura mais flexível, como as de redes definidas por software, sejam uma arquitetura vantajosa. Neste sentido, esta monografia apresenta uma arquitetura de cluster baseada em Raspberry Pi e redes definidas por software para ser usada na execução escalável de componentes de um sistema de detecção de ameaças à segurança da informação. A análise de desempenho mostra que há pontos positivos na arquitetura proposta.

Palavras-chave: SDN. Honeypot. Raspberry Pi. Cluster. Segurança. OpenFlow. Balanceamento de Carga.

Abstract

Davi de Menezes Pereira. **A SDN and Raspberry Pi-based honeypot cluster against security threats**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2022.

The large volume of data being transferred in communication networks has required high-performance computing platforms to detect, as fast as possible, threats to information security. However, the cost of these platforms is one of the factors that limit their use. Performing part of the processing closer to the data, with low-cost platforms can lead to a good cost-benefit ratio. In addition, the dynamicity required by current systems suggests that a more flexible architecture, such as software defined networks, is an advantageous architecture. In this sense, this monography proposes a cluster architecture based on Raspberry Pi and software defined networks to be used in the scalable execution of components of an information security threat detection system. Performance analysis shows that there are positive points in the proposed architecture.

Keywords: SDN. Honeypot. Raspberry Pi. Cluster. Security. OpenFlow. Load Balance.

Lista de abreviaturas

CCSL	Centro de Competência em Software Livre
CPU	*Central Processing Unit*
DDoS	*Distributed Denial of Service*
DHCP	*Dynamic Host Configuration Protocol*
DoS	*Denial of Service*
FAPESP	Fundação de Amparo à Pesquisa do Estado de São Paulo
HIHAT	High Interaction Honeypot Analysis Toolkit
HTTP	*Hypertext Transfer Protocol*
IME	Instituto de Matemática e Estatística
IoT	Internet das Coisas (*Internet of Things*)
IP	*Internet Protocol*
OSI	*Open System Interconnection*
SBC	Computador de placa única (*Single Board Computer*)
SDN	Rede definida por software (*Software Defined Network*)
SQL	*Structured Query Language*
TCP	Transmission Control Protocol
URL	Localizador Uniforme de Recursos (*Uniform Resource Locator*)
USP	Universidade de São Paulo

Lista de figuras

Lista de programas

Sumário

Introdução

Contexto

A quantidade crescente de aplicações web com diversos elementos interligados, como aquelas em cidades inteligentes e em Internet das Coisas (IoT), leva a um interesse cada vez maior na melhoria de sistemas, observado pelo crescente investimento em dispositivos globalmente conectados e em pesquisa na área de IoT, por exemplo. Porém, fatos recentes demonstram que esses sistemas e dispositivos não recebem a devida atenção quando o assunto é segurança da informação. Apesar de já existirem serviços e propostas com a intenção de atuar nesse problema, eles podem ser muito caros. Com isso, surge a necessidade da criação de métodos e dispositivos que sejam mais baratos no monitoramento e estudo de ameaças a esses sistemas. Além disso, a crescente dinamicidade desses sistemas faz com que eles precisem ser cada vez mais flexíveis a adaptáveis.

Trabalhos relacionados

Com essa demanda, é possível notar esforços, tanto na academia quanto na indústria, na busca por soluções para o problema.

Dentre eles, é possível citar: GT-BIS Campiolo *et al.*, 2018GT-BIS, 2018, que teve por objetivo o desenvolvimento de um sistema para análise de quantidades massivas de dados heterogêneos capturados em redes de computadores a fim de detectar incidentes de segurança, além de trabalhos que utilizaram clusters de Raspberry Pi no monitoramento de ataques: Jeremiah, 2019 e Tripathi e Kumar, 2018, que sugerem a utilização de Raspberry Pi como componente de um sistema de detecção de ameaças. Djanali *et al.*, 2014a, que sugere que um honeypot-cluster composto de Raspberry Pis é uma boa alternativa na detecção de ataques de injeção SQL. No contexto desses projetos, os estudos sugerem que o uso de plataformas de computação de baixo custo, em especial SBC (*single board computers*) como o Raspberry Pi, é uma boa alternativa para compor um sistema de detecção de ameaças. Outros trabalhos, como Zarca *et al.*, 2020, sugerem que *HoneyNets*, isto é, redes de honeypots, baseadas em SDN fornecem uma arquitetura flexível no combate a ciberataques, no contexto de aplicações em IoT.

Objetivo

Partindo das informações acima, este projeto visa desenvolver um honeypot a partir de um cluster composto por Raspberry Pis com o objetivo de monitorar e estudar acessos suspeitos em um servidor, tomando como base sistemas existentes de detecção de ameaças. Os tipos de ataques abordados neste trabalho serão de negação de serviço e de injeção SQL (*SQL injection*). Além disso, pode ser vantajoso que para cada um desses ataques o sistema possua uma configuração diferente. Será visto nos capítulos seguintes que o sistema final de testes não fez diferenciação entre os tipos de ataque, mas ainda é válido comentar sobre esses tipos de ataques, uma vez que a arquitetura SDN apresenta uma vantagem por sua maior flexibilidade em relação a mudanças na arquitetura da rede. Por exemplo, para ataques de negação de serviço, podem ser necessários 5 computadores, enquanto para o outro tipo de ataque pode ser necessário somente um. Pensando nesta situação, surge a necessidade de utilização de um balanceamento de carga para dar vazão ao ataque sem o atacante perceber. Um destaque desta proposta é o fato de que ela foi desenvolvida num ambiente de rede definida por software (SDN), o que exige diversas tomadas de decisão em torno da implementação de um controlador para a rede.

De forma geral, o trabalho explora principalmente conceitos aprendidos nas matérias Redes de Computadores e Sistemas Distribuídos (MAC0352) e Programação Concorrente e Paralela (MAC0219). As duas principais contribuições relevantes deste trabalho são o passo a passo e documentação da construção do ambiente do honeypot e a implementação do controlador SDN que permitirá a migração do tráfego sem alertar o atacante de que ele está sendo monitorado.

Estrutura da Monografia

Esta monografia está dividida em quatro partes: no Capítulo 1 são discutidos os conceitos básicos de redes tradicionais de computadores, redes definidas por software, segurança em redes, balanceamento de carga, honeypots e computadores de placa única; no Capítulo 2, é mostrado o processo de construção e implementação da solução discutida nesta introdução, utilizando-se os conceitos abordados no capítulo anterior; no Capítulo 3, é mostrado o resultado, alguns testes e a análise desses testes; no Capítulo 4 estão as conclusões e propostas de trabalhos futuros.

Capítulo 1

Conceitos Básicos

Esta seção descreve os conceitos básicos de *software* e *hardware* mais importantes para a compreensão do trabalho realizado.

1.1 Ataques contra aplicações Web

Ataques contra aplicações Web tentam explorar falhas nas diversas camadas da arquitetura da internet a fim de derrubar, ou conseguir acesso a algum tipo de informação, ou ferramenta, não projetada para este fim. Como usam o protocolo HTTP como base, essas ameaças podem levar à negação do serviço (DoS), ao acesso indevido a informação ou mesmo à execução de comandos arbitrários do lado do servidor. Além disso, por conta do HTTP ser um protocolo sem estado, muitas aplicações precisam da utilização de algum sistema gerenciador de banco de dados para armazenar informações, o que acaba sendo outro alvo para esses ataques.

Os tipos de ameaça de segurança que estamos considerando neste trabalho são os ataques de negação de serviço (DoS) e ataques de *SQL injection*.

Os do primeiro tipo são aqueles que causam um excesso de entradas nos *logs* do servidor. Dentre essas ameaças específicas, as mais comuns são as do tipo DoS (*Denial of Service*), que visam atacar um serviço com um excesso de acessos simultâneos, por vezes efetuados por diversos agentes diferentes. Neste último caso, o ataque recebe o nome de DDoS (*Distributed Denial of Service*). Os do segundo tipo aproveitam falhas em sistemas que interagem com bancos de dados SQL em que o atacante consegue inserir uma instrução SQL personalizada numa consulta na entrada da aplicação.

1.2 Redes Definidas por Software (SDN)

Nas redes tradicionais, a arquitetura é definida apenas por dispositivos físicos, onde os planos de encaminhamento de dados e controle de redes são inseparáveis, de forma que as regras de encaminhamento de dados dos dispositivos são gerenciadas por uma camada de controle criada pelo fabricante do aparelho, não abrindo a possibilidade de alteração

pelos administradores da rede. Essa falta de flexibilidade na configuração da rede ocasiona, por exemplo, que diversos aparelhos numa mesma rede possuam diversas configurações diferentes e, por vezes, incompatíveis entre si.

Rede definida por software (SDN) é outra abordagem de infraestrutura que abstrai os recursos de rede para um sistema virtualizado (RedHat, 2022). Ela separa as funções de encaminhamento de dados e de controle de rede para criar uma rede que possa ser gerenciada e programada de maneira central. Nessa configuração, as equipes de operações de TI controlam o tráfego de rede em topologias complexas por meio de um painel centralizado. Assim, elas não precisam gerenciar cada dispositivo de rede manualmente RedHat, 2022.

O servidor dedicado onde fica a definição dos encaminhamentos é o responsável por adicionar regras de fluxo de pacotes em cada dispositivo de rede, como permitir troca de dados para endereços IP específicos, impedir a passagem de pacotes de protocolos específicos ou bloquear o envio de dados para alguma porta. Esse desacoplamento traz vantagens tanto em abstração, pois permite testes de novos protocolos sem a necessidade do conhecimento do funcionamento interno dos dispositivos de rede, quanto em flexibilidade, em aplicações que mudam constantemente, facilitando a troca de configurações.

A arquitetura de SDN divide as redes em três camadas, conforme a Figura 1.1, que possuem tarefas específicas e que se comunicam apenas com sua camada adjacente: camada de aplicação, camada de controle e camada de dados.

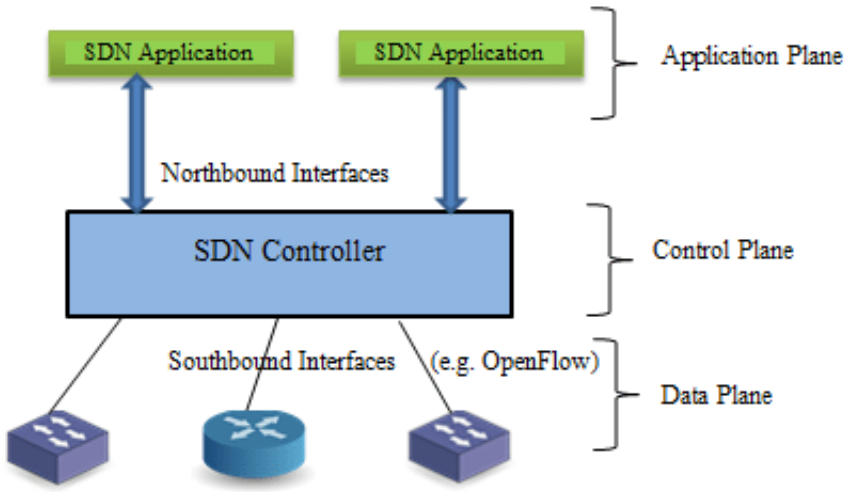

Figura 1.1: *Arquitetura SDN simplificada Mittal, 2018*

A primeira camada é a camada de aplicação. Nela estão as aplicações, as quais são os softwares que definem comportamentos dos fluxos da rede, passando essas informações para a camada mais de baixo. A próxima camada é a camada de controle, a qual recebe as instruções da camada de aplicação e manda para a camada de dados. Nela estão definidas, por software, as regras de encaminhamento da rede, o qual é o foco dessa abordagem. A última camada é a camada de dados (ou camada de infraestrutura), que recebe os dados e os processa conforme as instruções geradas na camada de aplicação. As comunicações entre as camadas mais baixas dessa arquitetura podem ser feitas seguindo um protocolo que permite o controle dos fluxos e das tabelas de fluxo dos dispositivos, como o OpenFlow.

1.2.1 Protocolo OpenFlow

A arquitetura OpenFlow é uma das mais conhecidas arquiteturas focadas em redes definidas por software. Ela define uma série de padrões que permitem o controle das chamadas tabelas de fluxos de cada dispositivo de encaminhamento. As tabelas de fluxos são responsáveis por guardar as informações do fluxo da rede, ou seja, são tabelas cujas entradas definem informações como regras de encaminhamento (algum valor específico do cabeçalho do pacote), ação (o que fazer caso a regra seja atendida) e estatísticas gerais sobre o fluxo.

Além disso, a arquitetura OpenFlow conta com um controlador, que está na camada de controle citada anteriormente, além de um canal seguro onde ocorre a comunicação entre o controlador e os dispositivos e um protocolo bem definido para essa comunicação, que tem o mesmo nome da arquitetura: OpenFlow.

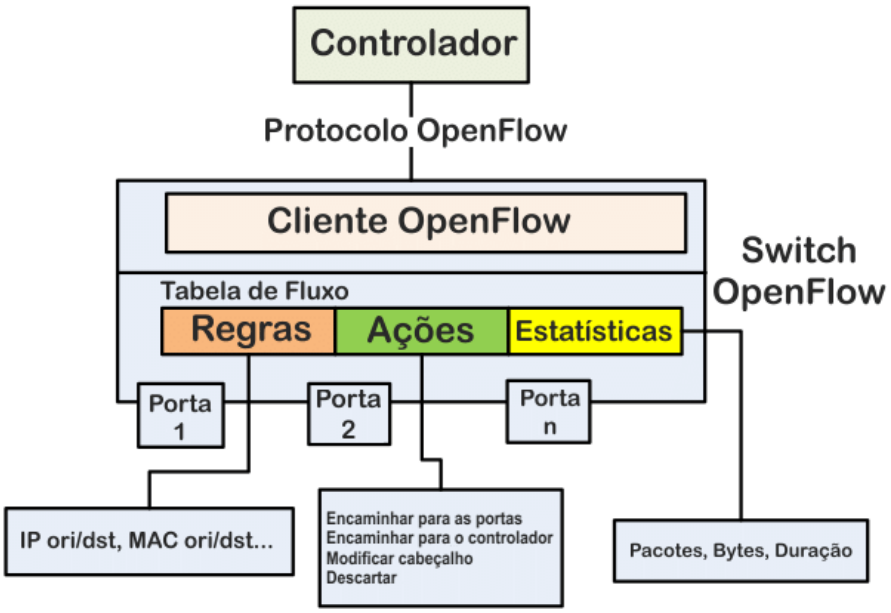

Figura 1.2: *Arquitetura OpenFlow* Joab Silva, *2014*

No diagrama da Figura 1.2, é possível encontrar os principais elementos da arquitetura OpenFlow. O *switch* OpenFlow é o dispositivo responsável pela comunicação da rede. Neles estão incluídos as tabelas de fluxos e o canal seguro, com o plano de dados. O controlador é quem gerencia, remotamente, os encaminhamentos de pacotes. A comunicação entre controlador e *switches* é feita pelo canal seguro através do protocolo OpenFlow. O pacote chega no dispositivo de encaminhamento, o qual verifica se existe algum fluxo cuja regra se encaixa com o pacote. Em caso afirmativo, a ação é executada com esse pacote, caso negativo o pacote é enviado para o controlador e esse permite que as aplicações de rede decidam o que fazer com esse pacote, adicionando um pequeno *overhead* na transmissão de dados. Quando essa nova decisão é tomada, ela é armazenada nos dispositivos de encaminhamento para evitar um tráfego intenso com o controlador e lá permanece por um determinado intervalo de tempo.

1.2.2 Controlador

Os controladores são os dispositivos que estão na camada de controle que, como explicado anteriormente, é onde são definidas as regras de encaminhamento dos pacotes. Neles está a lógica que define como as tabelas de fluxos dos dispositivos de encaminhamento devem ser construídas, abstraindo esse conceito para as aplicações e facilitando o desenvolvimento de novos comportamentos para as redes.

Existem diversos tipos de controladores, em diversos tipos de linguagens, como o NOX, desenvolvido em C++ e Python ou o Beacon, desenvolvido em Java, que permite a utilização na plataforma Android. Entretanto, este trabalho terá como foco um terceiro controlador chamado POX, escrito em Python, por conta de sua simplicidade e familiaridade do autor com a linguagem de programação.

1.3 Balanceamento de carga

Balanceamento de carga é uma técnica bastante conhecida em computação que tenta contornar limites de hardware de uma máquina individual através da distribuição do serviço por diversas máquinas. A carga que deseja-se balancear pode estar relacionada a diversos grupos: utilização de CPU, de armazenamento, ou de rede, por exemplo. Qualquer uma delas introduz o conceito de *clustering*, ou *server farm*, já que o balanceamento será, provavelmente, feito para vários servidores.

Em rede de computadores, geralmente o objetivo é balancear a carga de rede uniformemente entre dois ou mais computadores, enlaces de rede, discos rígidos ou outros recursos, a fim de trazer maior desempenho, tempo de resposta, ou maior confiabilidade através da redundância. As técnicas para isso consistem em, principalmente, reencaminhar o fluxo de dados por caminhos alternativos visando descongestionar os acessos aos servidores. Este balanceamento pode ocorrer a qualquer nível da camada OSI.

O balanceador pode ser implementado de várias formas, dependendo do serviço e arquitetura que se pretende balancear, e acaba servindo como uma interface entre quem está enviando os dados e os dispositivos de destino. Porém, numa abordagem tradicional, que exige o uso de hardwares de balanceamento, existem diversos problemas. O primeiro deles e o mais significativo é o custo muito elevado, tanto para adquirir um balanceador de última geração quanto para fazer a manutenção da rede física, como sugere o trabalho Carvalho Freitas Padilha Aguilar, 2018. Além disso, os dispositivos possuem um limite de servidores que podem ser conectados a eles, de modo a dificultar a escalabilidade do sistema, pois caso seja necessário adicionar mais máquinas que o limite, precisa-se adicionar mais um balanceador ou trocá-lo por um que tenha um limite maior, tornando o uso de hardware para essa tarefa pouco flexível a mudanças.

1.3.1 Balanceamento de carga numa SDN

A utilização de um dispositivo físico para a execução do balanceamento de carga tem um custo de implementação muito alto (o preço de bons dispositivos varia na casa dos milhares de dólares Carvalho Freitas Padilha Aguilar, 2018). Com isso, a aplicação de SDN como balanceador de carga diminui o custo e permite ao administrador maior

flexibilidade quando comparado a implementações diretas em hardware. A aplicação de SDN para resolver esse problema é bastante clara, já que a principal característica desses dispositivos é a compreensão do estado atual da rede interna e de como os servidores estão se comportando, exatamente o que a técnica de rede definida por software permite.

1.4 Cluster

Um cluster consiste em computadores conectados que trabalham em conjunto, de modo que podem ser considerados um único sistema, pois cada computador executa a mesma tarefa Wikipedia, 2022a.

1.5 Honeypot

Honeypot é um mecanismo de segurança em que o computador é configurado para detectar, desviar ou neutralizar tentativas de uso não autorizado de sistemas. Um honeypot consiste em um servidor que parece ser uma parte legítima do sistema, que contém informações atrativas para o atacante, sendo configurado para apresentar, propositalmente, falhas de segurança, mas que, na verdade, está isolado e controlado. Com isso, é possível monitorar e bloquear ou analisar invasores Wikipedia, 2022b.

Existem, principalmente, dois tipos principais de *design* para honeypots:

- Honeypots de produção
  - Servem como sistemas de isca dentro de redes e servidores totalmente operacionais, geralmente como parte de um sistema de detecção de intrusão.
  - Têm como objetivo desviar a atenção do usuário suspeito do sistema real, enquanto analisam atividades maliciosas para ajudar a mitigar vulnerabilidades.
- Honeypots de pesquisa
  - São usados para fim de pesquisa e aprimoramento de segurança.
  - Geralmente estão fora de algum tipo de rede de produção e coletam tipos de dados mais genéricos e rastreáveis que ajudam a analisar e entender o ataque.

O *design* escolhido para esse trabalho é o de honeypot de produção.

Além disso, existem três tipos principais de honeypots em relação ao comportamento:

- Honeypots puros
  - Sistemas de produção completos que monitoram ataques por *bug taps* no link que conecta o honeypot à rede. São considerados pouco sofisticados.
- Honeypots de baixa interação
  - Imitam serviços e sistemas que frequentemente atraem a atenção de ataques. Eles oferecem um método para coletar dados de ataques cegos, como *botnets* e

worms de *malware*.

- Honeypots de alta interação
  - Possuem configurações complexas que se comportam como infraestrutura de produção real. Eles não restringem o nível de atividade de um cibercriminoso, fornecendo informações abrangentes sobre segurança cibernética. No entanto, eles exigem maior manutenção e exigem experiência e o uso de tecnologias adicionais, como máquinas virtuais, para garantir que os invasores não possam acessar o sistema real.

O tipo de honeypot usado neste trabalho é o de alta interação.

1.6 Computadores de placa única (SBC)

Computadores de placa única, ou SBCs (*single board computers*), são computadores completos montados em apenas uma placa, ou seja, possuem processador, memória, entrada e saída de dados Ortmeyer, 2014. São vantajosos porque não têm um custo muito elevado, consomem pouca energia e, apesar de não serem tão potentes como computadores mais caros, podem ser utilizados para diversos fins, desde emuladores de vídeo game e *streaming boxes* até *desktops* simples com navegador web, editores de texto e até como nó de processamento em sistemas horizontalmente escaláveis, como em clusters. Um exemplo de computadores de placa única é o Raspberry Pi Raspberry Pi Foundation, 2021

Neste trabalho, o SBC escolhido foi o Raspberry Pi 3 model B+ Raspberry Pi Foundation, 2021. A Figura 1.3 apresenta a foto de uma das unidades que foi utilizada.

Figura 1.3: *Raspberry Pi 3 model B+*

Capítulo 2

Solução Baseada em SDN e SBC para honeypot

Este capítulo discute a implementação de um honeypot em uma SDN a partir de um cluster de Raspberry Pi's.

Parte do material utilizado para construir o cluster e o ambiente foram adquiridos durante o projeto da FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) Davi de Menezes Pereira, 2021 e o ambiente foi montado na sala 228 do Centro de Competência em Software Livre do Instituto de Matemática e Estatística da USP.

O sistema foi construído para simular um sistema que detecta requisições maliciosas em uma aplicação web e encaminha essas requisições para a aplicação ou para um honeypot que simula a aplicação.

2.1 Ferramentas de software utilizadas

2.1.1 Honeypot HIHAT

O *High Interaction Honeypot Analysis Toolkit* (HIHAT) é uma ferramenta *open source* que permite, de forma simples, transformar aplicações PHP em honeypots interativos. Ele funciona adicionando algumas linhas de código a cada página do *script* PHP, visando guardar detalhes de cada requisição ao banco de dados, enquanto não altera as principais funcionalidades da aplicação. Além disso, como mostra a Figura 2.1, o HIHAT é integrado com uma interface web que facilita a visualização dos dados pelo servidor honeypot.

Alguns trabalhos citados no início desta monografia sugerem que o HIHAT é uma boa opção por ser simples, mas robusto Müter *et al.*, 2008 Djanali *et al.*, 2014b.

2.1.2 MySQL

O MySQL é um sistema gerenciador de banco de dados relacional de código aberto usado na maioria das aplicações gratuitas para gerir suas bases de dados. O serviço utiliza a linguagem SQL (Structure Query Language – Linguagem de Consulta Estruturada), a

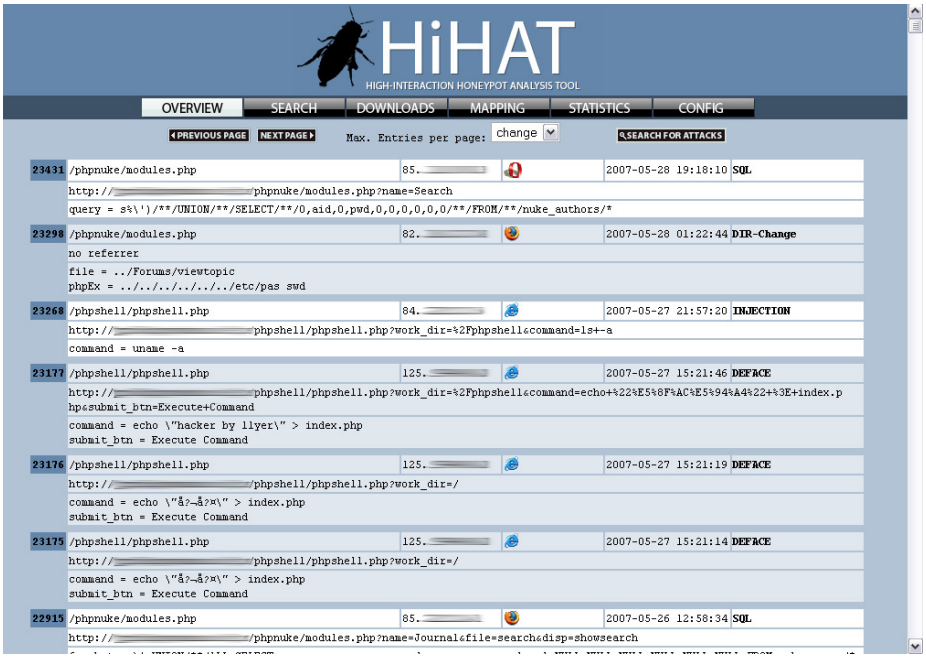

Figura 2.1: *Interface web do HIHAT*

qual é a linguagem mais popular para inserir, acessar e gerenciar o conteúdo armazenado num banco de dados.

2.1.3 Controlador POX

POX é um controlador de rede escrito em Python McCauley, 2015. Sua principal aplicação é agir como um controlador OpenFlow, mas também pode funcionar como um *switch* OpenFlow. Atualmente suporta o OpenFlow 1.0 e possui suporte específico para as extensões Open vSwitch e Open Nicira.

As versões do POX são nomeadas. A partir da versão "gar", POX utiliza Python 3. Neste trabalho, foi utilizada a versão "*fangtooth*", que suporta Python 2, por problemas com o pacote de DNS(dns.py) das versões que utilizam Python 3 (os motivos detalhados estão descritos numa *issue* do Github disponível em https://github.com/noxrepo/pox/issues/251). Pode ser executado em Linux, Mac OS ou Windows, porém algumas *features* não estão disponíveis dependendo do sistema. Neste trabalho, foi executado em Linux, sendo o que possui mais *features*.

2.1.4 OpenvSwitch

OpenVSwitch é um *switch* OpenFlow que funciona como um *switch* virtual em ambientes virtuais, como o KVM , 2016.

2.2 Hardware utilizado

2.2.1 Máquina principal

A máquina principal está no mesmo ambiente físico dos SBCs e conectada ao mesmo *switch* físico. Possui as seguintes propriedades:

- Sistema Operacional: CentOS Linux 7 (Core) x86_64
- Kernel: 4.9.48
- Processador: Intel i7-6700K, 8 núcleos, 4,0 GHz
- Memória RAM: 622 MiB

2.2.2 Computadores de placa única

Como dito anteriormente, os computadores de placa única escolhidos foram do tipo Raspberry Pi. Todos, exceto um, possuem as seguintes propriedades:

- Modelo: Raspberry Pi 3 Model B+
- Sistema Operacional: Raspberry Pi OS 2021-01-11
- Kernel: 5.4
- Processador: Broadcom BCM2837B0 64bits ARM Cortex-A53 Quad-Core
- Memória RAM: 1 GB

Ao todo, foram utilizadas 10 Raspberries, as quais foram dadas os seguintes nomes:

- rasp1
- rasp2
- rasp3
- rasp4
- rasp5
- rasp6
- rasp8
- rasp9
- rasp10
- rasp11

A Raspberry que seria a rasp7 apresentou problemas durante o desenvolvimento do projeto. O rasp11 foi reaproveitado de um projeto anterior e possui as seguintes configurações:

- Modelo: Raspberry Pi 3 Model B
- Sistema Operacional: Raspberry Pi OS 2021-01-11
- Kernel: 5.4
- Processador: Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- Memória RAM: 1 GB

2.2.3 Switch

Para a realização deste trabalho, não foi utilizado um *switch* com suporte a Openflow embutido. Em vez disso, foi utilizado um *switch* convencional e o *switch* Openflow foi executado através do OpenVSwitch, explicado anteriormente. Seguem as especificações do *switch*:

- Interface: 16 portas RJ45 10/100/1000Mbps (Auto Negociação/Auto MDI/MDIX)
- Mídia de Rede: 10Base-T: UTP cabo categoria 3, 4, 5 (máximo 100m); 100Base-TX/1000Base-T: UTP cabo categoria 5, 5e cabo (máximo 100m)
- Capacidade de Comutação: 32Gbps
- Taxa de Encaminhamento de Pacotes: 23.8Mpps
- Tabela de Endereços MAC: 8K
- Método de Transferência: Store-and-Forward

2.2.4 Cabos

Para conectar a máquina principal e os Raspberries no *switch* físico, foram utilizados cabos de rede *Ethernet cat5e*.

2.3 Arquitetura inicialmente proposta

A Figura 2.2 descreve a ideia inicial de arquitetura do ambiente para a aplicação do honeypot. Os elementos são: os usuários, o *switch* virtual (que está na mesma máquina que a *bridge*), o *loadBalancer*, o Controlador POX (que está na mesma máquina do *loadBalancer*) e os Raspberries, onde estão a aplicação normal e o honeypot. Todas as máquinas representam os Raspberries, exceto a dos usuários.

O responsável por receber e por direcionar os pacotes para os servidores é o *switch* OpenFlow. O controlador é responsável por receber as informações do administrador e aplicá-las no *switch*, alterando a configuração do balanceador, além de selecionar qual servidor deve receber a próxima requisição.

Nesse ambiente, existem dois tipos principais de fluxo das requisições. Primeiramente, um usário não malicioso faz uma requisição para o *switch*, que checa no controlador em qual categoria ela se encontra. Se for uma requisição normal, o controlador envia um

comando para o *switch* para direcionar o usuário para a máquina que contém o servidor normal. Se for uma requisição maliciosa, existem duas possibilidades: ou é uma tentativa de ataque de *SQL injection*, caso em que basta um Raspberry para atender o usuário, ou é uma tentativa de ataque de negação de serviço, caso em que o usuário é redirecionado para o cluster, que atua como um honeypot.

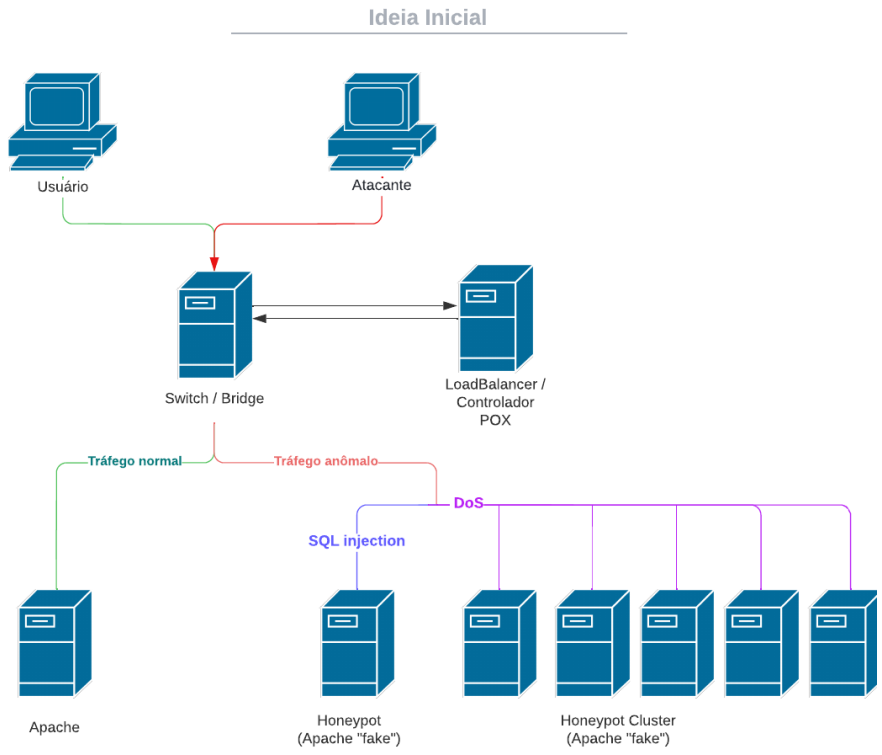

Figura 2.2: *Arquitetura inicialmente proposta*

2.4 Implementação do Honeypot

Como aplicação alvo do nosso sistema, foi criada uma página web simples em PHP contendo um formulário como mostrado na Figura 2.3. Para transformar essa aplicação em um honeypot, foram seguidos os passos descritos em https://hihat.sourceforge.net/installation.html.

2.5 Implementação da configuração de rede com os Raspberries

Os usuários e os Raspberries estão numa mesma rede local. Os Raspberries estão conectados fisicamente por um *switch* físico, assim como a máquina principal, *prestigio*, que representará os usuários no diagrama da Figura 2.2. Como os Raspberries devem estar numa SDN, uma das tarefas foi transformar um dos Raspberries, rasp9, em *switch* virtual

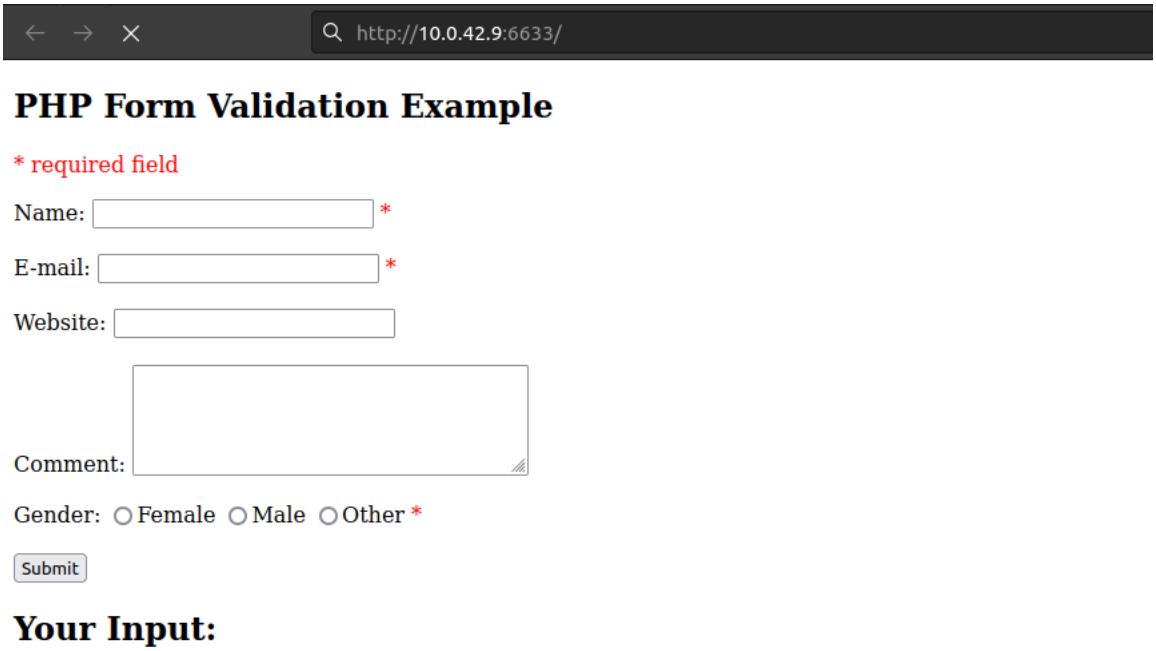

Figura 2.3: *Formulário simples em PHP utilizado como aplicação alvo nos testes*

Openflow. Além disso, parte da implementação foi feita remotamente, de forma que a *prestigio* era acessada via SSH, e, a partir daí, os Raspberries também foram acessados via SSH.

2.5.1 Configuração da rede SDN com os Raspberries

Primeiramente, no rasp9, foi feita a instalação do OpenVSwitch com um controlador de teste.

Programa 2.1 Instalação do OpenVSwitch

```
1   sudo apt-get install openvswitch-switch openvswitch-testcontroller
```

Para criar uma conexão entre os raspberries e fazer o fluxo de dados passar sempre pelo *switch* virtual (na rasp9), foi necessária a criação de uma *bridge*, que permite que máquinas em redes diferentes se reconheçam como se estivessem na mesma rede.

Programa 2.2 Criação da *bridge*

```
1   sudo ovs-vsctl add-br TCCBridge
2   sudo ovs-vsctl show #para checar se a bridge foi adicionada
3   sudo ifconfig TCCBridge up
```

Nesse momento, a interface TCCBridge apareceu na *interface configuration* (/sbin/ifconfig).

Depois disso, o próximo passo foi configurar a interface de rede ethernet *eth0* para suportar a TCCBridge e configurá-a com o IP do rasp9. Como essa sequência de comandos seria realizada remotamente e essas alterações derrubariam a conexão, foi pensada na utilização de uma *screen* para executar um script que realiza tais alterações em segundo plano. O seguinte *script* foi criado:

Programa 2.3 Script para adicionar a *bridge* na interface *eth0*

```
1  #!/bin/bash
2
3  sudo ovs-vsctl add-port TCCBridge eth0
4  sudo ifconfig eth0 0
5  sudo ifconfig TCCBridge 10.42.0.9/16
6
7  exit 0
```

Depois da criação do script (por exemplo, com o nome script.sh), foram executados os seguintes comandos:

Programa 2.4 Execução do script *eth0*

```
1  screen -S script
2  chmod +x script.sh
3  sleep 60; script.sh &
4  exit #desconectar do SSH
```

Dentro de um minuto, foi tentada a conexão com o rasp9, mas isso não foi possível. Outra tentativa, dessa vez com o rasp10, foi realizada, mas sem sucesso. Com isso, foi feita uma investigação da causa do problema.

2.5.2 Investigação do motivo do erro da conexão com os Raspberries

Após conectar o rasp9 em um monitor e teclado, foi possível ver que tanto a TCCBridge, quanto a interface *eth0* foram configuradas com o IP 10.42.0.9, o que indicou que o problema estava com o comando "*sudo ifconfig eth0 0*". Após rodar esse comando manualmente, percebeu-se que a placa rodou sem IP enquanto executava o comando, mas que logo depois o sistema operacional do Raspberry fez ele retornar ao IP 10.42.0.9.

Com isso, decidiu-se alterar as configurações do DHCP de modo a alterar esse comportamento. Após alterar o arquivo /etc/dhcpcd.conf acrescentando-se a linha do *eth0* e alterando-se *eth0* por TCCBridge, foram executados os seguintes comandos:

Programa 2.5 Reinicialização do DCHP com a nova configuração

```
1  sudo /etc/init.d/dhcpcd restart
2  systemctl daemon-reload
```

Porém, na saída do comando ifconfig, apareceu uma interface ovs-system com o IP 10.42.0.9 e a interface *eth0* continuou com o IP 10.42.0.9. Comentar as linhas "*interface*"e "*static*"do dhcpcd.conf e reiniciar o rasp9 não alterou esse comportamento.

Depois disso, alterou-se o ifconfig com os seguintes comandos:

Programa 2.6 Alteração de interfaces no *ifconfig*

```
1  sudo /sbin/ifconfig ovs-system 0
2  sudo /sbin/ifconfig eth0 0
3  sudo /sbin/ifconfig TCCBridge 10.42.0.9/16
```

Após a *reboot* no rasp9 e reinicialização do dhcpcd com os comandos abaixo, a rede ficou com a configuração desejada até o momento.

Programa 2.7 Reinicialização do daemon do DHCP

```
1  sudo killall -9 dhcpcd
2  sudo update-rc.d -f dhcpcd remove
3  sudo ovs-vsctl add-port TCCBridge eth0
4  sudo ifconfig eth0 0
5  sudo ifconfig TCCBridge 10.42.0.9/16
```

2.5.3 Continuação da configuração da rede SDN com os Raspberries

Após a resolução do problema anterior, os próximo passo consiste em criar placas de redes virtuais no rasp9 e depois conectá-las na *bridge*.

Programa 2.8 Criação de placas de redes virtuais e alteração da *bridge*

```
1  sudo ip tuntap add mode tap vPort1
2  sudo ip tuntap add mode tap vPort2
3  sudo ifconfig vPort1 up
4  sudo ifconfig vPort2 up
5  sudo ovs-vsctl add-port TCCBridge vPort1
6  sudo ovs-vsctl add-port TCCBridge vPort2
7  sudo ovs-vsctl show # Apenas para confirmar que vPort1 e vPort2 estão na
       bridge
8  sudo ifconfig # Apenas par confirmar que vPort1 e vPort2 aparecem como
       interfaces de rede
```

Depois disso, foi necessário fazer com que as raspberries conectassem-se ao *switch* no rasp9. Então, primeiramente, foram executados os comandos listados em Programa 2.9, que permitem que o rasp9 repasse pacotes para os outros raspberries.

Já nas outras raspberries, foram executados os comandos listados em Programa 2.10 para que o roteador padrão deles fosse o rasp9.

Programa 2.9 *IP forwarding* no rasp9

```
1   sudo sysctl -w net.ipv4.ip_forward=1
```

Programa 2.10 Mudança do roteador padrão

```
1   sudo /sbin/route add default gw 10.42.0.9
2   sudo /sbin/route del default gw 10.42.1.1
```

Nesse ponto, os pacotes de todas as raspberries estavam passando pela rasp9. Isso foi confirmado pela execução do comando "traceroute www.google.com.br".

O próximo passo foi testar se o OpenVSwitch estava realmente conectado em algum controlador. Nas configurações do controlador de teste que foi utilizado, em /etc/default/openvswitch-testcontroller , foi necessária a alteração da linha "LISTEN="pssl: para "LISTEN="ptcp:. Os comandos abaixo reiniciam o controlador teste com a nova configuração e configuram o OpenVSwitch para conectar-se ao controlador teste.

Programa 2.11 Reinício do controlador teste com a nova configuração

```
1   sudo /etc/init.d/openvswitch-testcontroller restart
2   sudo ovs-vsctl set-controller TCCBridge tcp:10.42.0.9:6653 ptcp:
3   sudo ovs-ofctl show TCCBridge #Para checar se mostra que o controlador está
        realmente conectado
```

Pode-se checar se o controlador teste consegue enxergar os pacotes desse raspberry. Em outro Raspberry, rasp8, foi executado:

Programa 2.12 Comandos para testar o funcionamento do controlador

```
1   ping -c 10 www.google.com.br
2   traceroute www.google.com.br
```

Após isso, no rasp9 foi executado:

Programa 2.13 Comando para confirmar que os pacotes passavam pelo controlador

```
1   sudo ovs-ofctl dump-flows TCCBridge | grep 10.42.0.8
```

Após esse comando, regras de controle referentes a tráfego de pacotes no rasp8 apareceram na saída padrão.

Para garantir que a máquina prestigio ficasse com uma faixa de IP diferente da das raspberries e garantir que o tráfego entre eles iria passar pelo rasp9, foram executados os seguintes comandos:

Programa 2.14 Comandos executados nas outras raspberries para definir a rasp9 como roteador padrão

```
1  # Na prestigio
2  sudo /sbin/ifconfig enp0s31f6 192.168.1.1
3  sudo /sbin/route add -net 10.42.0.0 netmask 255.255.0.0 gw 192.168.1.9
4  # Na rasp9
5  sudo /sbin/ifconfig TCCBRidge:1 192.168.1.9
```

2.5.4 Utilização do controlador POX com o OpenVSwitch

Com a rede configurada com o OpenVSwitch, utilizando-se o ovs-testcontroller como controlador teste, o próximo passo foi instalar o controlador POX e configurar a rede para utilizá-lo.

Seguindo as instruções descritas em https://noxrepo.github.io/pox-doc/html/, foi instalado o controlador POX no rasp9. Utilizando-se a versão padrão disponível no repositório no GitHub do POX, que utiliza Python 3, foram realizados os primeiros testes com o POX seguindo os comandos (no rasp9):

Programa 2.15 Utilização do POX

```
1  sudo /etc/init.d/openvswitch-testcontroller stop
2  python3 pox/pox.py log.level --DEBUG misc.of_tutorial #inicializa por padrão
       na porta 6633
3  sudo ovs-vsctl set-controller TCCBridge tcp:10.42.0.9:6633 ptcp:
```

Esses comandos desligam o controlador teste, ligam o POX na porta 6633 e mudam o controlador para o controlador que está rodando na porta 6633, que é o POX. Após a realização de alguns testes utilizando-se o rasp8 e os programas *traceroute* e *ping*, foi constatado que o controlador não estava conseguindo receber os pacotes devido a algum erro no código. Após investigação, em um *post* no site *stackoverflow* (*https://stackoverflow.com/questions/64312971/typeerror-ord-expected-string-of-length-1-but-int-found-error-in-pox-control*), foi possível constatar que a *branch* do repositório que suporta Python 3 estava com problemas no pacote dns.py. Após mudar para a *branch fangtooth*, que suporta Python 2, o erro foi resolvido.

2.5.5 Utilização de um *loadBalancer* POX existente

Considerando que o foco da proposta está na configuração e arquitetura geral do ambiente, foi necessária a utilização por um controlador com opções de *loadBalancer* já existentes.

O próprio controlador padrão do POX já está configurado com um módulo de *loadBalancer*(https://github.com/noxrepo/pox/blob/gar-experimental/pox/misc/ip_loadbalancer.py), porém, em Carvalho Freitas Padilha Aguilar, 2018 foi feita uma melhoria desse módulo, adicionando-se algumas opções a mais nos algoritmos de balanceamento. Para aproveitar os resultados desse trabalho anterior e realizar mais testes comparando diversos algoritmos, foi feita a tentativa de utilizar esse balanceador já criado antes.

Programa 2.16 Tentativa para utilizar o controlador melhorado

```
1  ./pox.py controlador_do_leonardo.ip_loadbalancer
2  ip=10.42.0.9
3  algorithm=round-robin
4  servers=10.42.0.1,10.42.0.3,10.42.0.4
```

Porém, essa tentativa não foi bem sucedida. Devido a problemas de compatibilidade da versão feita no trabalho anterior e a versão escolhida neste trabalho, o controlador não conseguia encontrar os servidores e a conexão entre os raspberries caía.

Com isso, foi utilizado o módulo de *loadBalancer* padrão do próprio controlador POX.

Programa 2.17 Teste realizado com o controlador padrão e três raspberries como servidores

```
1  ./pox.py misc.ip_loadbalancer
2  ip=10.42.0.9
3  servers=10.42.0.1,10.42.0.3,10.42.0.4
```

2.5.6 Mudanças na arquitetura

Dados os problemas de configuração citados neste capítulo, a arquitetura final precisou ser alterada segundo a Figura 2.4, em relação ao fluxo dos dados.

Essa decisão foi tomada levando-se em conta problemas de tempo para realização das atividades, mas também de forma a considerar que os resultados, observações e documentação dos problemas do processo de configuração da rede já consistem de resultados válidos para a comunidade interessada em sistemas que utilizam as mesmas ferramentas para a produção de diversas atividades.

Com isso, a principal mudança na arquitetura está na classificação do tráfego. Na nova arquitetura, classificamos apenas em dois tipos de tráfego, normal e malicioso, e não em 3 tipos, como anteriormente. Além disso, a outra mudança está na detecção das ameaças. Ao invés de realizar a detecção por algum tipo de análise detalhada do tipo de requisição utilizando-se um modelo de classificação existente, escolheu-se simular essa classificação tomando-se como base algum critério mais arbitrário. Por exemplo, se a requisição vem de certo IP, então o controlador direciona o usuário para o servidor, caso contrário, o usuário é direcionado para o honeypot.

Em relação aos raspberries, o rasp9 foi o controlador, o rasp10 agiu como o servidor normal e as outras máquinas (rasp1, rasp2, rasp3, rasp5, rasp6, rasp8 e rasp11) serviram como honeypot.

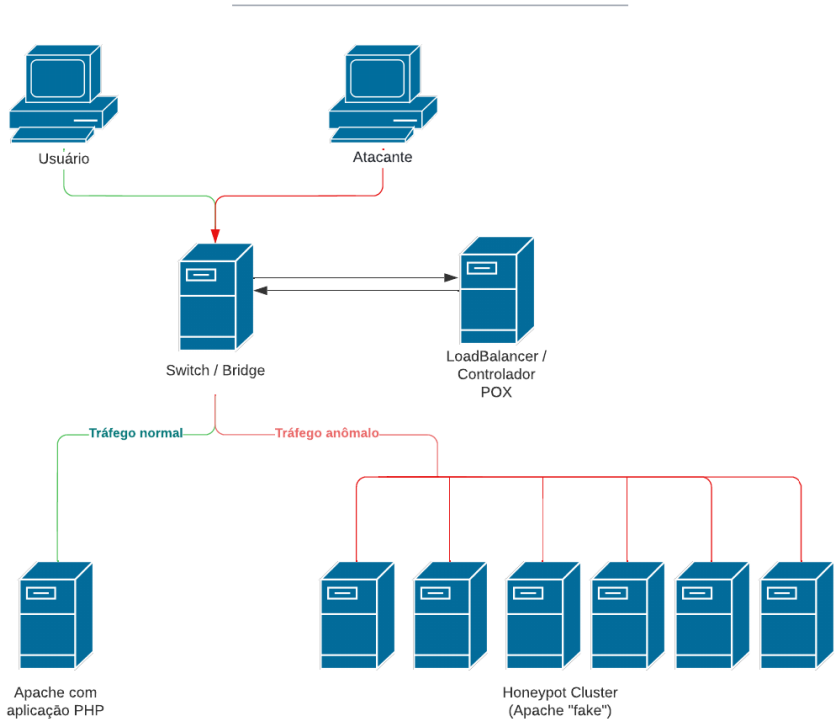

Figura 2.4: *Arquitetura final*

2.5.7 Controlador

Como explicado no tópico anterior, para fins de testar o ambiente, o controlador precisa classificar as requisições de algum modo arbitrário entre maliciosas e normais. Com isso, escolheu-se alternar as requisições entre maliciosas e normais a fim de testar os dois fluxos de dados de forma mais prática. Após classificar, o controlador direciona o pacote para o honeypot, isto é, para alguma das máquinas do honeypot utilizando-se o módulo *ip_loadbalancer*.

Para direcionar os pacotes, ou pacote iria para o *loadBalancer* (declarado no código como *iplb*), ou iria para o servidor real modificando-se os campos de destino do pacote, como mostrado no código em Programa 2.18.

Programa 2.18 Trecho do código para definição de destino do tráfego

```
1   # caso do malicioso
2   iplb._handle_PacketIn(msg)
3
4   # caso do normal
5   msg.action.append(of.ofp_action_nw_addr.set_dst(IPAddr("10.42.0.10")))
6   msg.actions.append(of.ofp_action_dl_addr.set_dst(EThAddr("00:00:00:00:00:10"))
        )
```

O código completo do controlador pode ser encontrado em https://github.com/davimp/

mac0499.

2.6 Ambiente real

O ambiente foi montado na sala 228CCSL do Instituto de Matemática e Estatística da Universidade de São Paulo como mostrado na Figura 2.5. Nessa foto, é possível ver a máquina principal (prestigio) e os raspberries conectados por cabos ethernet em um *switch*.

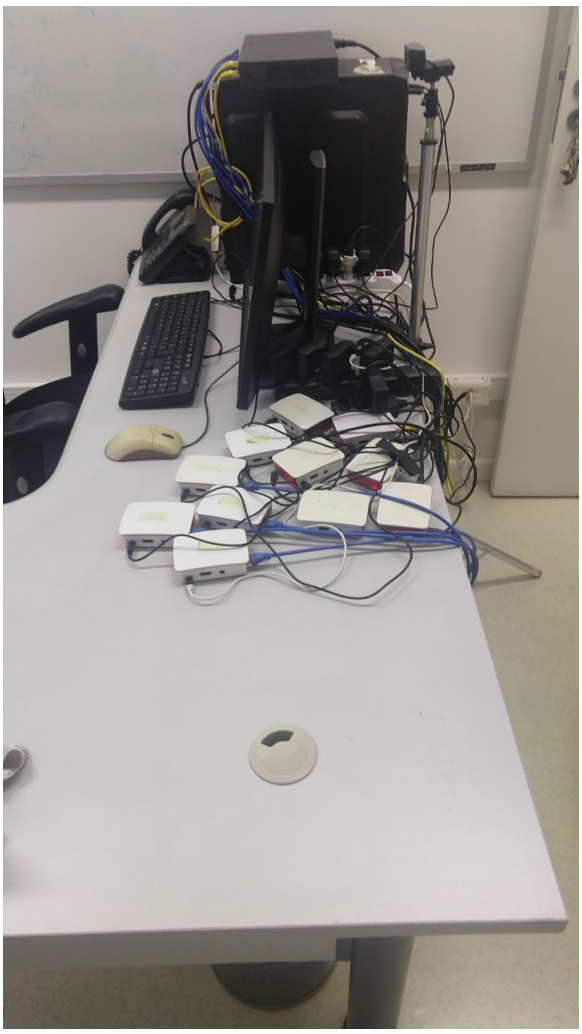

Figura 2.5: *Ambiente montado no bloco C do IME-USP*

Capítulo 3

Resultados e análise

Este capítulo é dedicado a discussão dos resultados obtidos no projeto, que consiste principalmente no funcionamento da construção do ambiente proposto.

3.1 Testes de capacidade

Inicialmente, a fim de avaliar a capacidade do cluster em termos de rede, foi utilizada a ferramenta *iperf*. No Máquina Principal (Figura 2.4) foi executado um servidor, e nos Raspberries foram executados clientes para medir o desempenho da rede em transferência de dados e em largura de banda. A quantidade de nós do cluster variou de 1 até 10.

A Figura 3.1 e a Figura 3.2 mostram os resultados desse experimento. Com era de se esperar, com o aumento dos nós há uma divisão da largura de banda entre esses nós.

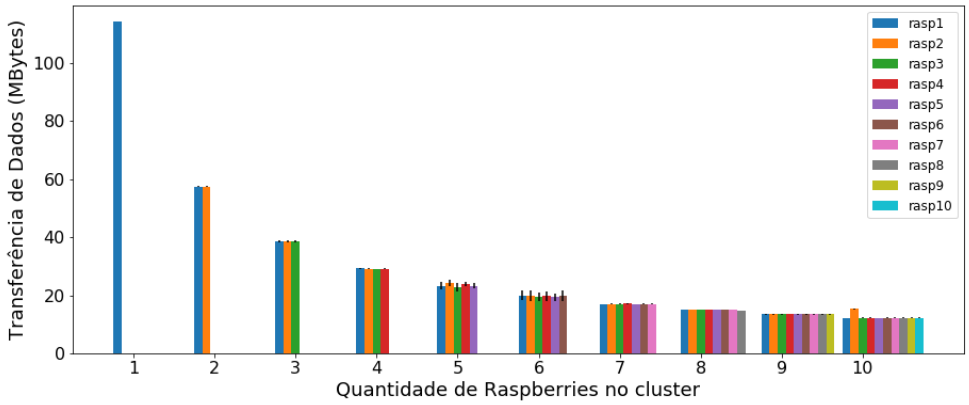

Figura 3.1: *Capacidade de rede em transferência de dados*

3.2 Testes do loadbalancer

A Figura 3.3, que mostra a porcentagem média de dados que cada Raspberry recebeu nos experimentos com 100 acessos direcionados ao honeypot, mostra que houve uma boa

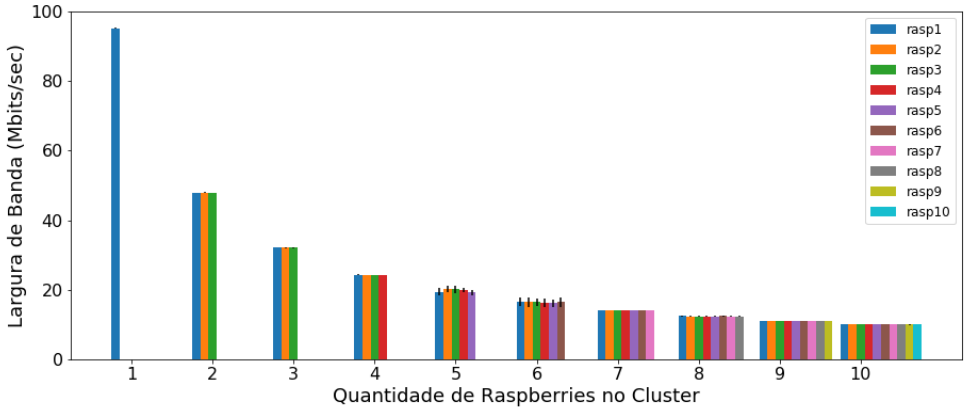

Figura 3.2: *Capacidade de rede em largura de banda*

distribuição de pacotes entre as máquinas feita pelo módulo de balanceamento padrão do controlador POX. Cada computador ficou com, aproximadamente, 14% dos pacotes.

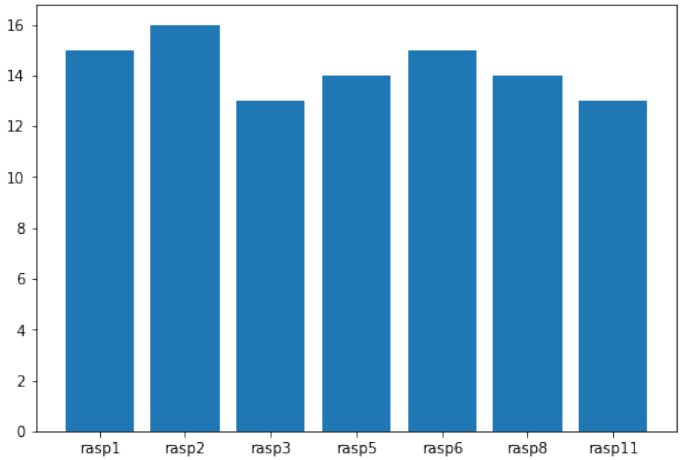

Figura 3.3: *Distribuição de pacotes pelo* loadbalancer *entre as raspberries do honeypot*

3.3 Funcionamento da aplicação

Como explicado anteriormente, os acessos foram alternadamente classificados em maliciosos ou normais. Para o usuário, ficou imperceptível a diferença entre acessar o honeypot e o servidor normal, uma vez que o tempo de resposta das aplicações era o mesmo. Isso pode ter acontecido devido à simplicidade da aplicação alvo, uma vez que era só um formulário, como também pela simplicidade da lógica executada pelo controlador.

Caso o acesso fosse classificado como malicioso, dados do acesso eram coletados pelo honeypot e armazenados num banco de dados SQL, o qual foi utilizado pelo HIHAT para a disposição de informações consideradas importantes sobre o ataque, que depois eram agrupadas numa tabela e dispositivos na interface do HIHAT.

3.3.1 Armazenamento das informações coletadas pelo honeypot

Como descrito no capítulo anterior, o honeypot armazena os dados coletados do usuário num banco de dados SQL num *database* chamado "honeyweb"(Figura 3.4), que contém duas tabelas: *main logs* e *binary tools* (Figura 3.5). A primeira contém os dados de acesso e a segunda contém ferramentas para a integração dos dados com a interface web do HIHAT. A Figura 3.6 mostra as colunas da tabela *main logs* e dá uma visão geral dos dados armazenados pelo honeypot.

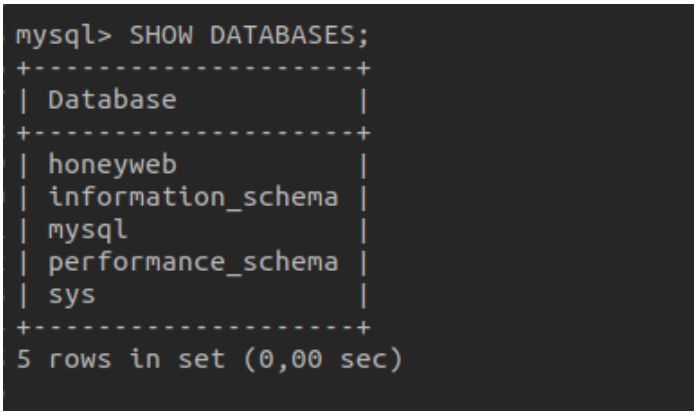

Figura 3.4: *Bases de dados do honeypot*

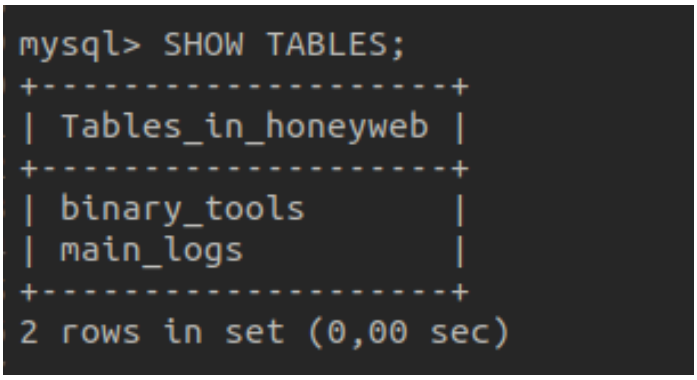

Figura 3.5: *Tabelas do honeypot (base de dados "honeyweb")*

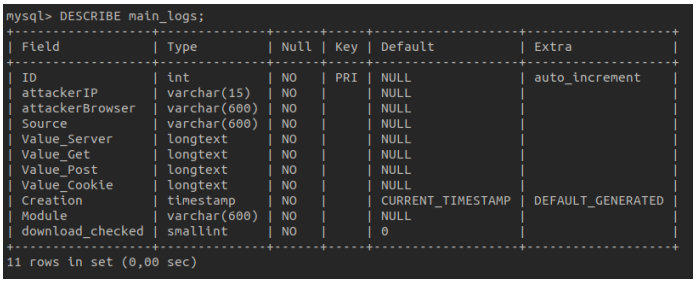

Figura 3.6: *Colunas da tabela* main logs *do honeypot*

Na Figura 3.7, é mostrado um exemplo de 5 acessos capturados pela honeypot e o valor

do método POST HTTP desses acessos. Como os acessos são alternadamente classificados entre maliciosos e normais, os acessos maliciosos são os com número ímpar.

Figura 3.7: *Conteúdo do banco de dados do honeypot exibindo os acessos maliciosos*

Capítulo 4

Conclusões

4.1 Cluster baseado em SDN e em SBCs

A proposta de arquitetura de *cluster* baseado em Raspberry Pi e SDN comportando-se como um honeypot apresentou-se viável quando colocado em ambientes que respeitem os limites de hardware do *cluster*. A configuração do sistema e construção do ambiente mostrou-se mais complicada do que o esperado, porém este trabalho também teve como proposta servir como um guia para trabalhos futuros que considerem a utilização de um sistema semelhante.

4.2 HIHAT

O honeypot HIHAT mostrou-se uma ferramenta muito simples e útil de ser utilizada, porém, como é possível ver em https://github.com/honeynet/HIHAT, em janeiro de 2019 o projeto foi descontinuado, portanto, não é recomendada a sua utilização em trabalhos futuros. Após mandar uma mensagem para o e-mail de suporte do projeto (hihatproject@gmail.com), foi confirmado que o projeto parou de ser mantido e atualizado, mas que os organizadores estariam abertos para receber futuras atualizações e melhoramentos e qualquer um poderia entrar em contato com eles caso estivesse interessado.

4.3 Controlador POX

O controlador POX, apesar de ser um dos mais utilizados atualmente, não apresentou utilização e suporte tão fácil quanto esperado inicialmente. Alguma de suas funções ainda estão com problemas de implementação e a documentação não é de fácil acesso.

4.4 Trabalhos Futuros

Para trabalhos futuros, existem diversas abordagens possíveis:

Em questão de hardware, é possível construir um sistema com outros tipos de SBCs e testar as capacidades desses novos sistemas em diferentes contextos. Também é possível a exploração de diferentes tipos de arranjo de hardware, por exemplo, a utilização de um Turing Pi (, s.d.) para um sistema mais compacto, eficiente e dinâmico (numa perspectiva de hardware).

Em relação à SDN, é possível testar outros controladores, como o Beacon (https://www.researchgate.net/publication/262152829_The_Beacon_OpenFlow_Controller) e comparar os desempenhos, e também a construção de controladores mais complexos, estudando-se o quanto isso afetaria no desempenho e experiência dos usuários da honeypot. Além disso, é possível a exploração de um sistema mais dinâmico de rede, que exija diferentes comportamentos dependendo do contexto, a fim de tirar vantagem das abstrações propostas por uma arquitetura SDN.

Referências

[2016] . *Open vSwitch*. https://www.openvswitch.org/. A Linux Foundation Collaborative Project, 2016 (citado na pg. 10).

[s.d.] . *Turing Pi*. https://turingpi.com/. Turing Machines Inc. (citado na pg. 28).

[Campiolo *et al.* 2018] R. Campiolo, L. A. F. Santos, W. A. Monteverde e D. M Batista. "Uma Arquitetura para Detecção de Ameaças Cibernéticas Baseada na Análise de Grandes Volumes de Dados". Em: *Workshops do SBRC – WSCDC*. 2018 (citado na pg. 1).

[Carvalho Freitas Padilha Aguilar 2018] Leonardo de Carvalho Freitas Padilha Aguilar. "Análise do uso de SDN para balanceamento de carga". Em: *Depositos de TCCs BCC IME USP*. 2018 (citado nas pgs. 6, 18).

[Davi de Menezes Pereira 2021] Davi de Menezes Pereira. *Processamento paralelo com plataformas de computação de baixo custo para análise de grandes volumes de dados de segurança*. https://bv.fapesp.br/pt/bolsas/195395/processamento-paralelo-com-plataformas-de-computacao-de-baixo-custo-para-analise-de-grandes-volumes-/. FAPESP, 2021 (citado na pg. 9).

[Djanali *et al.* 2014a] Supeno Djanali, FX Arunanto, Baskoro Adi Pratomo, Hudan Studiawan e Satrio Gita Nugraha. "Sql injection detection and prevention system with raspberry pi honeypot cluster for trapping attacker". Em: *2014 International Symposium on Technology Management and Emerging Technologies*. 2014, pgs. 163–166. doi: 10.1109/ISTMET.2014.6936499 (citado na pg. 1).

[Djanali *et al.* 2014b] Supeno Djanali, FX Arunanto, Baskoro Adi Pratomo, Hudan Studiawan e Satrio Gita Nugraha. "Sql injection detection and prevention system with raspberry pi honeypot cluster for trapping attacker". Em: *2014 International Symposium on Technology Management and Emerging Technologies*. 2014, pgs. 163–166. doi: 10.1109/ISTMET.2014.6936499 (citado na pg. 9).

[GT-BIS 2018] GT-BIS. *GT-BIS – Mecanismos para Análise de Big Data em Segurança da Informação*. http://gtbis.ime.usp.br/. Último acesso em 23/07/2021. GT-BIS, 2018 (citado na pg. 1).

[Jeremiah 2019] June Jeremiah. "Intrusion detection system to enhance network security using raspberry pi honeypot in kali linux". Em: *2019 International Conference on Cybersecurity (ICoCSec)*. 2019, pgs. 91–95. doi: 10.1109/ICoCSec47621.2019.8971117 (citado na pg. 1).

[Joab Silva 2014] Joab Silva. *Arquitetura OpenFlow*. https://www.researchgate.net/figure/Figura-24-Arquitetura-OpenFlow-Adaptado-de-Nunes-et-al-2014_fig3_324780227. Joab Silva, 2014 (citado na pg. 5).

[McCauley 2015] McCauley. *POX documentation*. https://noxrepo.github.io/pox-doc/html/. POX, 2015 (citado na pg. 10).

[Mittal 2018] Sangeeta Mittal. "Performance evaluation of openflow sdn controllers". Em: mar. de 2018, pgs. 913–923. isbn: 978-3-319-76347-7. doi: 10.1007/978-3-319-76348-4_87 (citado na pg. 4).

[Müter *et al.* 2008] Michael Müter, Felix Freiling, Thorsten Holz e Jeanna Matthews. "A generic toolkit for converting web applications into high-interaction honeypots". Em: (jan. de 2008) (citado na pg. 9).

[Ortmeyer 2014] Cliff Ortmeyer. "Then and Now: A Brief History of Single Board Computers". Em: *Electronic Design Uncovered, issue 06*. 2014, 1–4 (citado na pg. 8).

[Raspberry Pi Foundation 2021] Raspberry Pi Foundation. *Raspberry Pi 3 Model B+*. https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. Último acesso em 23/07/2021. Raspberry Pi Foundation, 2021 (citado na pg. 8).

[RedHat 2022] RedHat. *O que é rede definida por software?* https://www.redhat.com/pt-br/topics/hyperconverged-infrastructure/what-is-software-defined-networking. RedHat, 2022 (citado na pg. 4).

[Tripathi e Kumar 2018] Shyava Tripathi e Rishi Kumar. "Raspberry pi as an intrusion detection system, a honeypot and a packet analyzer". Em: *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*. 2018, pgs. 80–85. doi: 10.1109/CTEMS.2018.8769135 (citado na pg. 1).

[Wikipedia 2022a] Wikipedia. *Cluster (computing)*. https://simple.wikipedia.org/wiki/Cluster_(computing). Wikipedia, 2022 (citado na pg. 7).

[Wikipedia 2022b] Wikipedia. *Honeypot (computing)*. https://en.wikipedia.org/wiki/Honeypot_(computing). Wikipedia, 2022 (citado na pg. 7).

[Zarca *et al.* 2020] Alejandro Molina Zarca, Jorge Bernal Bernabe, Antonio Skarmeta e Jose M. Alcaraz Calero. "Virtual iot honeynets to mitigate cyberattacks in sdn/nfv-enabled iot networks". Em: *IEEE Journal on Selected Areas in Communications* 38.6 (2020), pgs. 1262–1277. doi: 10.1109/JSAC.2020.2986621 (citado na pg. 1).