UNIVERSIDADE D
COIMBRA

Paulo Pais

Mitigação de ataques HTTP/2 rapid reset através de mecanismos SDN

Dissertação no âmbito do Mestrado em Segurança Informática, orientada pelo Professor Tiago Cruz e Professor Bruno Sousa, e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho 2024

DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA
FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Paulo Pais

MITIGAÇÃO DE ATAQUES HTTP/2 RAPID RESET ATRAVÉS DE MECANISMOS SDN

Dissertação no âmbito do Mestrado em Segurança Informática, orientada pelo Professor Tiago Cruz e Professor Bruno Sousa, e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho 2024

DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA
FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Paulo Pais

HTTP/2 RAPID RESET ATTACK MITIGATION VIA SDN MECHANISMS

Dissertation in the context of the Masters in Informatics Security, advised by Professor Tiago Cruz and Professor Bruno Sousa, and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

Jully 2024

Agradecimentos

Mais uma jornada conquistada, olho para trás e reconheço que o caminho percorrido foi bastante desafiador mas enriquecedor apenas tornado possível pela presença e apoio de várias pessoas importantes na minha vida. É com profunda gratidão que dedico este espaço para agradecer a todos que contribuíram para a minha jornada académica e pessoal.

Em primeiro lugar, gostaria de expressar minha sincera gratidão aos meus professores pela sua disponibilidade. A cada um de vocês, que não só me guiaram com o seu conhecimento e sabedoria, mas também me inspiraram com sua paixão e dedicação, o meu mais profundo agradecimento. Vocês não foram apenas professores, mas verdadeiros mentores que me desafiaram a ir mais além e a explorar o vasto mundo do conhecimento com curiosidade e determinação. A paciência, o encorajamento e a orientação que recebi foram fundamentais para o meu crescimento académico e pessoal.

À minha família, meu eterno agradecimento. Vocês foram a minha fortaleza e o meu refúgio nos momentos mais desafiadores. A cada passo desta jornada, senti o calor do vosso amor incondicional e o apoio constante. Vocês acreditaram em mim até quando duvidei de mim mesmo, e essa confiança foi o vento sob as minhas asas que me permitiu voar mais alto. Agradeço por todos os sacrifícios feitos, pelas palavras de incentivo e por estarem sempre ao meu lado, celebrando cada pequena conquista e levantando-me em cada queda.

Aos meus amigos e colegas, agradeço pela camaradagem, pelos momentos de descontração e pelas discussões estimulantes. Vocês tornaram esta jornada académica não apenas educativa, mas também alegre e memorável.

Por fim, agradeço a todos que, direta ou indiretamente, contribuíram para a realização desta dissertação. Cada contribuição foi valiosa e ajudou a moldar tanto o trabalho final quanto a pessoa que me tornei.

Resumo

Esta dissertação investiga a mitigação de ataques *HTTP/2 rapid reset* utilizando mecanismos de Software Defined Networks (SDN). O estudo propõe o desenvolvimento de um ambiente de testes que integra controladores SDN, protocolos específicos e virtualização através do VMware. O objetivo é avaliar estratégias de mitigação contra ataques *rapid reset*, explorando a programabilidade e flexibilidade das SDN. A escolha da linguagem Python foi fundamental devido à sua simplicidade, robustez e ampla adoção na comunidade de redes e segurança cibernética.

A pesquisa detalha o processo de construção do ambiente de testes, que inclui a configuração dos controladores SDN, a simulação de dispositivos de rede virtuais e a integração com o VMware para criar um cenário de rede dinâmico e realista. Diversas estratégias de mitigação para o ataque *rapid reset* foram testadas. Os resultados indicam que os mecanismos de defesa implementados apresentaram tempos de mitigação baixos, assegurando uma resposta rápida e eficaz a potenciais ataques, sem comprometer a performance do sistema.

Este trabalho contribui significativamente para a área da segurança cibernética ao demonstrar como as características únicas das SDN, como a visibilidade abrangente da rede e a capacidade de reprogramação dinâmica, podem ser utilizadas para detetar, isolar e neutralizar ameaças de botnets de maneira eficaz. Adicionalmente, todos os *scripts* desenvolvidos foram disponibilizados no GitHub, facilitando a utilização e expansão pela comunidade. As conclusões desta dissertação oferecem uma base sólida para futuras pesquisas na aplicação de tecnologias SDN na segurança de redes.

Palavras-Chave

Botnets, SDN (Redes Definidas por Software): Deteção, Prevenção, DDoS, Mitigação de Ataques, IDS (Sistema de Deteção de Intrusão), IPS (Sistema de Prevenção de Intrusão)

Abstract

This dissertation investigates the mitigation of HTTP/2 rapid reset attacks using SDN mechanisms. The study proposes the development of a test environment that integrates SDN controllers, specific protocols, and virtualization through VMware. The objective is to evaluate mitigation strategies against rapid reset attacks, leveraging the programmability and flexibility of SDN. The choice of Python was essential due to its simplicity, robustness, and wide adoption in the networking and cybersecurity community.

The research details the process of building the test environment, which includes configuring SDN controllers, simulating virtual network devices, and integrating with VMware to create a dynamic and realistic network scenario. Various mitigation strategies for the rapid reset attack were tested, including the analysis of the effectiveness of different detection and response algorithms. The results indicate that the implemented defense mechanisms exhibited low mitigation times, ensuring a quick and effective response to potential attacks without compromising system performance.

This work significantly contributes to the field of cybersecurity by demonstrating how the unique characteristics of SDN, such as comprehensive network visibility and dynamic reprogramming capabilities, can be used to effectively detect, isolate, and neutralize botnet threats. Additionally, all developed scripts have been made available on GitHub, facilitating their use and expansion by the community. The conclusions of this dissertation provide a solid foundation for future research in the application of SDN technologies in network security.

Keywords

Botnets, SDN (Software-Defined Networking), Detection, Prevention, DDoS, Attack Mitigation, IDS (Intrusion Detection System), IPS (Intrusion Prevention System)

Conteúdo

Acrónimos

DDoS Distributed Denial-of-Service Attack.

DSR Design Science Research.

DTLS Datagram Transport Layer Security.

IDS Intrusion Detection System.

IoT Internet of Things.

IPS Intrusion prevention system.

NAT Network Address Translation.

NFV Network Functions Virtualization.

ODL OpenDaylight.

ONF Open Networking Foundation.

ONOS Open Network Operating System.

OSGi Open Services Gateway initiative.

OSI open systems interconnection.

OSPF Open Shortest Path First.

P4-int Programming Protocol-independent Packet Processors.

QOTD Quote of the Day.

QUIC Quick UDP Internet Connections.

RPS requests per second.

SDN Software Defined Networks.

SNMP Simple Network Management Protocol.

STP Spanning Tree Protocol.

VM Virtual Machine.

Lista de Figuras

Capítulo 1

Introdução

Como sabemos, a sociedade da informação permitiu a disseminação da informação a toda a sociedade através das novas tecnologias. De facto, o acesso à informação quebrou as barreiras físicas e temporais, pois tudo passou a estar acessível a qualquer hora e a partir de qualquer lugar com apenas um *click*. À medida que a sociedade da informação foi evoluindo, foram surgindo novos problemas e desafios nomeadamente no que respeita à segurança da informação, o que se traduziu no desenvolvimento de novos mecanismos de defesa de sistemas, redes e programas.

Uma das formas utilizadas para ter acesso à informação, de forma maliciosa são os ataques através de *botnets*. Diariamente, somos confrontados com este tipo de ataques, como tal, tornou-se indispensável o desenvolvimento de métodos avançados de deteção e prevenção através da utilização de modelos específicos de deteção.

Esta dissertação foca-se na deteção e mitigação de ataques cibernéticos, com ênfase em ataques do tipo *HTTP/2 rapid reset*, utilizando os mecanismos das SDN. Através da flexibilidade e controlo oferecidos pelas SDN, pretende-se desenvolver estratégias eficazes para identificar e mitigar ameaças de forma dinâmica e eficiente.

1.1 Motivação

Tendo por base a inovação em tecnologia de redes e da segurança cibernética, esta dissertação aborda três elementos cruciais do panorama digital contemporâneo: as SDN, as *botnets* e a mitigação de ataques específicos, como o *HTTP/2 rapid reset*, utilizando a flexibilidade e o controlo centralizado oferecidos pelas SDN. A relevância desta investigação reside na crescente dependência de infraestruturas de rede robustas e seguras, que são fundamentais para o funcionamento eficaz de inúmeros sistemas críticos, desde as comunicações empresariais até aos serviços essenciais de governo e da saúde.

As SDN representam um avanço significativo na forma como as redes são projeta-

das e operadas, proporcionando uma abordagem programável e centralizada que promete aumentar a eficiência, a adaptabilidade e a escalabilidade das mesmas. Esta arquitetura inovadora permite uma gestão dinâmica do tráfego de rede, não só a implementação de políticas de segurança sofisticadas, mas também a resposta rápida a mudanças nas condições da rede. No entanto, a natureza centralizada das SDN também introduz potenciais vulnerabilidades, tornando assim essencial a exploração de métodos para fortalecer a segurança contra as ameaças cibernéticas emergentes.

As *botnets*, ou seja, as redes de dispositivos comprometidos controlados por atacantes, constituem uma das ameaças mais persistentes e evasivas no ciberespaço. Capazes de executar uma variedade de ataques maliciosos, incluindo o infame *HTTP/2 rapid reset*, que pode desestabilizar as redes pois força o encerramento prematuro de conexões, as *botnets* requerem estratégias de deteção e mitigação totalmente eficazes. Neste contexto, a natureza distribuída e frequentemente camuflada das *botnets* exige uma abordagem de segurança que seja tanto adaptável quanto inteligente, características inerentes às SDN.

Este estudo é motivado pela necessidade crítica de entender como as SDN podem ser aplicadas não apenas para aprimorar a funcionalidade e o desempenho das redes, mas também como um poderoso instrumento na luta contra as *botnets* e na mitigação de ataques disruptivos como o *HTTP/2 rapid reset*. Investigaremos como as características únicas das SDN, como a visibilidade abrangente da rede e a capacidade de reprogramação dinâmica, podem ser utilizadas para detetar, isolar e neutralizar ameaças de *botnets* de maneira eficaz, minimizando o impacto deste tipo de ataques e fortalecendo a resistência da rede.

A dissertação visa contribuir significativamente para a área da segurança cibernética, oferecendo conhecimentos valiosos sobre a interseção entre a inovação em redes e a defesa contra ameaças cibernéticas. Ao focar o estudo na aplicação da SDN para combater *botnets* e mitigar ataques específicos, este trabalho busca não apenas avançar o conhecimento académico neste domínio do conhecimento, mas também fornecer orientações práticas para profissionais da área, ajudando a moldar estratégias de segurança de rede mais eficazes e resilientes para o futuro digital. Adicionalmente, todos os *scripts* desenvolvidos neste trabalho serão disponibilizados no GitHub.

1.2 Identificação do problema

No panorama atual da tecnologia de redes, as SDN representam um avanço significativo, promovendo uma gestão de rede mais flexível, eficiente e centralizada. Entretanto, à medida que as redes se tornam mais dinâmicas e programáveis, elas também se tornam alvos potenciais para uma variedade crescente de ameaças cibernéticas. Entre essas ameaças, as *botnets*, redes de dispositivos comprometidos controlados remotamente por atacantes, destacam-se pela sua capacidade de realizar ataques em larga escala, como o ataque *HTTP/2 rapid reset*[CVE-2023-44487]. Este ataque, caracterizado pela interrupção abrupta dos *streams* através do cancelamento massivo, através do uso de *frames RST-STREAM*, pode causar

desestabilização significativa das redes, resultando em interrupções de serviço e comprometimento da integridade e disponibilidade da rede. Assim, o problema específico a ser abordado é como utilizar as capacidades intrínsecas das SDN para desenvolver e implementar estratégias eficazes de mitigação contra ataques *HTTP/2 rapid reset*. A questão central envolve a identificação de métodos para detetar e responder a esses ataques em tempo real, minimizando o impacto na rede e mantendo a continuidade e a qualidade dos serviços. Além disso, e tentando ir um pouco mais além, o desafio inclui a necessidade de manter a flexibilidade e a adaptabilidade das SDN, garantindo ao mesmo tempo a segurança eficiente da rede contra ameaças cibernéticas avançadas.

Este problema é de particular relevância no contexto de infraestruturas críticas, como as da saúde que lidam com questões de vida e de morte e das redes empresariais, onde a estabilidade e a segurança da rede são fundamentais para a operação diária e a continuidade dos negócios. Assim, a identificação e a mitigação eficaz de ataques *HTTP/2 rapid reset* em ambientes SDN representam uma contribuição significativa para o campo da segurança cibernética e da tecnologia de redes, abordando um assunto pouco desenvolvido na literatura atual e fornecendo soluções práticas para a sua mitigação.

1.3 Contribuição

Este estudo visa contribuir para a área da segurança cibernética, oferecendo *insights* valiosos sobre a aplicação da SDN na deteção e mitigação de *botnets* e ataques específicos. Ao investigar as características únicas das SDN, como a visibilidade abrangente da rede e a capacidade de reprogramação dinâmica, este trabalho busca não apenas avançar o conhecimento académico, mas também fornecer orientações práticas para profissionais da área. Adicionalmente, o código desenvolvido será disponibilizado em código aberto no GitHub, facilitando a sua utilização e expansão pela comunidade. A dissertação também inclui a implementação de uma *framework* com validação de medidas de mitigação para este tipo de ataques.

1.4 Estrutura

Esta dissertação encontra-se organizada sob a forma de 9 capítulos, partindo de um nível de contexto mais amplo dedicado à introdução de tecnologias e conceitos-chave e caminhando progressivamente para os detalhes técnicos de conceção, implementação e validação das soluções propostas, cobrindo ainda as questões metodológicas e da organização do plano de trabalhos. Assim sendo, a estrutura adotada contempla os seguintes capítulos:

1. Introdução:

   Neste capítulo é estabelecido o contexto da dissertação, partindo da motivação por trás do estudo, destacando-se a sua relevância e a necessidade de

realizar pesquisas sobre deteção e prevenção de *botnets* em SDN, é identificado o problema específico que vai ser abordado. Nesta fase inicial também é apresentada a estrutura geral da investigação, pautada por um fio condutor: a originalidade da proposta apresentada para prevenir e/ou mitigar os ataques *botnets*.

2. Conceitos: Nesta capítulo, são introduzidos os conceitos fundamentais necessários para entender a dissertação. Para tal, é apresentada uma explicação detalhada sobre SDN incluindo a sua definição, vantagens, e tecnologias relacionadas como o openflow e o Programming Protocol-independent Packet Processors (P4-int). Também são discutidos os controladores de rede, com principal relevância em ONOS, OpenDaylight (ODL) ou Ryu. O conceito de *botnets* é explorado pormenorizadamente, bem como a sua estrutura, são ainda apresentados e os ataques Distributed Denial-of-Service Attack (DDoS) e *HTTP/2 rapid reset*, incluindo a mecânica desses ataques e projeções sobre seu desenvolvimento futuro.

3. Estado da arte: Neste novo capítulo, é feita uma análise aprofundada do conhecimento e das pesquisas existentes relacionadas com o tema escolhido. Neste ponto, descreve-se a estratégia de pesquisa adotada, incluindo a seleção de palavras-chave, critérios de inclusão e exclusão, e o impacto dessas escolhas na revisão bibliográfica. Além disso, é apresentada uma revisão abrangente da literatura existente, analisando-se tanto mecanismos de ataque quanto estratégias de defesa e mitigação. São ainda apresentadas propostas e discutidos alguns dos resultados obtidos na literatura.

4. Metodologia e planeamento: Este capítulo desenvolve-se tendo por base a abordagem metodológica adotada para a pesquisa e o planeamento da dissertação. Neste ponto, descreve-se um caso de estudo específico, propondo uma abordagem para mitigação do ataque *HTTP/2 rapid reset*. A metodologia de Design Science Research (DSR) é explicada como o método escolhido para direcionar a pesquisa.

5. Implementação do ambiente de testes: O objetivo principal deste capítulo é estabelecer uma estrutura robusta que garanta níveis elevados de segurança e resiliência na rede. Através da análise em tempo real e da gestão adaptativa de ameaças, o sistema desenvolvido visa ser uma ferramenta essencial na luta contra ameaças cibernéticas. Este proporcionará um ambiente seguro para operações de rede, capaz de identificar, mitigar e responder rapidamente a ataques, assegurando a integridade e a continuidade dos serviços de rede. A implementação inclui a integração de tecnologia de monitorização, deteção de intrusões e respostas automatizadas a incidentes, tudo com o propósito de criar uma defesa pro-ativa e dinâmica contra ciberameaças.

6. Simulação do ataque: Este capítulo descreve os tipos de testes realizados para detetar e mitigar ataques no sistema de segurança de rede, bem como a eficácia das metodologias implementadas. Os testes foram conduzidos com o objetivo de validar a robustez e a eficiência do sistema em identificar e neutralizar ameaças cibernéticas.

7. Análise dos tempos de execução dos mecanismos de defesa: Neste capítulo, realizamos uma análise detalhada dos tempos de execução dos diversos mecanismos de defesa implementados no nosso sistema de segurança. Os dados recolhidos registam o tempo de execução de cada tipo de mecanismo de defesa em resposta a eventos específicos na rede. Compreender esses tempos de execução é crucial para avaliar o desempenho e a eficiência das nossas estratégias de defesa. Esta análise permite identificar possíveis congestionamentos e áreas de melhoria, assegurando que o sistema possa responder de maneira eficaz e oportuna a ameaças cibernéticas, garantindo a integridade e a resiliência da rede.

8. Conclusão: Por fim são apresentadas as principais descobertas, contribuições e implicações da pesquisa. Neste ponto é feita uma reflexão sobre os objetivos alcançados e discute-se possíveis direções para pesquisas futuras.

Capítulo 2

Conceitos

2.1 Redes definidas por software

Neste capítulo, abordaremos dois tópicos fundamentais e interligados no mundo da tecnologia de redes: SDN e *botnets*. Enquanto as SDN representam uma abordagem inovadora e eficiente na gestão de redes, as *botnets* são uma forma de ameaça cibernética que se tornou cada vez mais presente e relevante na era digital.

Este capítulo visa fornecer uma compreensão detalhada dos conceitos da SDN, as suas vantagens e tecnologias associadas, bem como uma visão geral das *botnets*, destacando a importância da segurança em redes definidas por software.

2.1.1 Definição de rede definidas por software

A Rede Definida por Software (SDN) representa um avanço significativo na forma como as redes de computadores são geridas e operadas. Na realidade, ao dissociar a camada de controlo da camada aplicacional, a SDN como podemos visualizar na Figura 2.1, oferece um modelo de gestão de rede mais flexível, ágil e inovador [Soma, 2019].

- A camada aplicacional, também conhecida como *data plane*, é responsável pela receção e transmissão de pacotes na rede. É composto por portas e uma tabela de encaminhamento (*forwarding table*), que é, tal como o nome indica, consultada para encaminhar os pacotes recebidos. O encaminhamento é feito com base no cabeçalho do pacote e numa correspondência na tabela. Dependendo do caso, o pacote é encaminhado diretamente pelo plano de dados ou é encaminhado para o plano de controlo.

- A camada de controlo, também conhecido como *control plane*, é responsável por manter a tabela de encaminhamento atualizada, de forma a que o plano de controlo encaminhe o maior tráfego possível. Processa os diferentes protocolos existentes para manter a tabela atualizada e, assim, é capaz

de ter noção da topologia da rede. Os exemplos de protocolos neste plano incluem o Spanning Tree Protocol (STP) e o Open Shortest Path First (OSPF). Na SDN, o plano de controlo é centralizado e externo ao domínio do switch, o que permite uma maior flexibilidade e controlo na rede.

- Camada de infraestrutura, também conhecido como *management plane*, é a interface do switch com o controlador da rede, permitindo que os controladores configurem e monitorizem as estatísticas do switch. Este é capaz de alterar o estado dos outros dois planos anteriormente apresentados (controlo e aplicacional). Neste plano, os protocolos mais exemplificativos incluem o Simple Network Management Protocol (SNMP), usado para gerir e monitorizar dispositivos de rede [Soma, 2019].

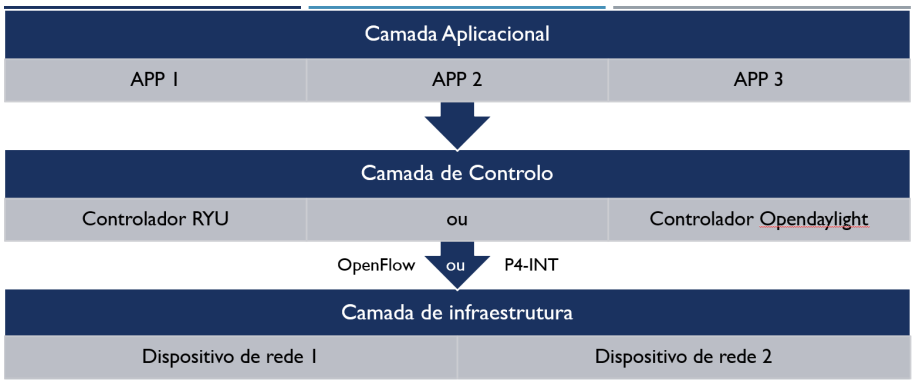


Figura 2.1: Diagrama SDN

2.1.2 Vantagens e desvantagens da SDN

Entre os benefícios mais visíveis da SDN estão a redução de custos, a maior flexibilidade e escalabilidade, e a gestão simplificada. Além disso, a SDN é útil na construção de nuvens privadas, públicas e híbridas, aumentando a agilidade no desenvolvimento e oferta de aplicações, adaptando-se melhor às necessidades dos negócios.

A SDN possibilita a criação de interfaces de rede através de um sistema de controlo que permite ao software monitorizar, definir e alterar a comutação da rede. Esses controladores da SDN funcionam como sistemas operacionais de redes de computadores, agindo como ponte entre as aplicações e o hardware, possibilitando a criação de aplicações personalizadas para cada rede específica, como tal, o uso da SDN traz vantagens competitivas significativas. Na verdade, as informações mais recentes sobre este tema indicam que a maioria das empresas que utilizam SDN ou Network Functions Virtualization (NFV) afirmam obter vantagens competitivas.

Além disso, a tecnologia SDN é vista como um impulsionador para a evolução da computação em nuvem, auxiliando na criação de aplicações na nuvem e na redução dos custos de rede, uma vez que elimina a necessidade de hardware específico e caro, permitindo que os custos sejam proporcionais ao consumo. [Tele-

com, 2018] A SDN ajuda, por isso, a melhorar a qualidade dos serviços e produtos oferecidos, gerando lucros a longo prazo[Group, 2018].

As principais vantagens da SDN são:

1. Flexibilidade: ao centralizar o controlo da rede, a SDN permite uma programação e gestão mais flexíveis. Ou seja, a rede pode ser adaptada de forma rápida e eficaz para atender a requisitos específicos, como o escalonamento para lidar com aumentos de tráfego ou a mudança de políticas de segurança.

2. Agilidade: a centralização do controlo também torna as redes SDN extremamente ágeis. Mudanças que anteriormente exigiriam reconfigurações manuais em vários dispositivos agora podem ser implementadas rapidamente através do controlador. Esta característica é particularmente valiosa em ambientes que exigem respostas rápidas a mudanças nas condições de rede ou demandas das aplicações.

3. Inovação: a capacidade de programar e personalizar o comportamento da rede abre novas possibilidades para a inovação. Esta qualidade pode incluir a criação de novos serviços de rede, o desenvolvimento de soluções personalizadas para desafios específicos de negócios, ou até mesmo a experimentação com novos protocolos de rede, tecnologias emergentes ou até conceitos inovadores, como é o caso dos Data Diodes [de Freitas et al., 2019].

4. Automação: a automação é uma das maiores vantagens da SDN. Tarefas repetitivas ou complexas de gestão de rede, como a configuração de roteamento ou a aplicação de políticas de segurança, podem ser automatizadas. Esta capacidade economiza tempo, reduz a carga de trabalho dos administradores de rede, e minimiza o risco de erros humanos.

5. Implementação de políticas de rede: a lógica de controlo centralizada da SDN facilita a implementação uniforme de políticas de rede em toda a infraestrutura, ou seja, garante que todas as partes da rede estejam alinhadas com as diretrizes de segurança, desempenho e conformidade da organização.

Desvantagens e desafios associados às SDN, especialmente em relação à segurança:

1. Suscetibilidade a Ataques centralizados: devido à natureza centralizada do controlo em redes SDN, estas podem ser particularmente vulneráveis a ataques que visam o controlador SDN. Como tal, se um atacante conseguir comprometer o controlador, ele poderá potencialmente ganhar controlo sobre toda a rede.

2. Superfície de ataque ampliada: a separação do plano de controlo do plano de dados e a introdução de APIs para gestão e comunicação aumentam a superfície de ataque, oferecendo mais pontos de entrada para possíveis ataques.

3. Complexidade de gestão: a flexibilidade e a capacidade de programação das SDN também introduzem complexidade no gestão da rede. Configurações incorretas ou políticas de segurança inadequadas podem inadvertidamente expor a rede a riscos adicionais.

4. Desafios de interoperabilidade: integrar soluções SDN em ambientes de rede existentes pode apresentar desafios de interoperabilidade com hardware e software ultrapassado, o que pode limitar a eficácia e a segurança da implementação das SDN.

5. Dependência de software: como as SDN dependem fortemente de software para gestão de rede e controlo de tráfego, são suscetíveis a vulnerabilidades de software, como *bugs* ou software desatualizado, que podem ser exploradas por atacantes.

6. Desafios de escalabilidade: embora as SDN promovam a escalabilidade, gerir e manter o desempenho e a segurança em redes de grande escala podem ser tarefas desafiadoras, exigindo recursos adicionais e estratégias de gestão complexas.

Em suma, a SDN transforma a gestão de redes de uma operação baseada em hardware para uma operação centrada em software, trazendo uma maior adaptabilidade e eficiência. Estando o digital em constante evolução, a escolha desta abordagem responde, não só às necessidades atuais de ambientes de rede dinâmicos, mas também estabelece uma base sólida para futuras inovações no campo das redes de computadores [Sanjeetha et al., 2022].

2.2 Protocolos SDN

2.2.1 OpenFlow

O OpenFlow foi um dos primeiros protocolos de comunicação a ser adotado para SDN e é amplamente usado para implementar o conceito de separação entre o plano de controlo e o plano de dados. Principais caraterísticas [Pereira et al., 2019]:

- Funcionalidade: permite que o controlador SDN interaja com o plano de encaminhamento (data plane), definindo regras de fluxo e comportamento de encaminhamento através de uma interface padronizada.

- Evolução independente: facilita a atualização e o desenvolvimento do plano de controlo sem a necessidade de alterar o hardware, permitindo a gestão unificada de dispositivos de rede de diferentes fabricantes.

- Limitações: tende a ser mais restrito em termos de programação do que P4-int, pois trabalha com um conjunto fixo de campos de cabeçalho e tipos de pacotes passíveis de manipulação.

2.2.2 P4-INT

O P4-int é um protocolo de programação que permite especificar como os pacotes são processados no plano de dados, de maneira altamente flexível e independente de protocolos. Principais caraterísticas:

- Funcionalidade: com o P4-int, é possível definir cabeçalhos personalizados e especificar a lógica de encaminhamento de pacotes de maneira muito granular e dinâmica.
- Independência de protocolo: o P4-int não está vinculado a protocolos específicos, permitindo que os programadores inovem e criem novos protocolos de rede conforme as suas necessidades.
- Capacidade de programação: os dispositivos podem ser reprogramados em campo para suportar novos cabeçalhos e tipos de pacotes, o que não é possível no OpenFlow.
- Controlo de propriedade intelectual: o P4-int oferece aos utilizadores a possibilidade de manter o controlo sobre seus algoritmos e inovações sem depender dos fabricantes de equipamentos de rede.
- Inovação: este protocolo permite que os controladores de rede tenham uma capacidade sem precedentes para inovar no plano de dados, desde a criação de novos protocolos até a otimização do processamento de pacotes para aplicações específicas.

Ao comparar estes dois protocolos em relação às abordagens para o processamento de pacotes em SDN verificamos que o OpenFlow é caracterizado por um modelo de processamento baseado em tabelas com campos pré-definidos, oferecendo uma estrutura mais rígida, mas bem estabelecida para a configuração de dispositivos de rede. Por seu lado, o P4-int apresenta uma abordagem mais flexível e programável, permitindo a definição personalizada de cabeçalhos de pacotes e o comportamento de processamento, o que facilita a inovação e a adaptação a requisitos específicos de rede. Essa flexibilidade do P4-int torna-o particularmente adequado para ambientes de pesquisa e desenvolvimento que exigem a implementação de protocolos novos ou experimentais. Em relação ao controlo, ambos os protocolos aumentam o controlo da rede sobre os seus sistemas, mas o P4-int oferece um nível de controlo ainda mais profundo e flexível sobre o comportamento do plano de dados [Hu et al., 2014].

2.3 Sistemas operacionais de rede (controladores)

O Open Network Operating System (ONOS), ODL e Ryu são os mais populares e avançados projetos de Sistemas operacionais de rede (NOS) de código aberto disponíveis, todos têm como objetivo fornecer alta escalabilidade, fiabilidade e extensibilidade [Yoon et al., 2017].

2.3.1 Ryu

Ryu é um controlador SDN baseado em Python que desempenha um papel crucial na melhoria da adaptabilidade da rede ao moldar de forma transparente a gestão do tráfego. Funciona como o "cérebro"do sistema SDN, transmitindo informação e dando instruções para adaptar o tráfego nos switchs e routers através de APIs. O Ryu é utilizado para obter estatísticas de captura de tráfego criados a partir de arquivos pcap, o que o torna essencial para a deteção e mitigação de ataques de *botnet* dentro de um ambiente SDN [Sanjeetha et al., 2022]. Este controlodor é certificado para trabalhar com uma variedade de dispositivos de encaminhamento baseados em OpenFlow, destacando-se pela sua adaptabilidade e facilidade de utilização na orquestração de redes inteligentes [Uddin and Monir, 2020].

2.3.2 ONOS (Open Network Operating System)

O ONOS é um sistema operacional de rede de código aberto, projetado especificamente para redes SDN. Este sistema foi desenvolvido e mantido pela Open Networking Foundation (ONF), disponibilizando o controlo centralizado sobre a rede e os seus dispositivos.

Este solução utiliza o *framework* Open Services Gateway initiative (OSGi) e fornece uma arquitetura modular que suporta o desenvolvimento de *plugins*. Além disso, apresenta várias características únicas, como escalabilidade dinâmica, tolerância a falhas e alta disponibilidade, por isso, é adequado para redes de provedores de serviços, *data centers* e redes empresariais de grande dimensão, como a Huawei e Nokia.

Em suma, este sistema oferece várias funcionalidades de segurança para garantir a estabilidade e a postura de segurança das redes. Uma dessas funcionalidades é a extensão do modo de segurança, que impõe políticas de segurança para cada aplicação. O subsistema de segurança garante que quaisquer tentativas de uso que não tenham sido explicitamente planeadas e autorizadas pelo utilizador sejam negadas. A extensão do modo de segurança visa garantir que o controlador ONOS analisa a política antes da ativação, ou seja que verifica a autorização associando a aplicação a política durante a execução [Yoon et al., 2017].

2.3.3 OpenDaylight (ODL)

ODL é um controlador SDN modular de código aberto baseado em Java, que permite criar funções e serviços de rede de forma modular e flexível. É apoiado por uma grande comunidade com mais de 600 profissionais e programadores, como tal, é considerado um dos controladores mais importantes e populares da paisagem SDN.

As vantagens do ODL incluem uma arquitetura modular e conetável, o que permite construí-lo a partir de uma variedade de componentes de forma flexível e

personalizável, ou seja, adequada às necessidades de cada utilizador. O ODL usa uma abordagem programática centralizada, o que permite um alto nível de controlo sobre a infraestrutura de rede e um alto desempenho do controlador. Além disso, a sua API baseada em JSON abstrai a complexidade subjacente das tecnologias de rede, facilitando o uso por parte do utilizador.(ver Figura 2.2)

Uma das principais limitações do ODL é a centralização do plano de controlo, que, como mencionado anteriormente, pode ser explorada por utilizadores mal-intencionados em ataques DDoS com alvo no controlador. Além disso, o ODL tem menor desempenho quando comparado com outros controladores por ser tão personalizável e fácil de programar.

Apesar das desvantagens, o ODL é uma plataforma importante para a utilização em SDN, pela sua arquitetura modular, pela ampla gama de recursos e pela grande comunidade de programadores, que garantem a constante evolução das funcionalidades. Na verdade, em diferentes testes, o ODL apresentou liderança em termos de aproveitamento de largura de banda, no entanto também registou algumas desvantagens em relação a outros controladores em termos de robustez e perda de pacotes. [Tran et al., 2018] [Uddin and Monir, 2020]

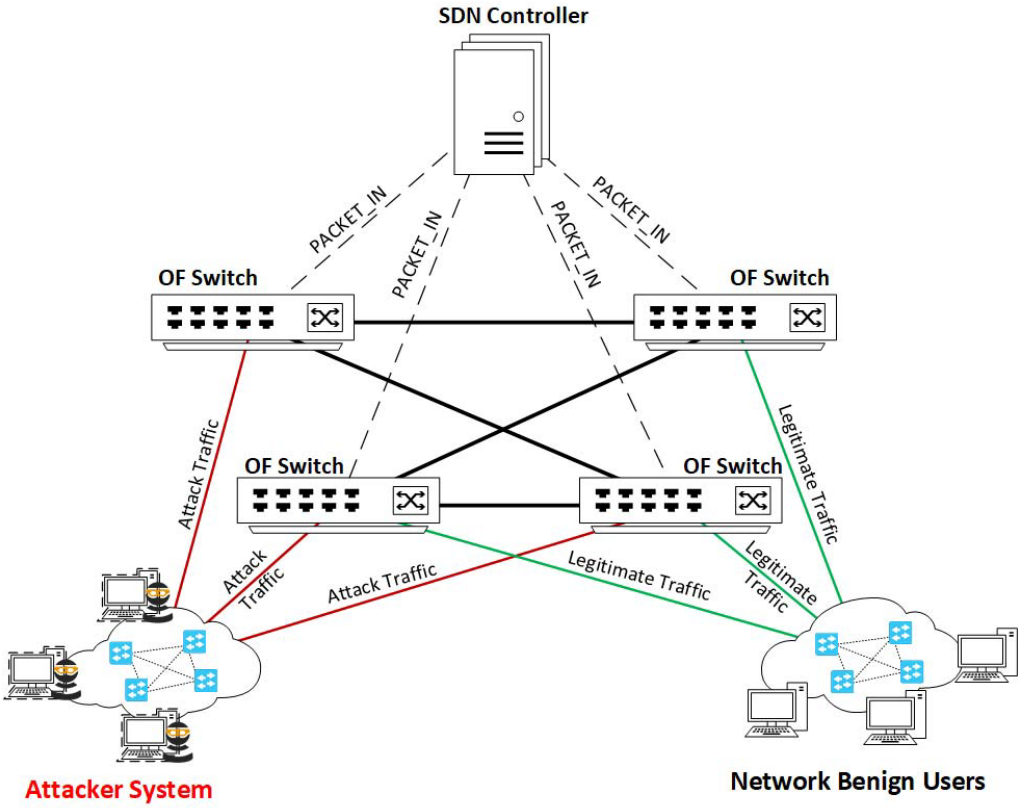


Figura 2.2: OpenDayLight Architecture
[Tran et al., 2018]

2.4 Botnets

Botnets são redes de computadores infetados por malware e controlados remotamente por cibercriminosos. Neste caso, os computadores, também chamados de *bots*, são usados para executar ataques coordenados ou outras atividades maliciosas [kaspersky, 2018].

Esta definição resulta da combinação das palavras "robot"e "network", indicando uma rede de "robots"controlados remotamente. Estes cibercriminosos usam *botnets* para executar uma variedade de atividades mal-intencionadas, como ataques DDoS, fraudes de anúncios, propagação de malware, roubo de identidade e disseminação de spam. O tamanho da *botnet* pode variar significativamente, indo de centenas a milhões de dispositivos infetados. Tal como mencionado, a infeção dos dispositivos para formar uma *botnet* ocorre por meio de diversas técnicas, incluindo *phishing*, exploração de vulnerabilidades e instalação de malware por downloads maliciosos. A sofisticação desses métodos torna a deteção e prevenção um desafio constante para os especialistas em segurança cibernética.

Tal como anteriormente mencionado, as *botnets* normalmente têm origem numa infeção por malware, como por exemplo o cavalo de Troia, que se espalha para outros dispositivos ligados. A infeção pode ocorrer por meio de e-mails maliciosos, downloads ilegais ou clicando em links suspeitos. Para prevenir infeções por *botnets*, é crucial manter o sistema operacional e o software antimalware atualizados, evitar clicar em links ou abrir anexos suspeitos em e-mails, e realizar downloads apenas de fontes confiáveis.

Em redes SDN, onde a configuração e o controlo da rede são centralizados e programáveis, *botnets* exploraram as vulnerabilidades para controlar dispositivos ou realizar ataques. Neste contexto, torna-se essencial a implementação de soluções de segurança robustas e sistemas de monitorização para identificar e mitigar essas ameaças. A deteção eficaz de *botnets* em redes SDN requer ferramentas de segurança que possam identificar tráfego anormal e comportamentos suspeitos, permitindo uma rápida resposta a incidentes de segurança.

Estes ataques podem afetar significativamente o desempenho e a disponibilidade da rede, além de comprometer a segurança dos dados. Os efeitos de *botnets* podem ser extensos, afetando tanto indivíduos quanto organizações[Gatefy, 2021] [kaspersky, 2018].

A estrutura de uma *botnet* é complexa e frequentemente envolve uma comunicação descentralizada entre *bots* e o controlador, também chamado de *botmaster*. Canais como IRC ou redes *peer-to-peer* são comuns para este tipo de comunicação, pois proporcionam um ambiente indetetável para os atacantes. Além disso, técnicas de polimorfismo e persistência são usadas para evitar deteções e garantir a eficácia e longevidade da *botnet* [Gatefy, 2021] [kaspersky, 2018].

2.5 Ataques

Num momento inicial destas violações, o atacante compromete uma série de dispositivos através de malware para criar uma *botnet*. Esses dispositivos podem incluir computadores pessoais, routers, câmaras de segurança e outros dispositivos conectados à internet (Internet of Things (IoT)). Uma vez que a *botnet* está estabelecida, cada dispositivo comprometido é usado para enviar solicitações ao servidor, aplicação ou rede alvo, de forma coordenada e em grande volume. O volume maciço de tráfego gerado pela *botnet* excede a capacidade do alvo de processar solicitações legítimas. Como resultado, o alvo torna-se lento ou completamente inacessível para os utilizadores legítimos. Os ataques DDoS podem variar em complexidade, desde solicitações simples de conexão até métodos mais sofisticados que exploram vulnerabilidades específicas no alvo [Cloudflare, 2022].

À medida que são desenvolvidos novos mecanismos de defesa de sistemas, redes e programas, surgem novas formas de ataque, tais como:

1. Amplificação de DTLS: os ataques de amplificação exploram protocolos vulneráveis para aumentar o volume de tráfego direcionado ao alvo. No caso do Datagram Transport Layer Security (DTLS), os atacantes podem provocar respostas maiores do que a solicitação original, resultando num ataque mais potente (ver Figura 2.3).

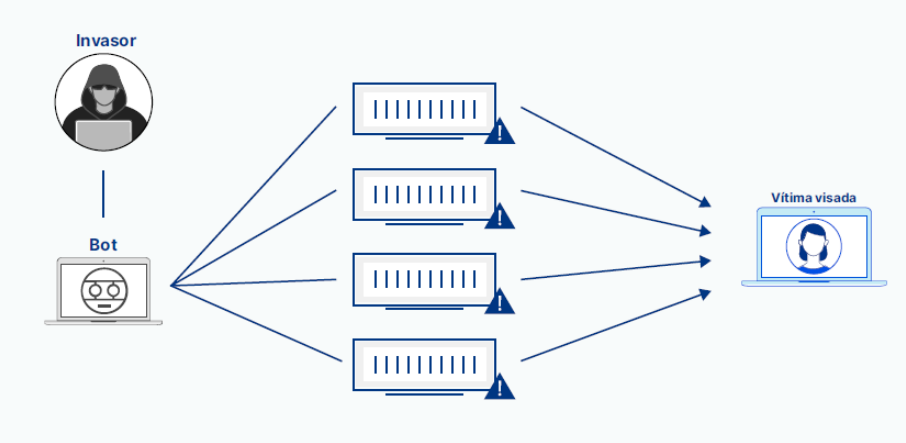


Figura 2.3: Amplificação de DTLS
[Cloudflare, 2022]

2. Ataques no Protocolo QOTD: o Quote of the Day (QOTD) é um protocolo menos conhecido que pode ser explorado para ataques de reflexão/ amplificação, devido à sua natureza de resposta automática a solicitações de entrada (ver Figura 2.4).

3. Ataques pelo Protocolo QUIC: o Quick UDP Internet Connections (QUIC) é um protocolo recente projetado para conexões mais rápidas e seguras. No entanto, a sua utilização crescente torna-o um alvo perfeito para atacantes, que sabem explorar as suas características para realizar ataques DDoS (ver Figura 2.5).

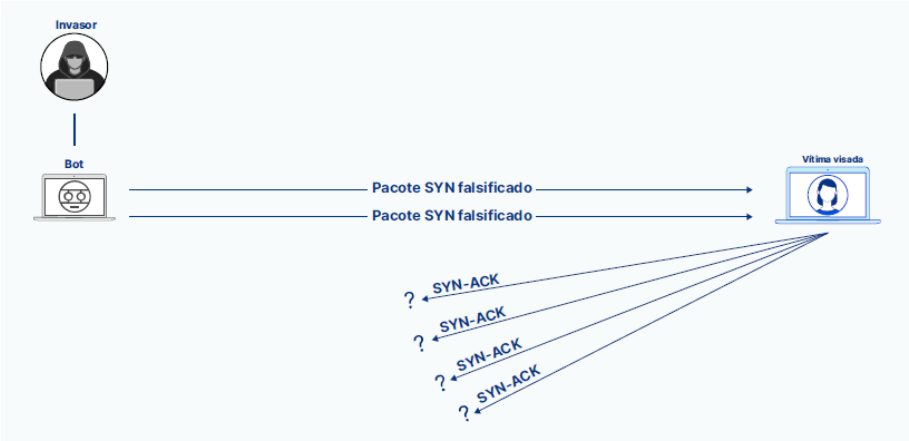


Figura 2.4: Ataques no Protocolo QOTD
[Cloudflare, 2022]

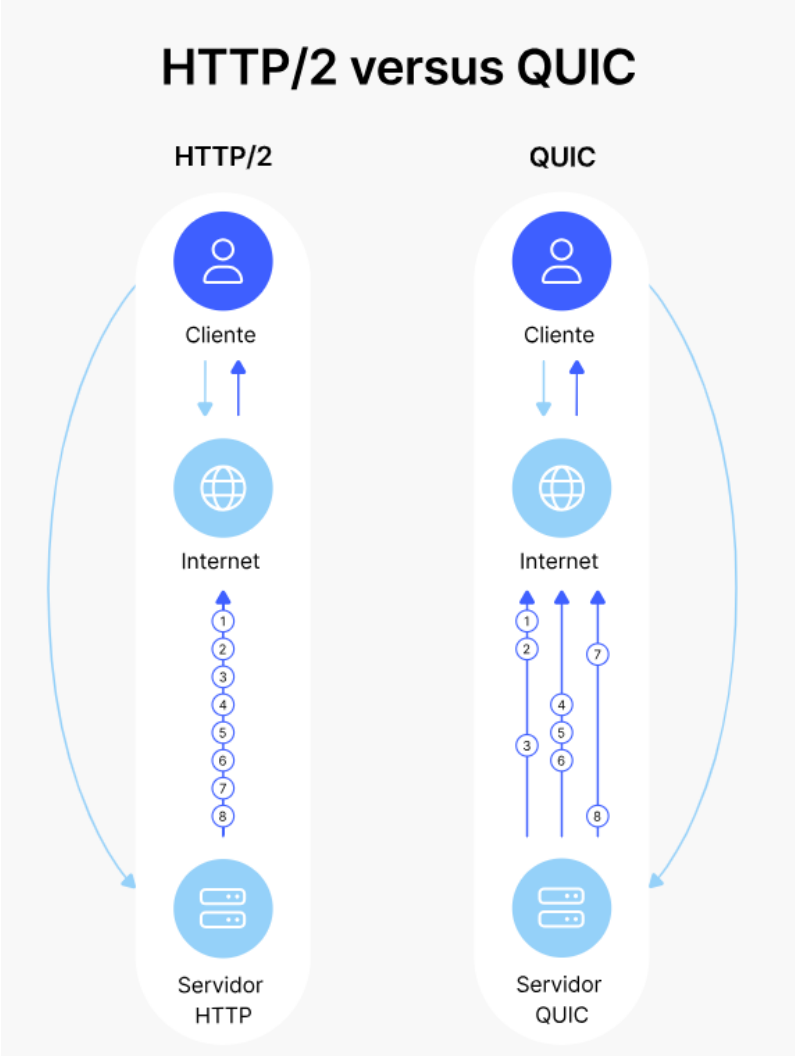


Figura 2.5: Ataques no Protocolo QUIC
[Nakutavičiūtė, 2020]

4. Foco na Camada 7 (Aplicacional): estes ataques concentram-se na camada das aplicações do modelo open systems interconnection (OSI), onde ocor-

rem as interações entre o utilizador e a aplicações, como páginas web e serviços API. Geralmente, envolvem um fluxo de solicitações HTTP ou HTTPS para o servidor. A intensidade destes ataques é frequentemente medida em requests per second (RPS). Um grande número de RPS pode exceder a capacidade do servidor de processar solicitações legítimas, levando à negação de serviço. Os ataques à camada de aplicações, como o *HTTPS/2 rapid reset* exploram vulnerabilidades no protocolo HTTP/2, utilizados para a comunicação web. Neste tipo de ataque, são enviados uma série de *frames RST-STREAM* de forma rápida e repetida para interromper as conexões HTTP/2 estabelecidas entre clientes e servidores. Este tipo de ataques podem ser mais difíceis de detetar do que ataques DDoS de rede mais tradicionais, uma vez que estes imitam o tráfego normal do utilizador (ver Figura 2.6) [Cloudflare, 2022].

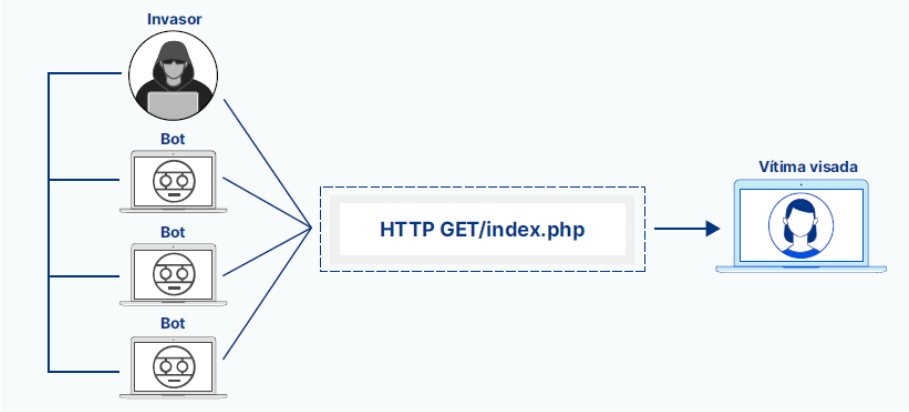


Figura 2.6: Camada de aplicações
[Cloudflare, 2022]

2.6 Resumo

Este capítulo apresentou uma visão abrangente dos conceitos/pressupostos fundamentais necessários para compreender o desenvolvimento desta dissertação. Verificou-se que:

- a SDN destacou-se pela sua flexibilidade, eficiência e potencial para melhorar a segurança da rede, apesar dos desafios associados.
- a introdução ao protocolo OpenFlow e P4-INT proporcionou uma base sólida para compreender as tecnologias subjacentes à SDN.
- a análise dos controladores, com especial foco para o Ryu destacou-se pela sua adaptabilidade e facilidade de utilização na orquestração de redes inteligentes.
- análise das *botnets* e dos ataques DDoS ressaltou a importância de uma abordagem pro-ativa na deteção e mitigação de ameaças.

- o ataque *HTTP/2 rapid reset* é particularmente difícil de detetar, pois como a maioria dos ataques à camada aplicacional, simula o tráfego legítimo de uso comum.

Com esses conceitos/pressupostos bem estabelecidos, estamos melhor preparados para explorar as soluções propostas e as pesquisas existentes nas secções seguintes.

Capítulo 3

Estado da arte

Neste segmento, exploraremos o estado da arte em pesquisa bibliográfica, um componente crucial no desenvolvimento de qualquer estudo académico ou profissional. A pesquisa bibliográfica é a espinha dorsal de qualquer investigação, fornecendo uma base sólida de conhecimento existente e novas perspetivas.

Este capítulo é estruturado por subsecções, cada uma abordando um aspeto fundamental do processo de pesquisa bibliográfica, desde a estratégia selecionada até à escolha final da literatura

3.1 Estratégia de pesquisa

3.1.1 Seleção de palavras-chave

Esta seleção e utilização de palavras-chave foi a estratégia escolhida para orientar a pesquisa perante o enorme volume de informação existente sobre segurança cibernética. Cada uma das palavras selecionada abrange uma dimensão importante do campo e, quando combinadas, oferecem um panorama abrangente das tecnologias, técnicas e desafios envolvidos.

- *Botnets*: redes de dispositivos infetados com malware e controlados por um atacante. Pesquisas com esta palavra-chave geralmente exploram métodos de deteção, controlo e mitigação de *botnets*.
- SDN: redes de computadores virtuais, que permitem uma gestão centralizada e configurável. Esta palavra permitiu dar ênfase à tecnologia SDN, usualmente utilizada para melhorar a segurança de rede e a eficiência da mitigação de ataques.
- Deteção(Detection): a deteção de ameaças é uma parte fundamental da segurança cibernética. Esta palavra direcionou a pesquisa para dados sobre as últimas tecnologias e estratégias para identificar ataques cibernéticos atuais ou no futuro.

- Prevenção(Preventing): a prevenção de ameaças assenta no preceito de evitar os ataques antes que eles ocorram, como tal inclui pesquisas sobre as melhores práticas, políticas de segurança e tecnologias preventivas.

- DDoS: os ataques de negação de serviço distribuída são uma grande ameaça na segurança cibernética. Este termo permitiu abordar métodos de mitigação, análise de tendências de ataques e desenvolvimento de estratégias de defesa.

- Mitigação de ataques(Attack mitigation): refere-se às técnicas e estratégias usadas para diminuir o impacto dos ataques cibernéticos. Esta palavra conduziu a estudos sobre respostas a incidentes, recuperação de sistemas e medidas de resiliência.

- IDS (Sistema de Deteção de Intrusão): um Intrusion Detection System (IDS) é usado para detetar atividades suspeitas ou maliciosas dentro de uma rede. Este conceito permitiu focar a pesquisa em novos modelos de IDS, integração com outras tecnologias de segurança e otimização de desempenho.

- IPS (Sistema de Prevenção de Intrusão): um Intrusion prevention system (IPS) é uma forma avançada de IDS que não só deteta, mas também desenvolve medidas para prevenir a intrusão. As pesquisas sobre IPS permitem explorar técnicas de bloqueio automático, integração com *firewalls* e estratégias de resposta a ameaças.

- *HTTP/2 rapid reset*: neste ataque, é enviado uma série de *frames RST-STREAM* de forma rápida para cancelar os *streams* dentro da ligação TCP estabelecida entre cliente e servidor.

3.1.2 Critérios de inclusão e exclusão

A combinação das palavras-chave selecionadas com os critérios de inclusão e exclusão permitiram recuperar informação pertinente para a investigação desenvolvida.

Critérios de inclusão

- Atualidade do tema: priorizar pesquisas sobre questões atuais garante que a informação seja relevante e aplicável aos desafios e tecnologias modernas no campo da segurança cibernética.

- Credibilidade da fonte: incluir apenas estudos baseados em fontes confiáveis e revistos por pares assegura a credibilidade e a validade científica das informações recolhidas.

- Ataques realizados nos últimos 2 anos: permite a análise de pesquisas que estudam *botnets* sobre as ameaças mais recentes, particularmente em ambientes de SDN.

Critérios de exclusão

- Idade do ataque superior a 2 anos: excluindo estudos sobre ataques antigos, a pesquisa concentra-se em tendências atuais e técnicas emergentes. Este critério é particularmente relevante em segurança cibernética, onde as ameaças evoluem rapidamente.

- Foco exclusivo em mitigação: ao excluir pesquisas que se concentram apenas em mitigação, a revisão enfatiza a deteção e prevenção, elementos cruciais para interromper ataques antes que eles causem danos significativos.

- Estudos intensivos do ataque: descartar estudos que aprofundam exaustivamente os detalhes específicos de um único ataque ajuda a manter a pesquisa ampla e aplicável a uma variedade de cenários, em vez de limitada a um caso particular.

3.1.3 Impacto na revisão bibliográfica

A aplicação destes critérios assegura que a revisão bibliográfica seja focada em estudos atualizados e relevantes que abordam diretamente a deteção e prevenção de *botnets* em ambientes de SDN. Ao concentrar a atenção em pesquisas de alta qualidade e dando relevância à prática, é mais fácil compreender as tendências atuais, identificar as melhores práticas e contribuir significativamente para possíveis avanços no campo da segurança cibernética.

Em suma, uma abordagem equilibrada entre exclusão e inclusão cria uma base sólida para uma revisão bibliográfica informativa e inovadora.

3.1.4 Base de dados

A consulta em bases de dados académicas e bases de dados profissionais e técnicas, permite o acesso a informação variada e pertinente sobre a deteção e prevenção de *botnets*.

Bases de dados académicas

- IEEE Xplore: é uma das fontes mais valiosas para pesquisa em engenharia elétrica e ciência da computação. Os documentos da IEEE oferecem *insights* técnicos profundos, estudos de caso e pesquisas inovadoras, o que torna esta base de dados especialmente valiosa para tecnologias emergentes em segurança cibernética.

- ACM Digital Library: esta biblioteca é essencial para pesquisas em ciência da computação e tecnologia da informação, uma vez que oferece uma vasta coleção de publicações.

- Scopus: é uma base de dados bibliográfica abrangente e uma ferramenta de pesquisa amplamente utilizada por investigadores, cientistas e profissionais em diversas áreas do conhecimento. É uma das maiores bases de dados académicos do mundo.

Fontes de informações profissionais e técnicas

- Cloudflare: como líder nos serviços de segurança e desempenho de internet, os recursos da Cloudflare fornecem informações práticas e estudos de caso sobre ataques DDoS e estratégias de mitigação em ambientes reais.

- Kaspersky: conhecida pelas suas soluções de segurança cibernética, a Kaspersky oferece informações sobre tendências de malware, incluindo a evolução e o comportamento de *botnets*.

- Cisco: é uma autoridade em redes e segurança cibernética. Os seus recursos oferecem informações sobre as melhores práticas, as mais recentes soluções de segurança e inovações tecnológicas em SDN.

- Red Hat: especializada em soluções de software de código aberto, a Red Hat partilha informações sobre a implementação de segurança em ambientes de SDN e o uso de tecnologias de código aberto para aprimorar a segurança cibernética.

- Gatefy e Stefanini: ambos oferecem conhecimento acerca de soluções específicas de segurança cibernética e práticas de gestão de segurança. São consultados como recursos adicionais para entender as soluções práticas e estratégias implementadas no campo.

3.2 Escolha de literatura

Todos os documentos reunidos na pesquisa determinada pelos critérios de exclusão e inclusão previamente selecionados serão objeto de análise e reflexão, através da seguinte estratégia:

- Definição objetivos
- Categorização por temas ou tópicos
- Criação de uma matriz de comparação
- Identificação de tendências e padrões
- Sintetização e apresentação de resultados
- Consulta de revisões sistemáticas

3.3 Descrição dos documentos

Neste capítulo, a seleção de controladores e protocolos foi guiada pela avaliação criteriosa das suas vantagens em termos de compatibilidade e desempenho. Após a análise minuciosa de uma vasta gama de documentos, tornou-se evidente a inviabilidade de abordar cada um deles. Portanto, a escolha foi direcionada para aqueles que não apenas se alinham aos requisitos técnicos do projeto, mas também se destacam em eficiência e segurança. Esses dois pilares fundamentais sustentam a integridade e a robustez da dissertação, refletindo a importância de uma infraestrutura confiável e segura.

3.3.1 Eficiência dos Controladores

A tabela presente na Figura 3.1, fornece uma visão comparativa de cada controlador SDN, destacando as suas principais características para uma melhor compreensão. A análise dos controladores inclui aspetos como consistência, linguagem, falhas e arquitetura, proporcionando uma base sólida para a escolha do controlador mais adequado para diferentes ambientes de rede.

Name	Architecture	Northbound API	Consistency	Faults	License	Prog. language	Version
Beacon [186]	centralized multi-threaded	ad-hoc API	no	no	GPLv2	Java	v1.0
DISCO [185]	distributed	REST	—	yes	—	Java	v1.1
ElastiCon [229]	distributed	RESTful API	yes	no	—	Java	v1.0
Fleet [200]	distributed	ad-hoc	no	no	—	—	v1.0
Floodlight [189]	centralized multi-threaded	RESTful API	no	no	Apache	Java	v1.1
HP VAN SDN [184]	distributed	RESTful API	weak	yes	—	Java	v1.0
HyperFlow [195]	distributed	—	weak	yes	—	C++	v1.0
Kandoo [230]	hierarchically distributed	—	no	no	—	C, C++, Python	v1.0
Onix [7]	distributed	NVP NBAPI	weak, strong	yes	commercial	Python, C	v1.0
Maestro [188]	centralized multi-threaded	ad-hoc API	no	no	LGPLv2.1	Java	v1.0
Meridian [192]	centralized multi-threaded	extensible API layer	no	no	—	Java	v1.0
MobileFlow [223]	—	SDMN API	—	—	—	—	v1.2
MuL [231]	centralized multi-threaded	multi-level interface	no	no	GPLv2	C	v1.0
NOX [26]	centralized	ad-hoc API	no	no	GPLv3	C++	v1.0
NOX-MT [187]	centralized multi-threaded	ad-hoc API	no	no	GPLv3	C++	v1.0
NVP Controller [112]	distributed	—	—	—	commercial	—	—
OpenContrail [183]	—	REST API	no	no	Apache 2.0	Python, C++, Java	v1.0
OpenDaylight [13]	distributed	REST, RESTCONF	weak	no	EPL v1.0	Java	v1.{0,3}
ONOS [117]	distributed	RESTful API	weak, strong	yes	—	Java	v1.0
PANE [197]	distributed	PANE API	yes	—	—	—	—
POX [232]	centralized	ad-hoc API	no	no	GPLv3	Python	v1.0
ProgrammableFlow [233]	centralized	—	—	—	—	C	v1.3
Pratyaastha [198]	distributed	—	—	—	—	—	—
Rosemary [194]	centralized	ad-hoc	—	—	—	—	v1.0
Ryu NOS [191]	centralized multi-threaded	ad-hoc API	no	no	Apache 2.0	Python	v1.{0,2,3
SMaRtLight [199]	distributed	RESTful API	yes	yes	—	Java	v1.0
SNAC [234]	centralized	ad-hoc API	no	no	GPL	C++	v1.0
Trema [190]	centralized multi-threaded	ad-hoc API	no	no	GPLv2	C, Ruby	v1.0
Unified Controller [171]	—	REST API	—	—	commercial	—	v1.0
yanc [196]	distributed	file system	—	—	—	—	—

Figura 3.1: Controladores SDN
[Kreutz et al., 2014]

No documento "Evaluation of four SDN controllers with *firewall* modules"[Uddin and Monir, 2020] verificou-se que:

- ODL:

  - Apresentou o melhor desempenho em termos de capacidades de aproveitamento de largura de banda e demonstrou a menor quantidade de perda de pacotes;

  - Alcançou a maior taxa de transmissão de dados sem restrições, atingindo 32.75 Gbits/segundo;

  - Utilizou a versão Beryllium, preferida devido ao seu histórico de desempenho estável.

- POX:

  - Apresentou taxas de transmissão de largura de banda razoáveis, porém, falhou em termos de taxa de entrega de pacotes;

  - Mostrou consistência na orquestração de topologias personalizadas, com menos falhas e erros, tornando-o uma escolha conveniente para pesquisas e bancos de testes de rede.

- Floodlight:

  - Demonstrou um desempenho razoável para taxas de transmissão menores;

  - O desempenho diminuiu consideravelmente quando a taxa de transmissão aumentou, resultando num aumento significativo na perda de pacotes.

- Ryu:

  - Apresentou o menor rendimento em termos de transmissão ilimitada;

  - Mostrou consistência na preservação da taxa de entrega de pacotes.

Em resumo, o ODL destacou-se como o controlador com o melhor desempenho em termos de *throughput*, apresentando alta capacidade de largura de banda e baixa perda de pacotes. No entanto, penso que a escolha do controlador mais adequado dependerá sempre das necessidades e requisitos específicos da rede, bem como da conveniência para os administradores de rede

3.3.2 Segurança dos Controladores

O documento [Dziong et al., 2016] aborda à temática de segurança de controladores apresentando os seguintes resultados:

- ODL:

  - Melhorias de segurança: destaca-se pela segurança com funcionalidades como AAA (Autenticação, Autorização e Auditoria) e SNBI (Infraestrutura Segura de Inicialização de Rede) para controlar o acesso e gerir a autenticação de switches.;

- Mitigação de *spoofing*: combate a falsificação de identidade através da autenticação baseada em *tokens* e procedimentos rigorosos de autenticação de switches;

- Segurança no fluxo de dados: utiliza TLS para segurança das interfaces sul e HTTPS para as interfaces norte, protegendo contra adulterações e divulgação de informações;

- Vulnerabilidades abordadas: as diferentes vulnerabilidades detetadas anteriormente, como ataques XXE na SBI, foram mitigadas em versões recentes ao usar o protocolo NETCONF.

- ONOS:

  - Modelo de Acesso Conservador: implementa controlo de acesso rigoroso através da auditoria de aplicações e monitorização constante, melhorando assim a segurança;

  - Permissão de Acesso Detalhada: oferece controlo de acesso detalhado para aplicações e utilizadores, categorizando o acesso aplicacional para prevenir acesso não autorizado e divulgação de informações;

  - Segurança de Comunicação: garante fluxos de dados seguros na SBI e NBI com TLS e HTTPS, sendo assim similar ao OpenDaylight.

- Ryu:

  - Vulnerabilidades de Arquitetura: devido à sua arquitetura monolítica, o Ryu é suscetível a *spoofing*, pois carece de um mecanismo de autenticação tanto na NBI quanto na SBI;

  Exposição de Processos: o processo central é vulnerável a adulterações, e a falta de mecanismos de controlo de acesso expõe Ryu a potenciais ataques DDoS e ao envio não autorizado de mensagens;

  Divulgação de Informações: a comunicação entre o controlador e a NBI não é criptografada, tornando-a vulnerável a ataques de intercetação (MITM) e a adulteração de dados.

Em suma, cada controlador apresenta vantagens e desvantagens, no entanto, o ODL é o mais recomendado, ao nível do desempenho e ao nível dos mecanismos de segurança, em resultado de constante melhorias de segurança e da sua estrutura modular.

3.3.3 Detalhes de protocolos

O documento "A Method for Comparing OpenFlow and P4": [Hu et al., 2014] analisa os diferentes detalhes dos protocolos, tais como:

- Flexibilidade e customização: O P4 oferece um alto grau de flexibilidade e customização, permitindo que os administradores de rede criem protocolos e algoritmos personalizados sem depender da entrada do fabricante. Esta característica pode ser vantajosa para os administradores que procuram soluções sob medida para requisitos específicos da rede.

- Independência de Protocolo: a independência de protocolo do P4 permite a criação de campos de cabeçalho conforme necessário, proporcionando um nível de adaptabilidade que pode não ser tão facilmente alcançado com o OpenFlow. Esta particularidade pode ser especialmente benéfica em cenários que exigem processamento personalizado de pacotes.

- Desempenho e eficiência: A comparação apresentada no artigo sugere que pode haver uma diferença mínima de desempenho entre o *firmware* otimizado do OpenFlow e o processador de pacotes P4 no Zodiac FX, no entanto, também se verifica uma redução significativa no tamanho do binário do *firmware* P4 em comparação com a versão do OpenFlow, indicando possíveis reduções de código e ganhos de eficiência com o P4.

- Desenvolvimento e inovação futuros: Como o P4 permite planos de dados programáveis independentes de protocolo, os administradores de rede podem considerá-lo propício para desenvolvimento e inovação futuros na funcionalidade de dispositivos de rede.

- Consideração de casos de uso: Os administradores de rede devem considerar cuidadosamente os casos de uso específicos e os requisitos das redes ao avaliar o OpenFlow e o P4. As capacidades e limitações únicas de cada abordagem podem torná-las mais adequadas ou menos eficazes para diferentes aplicações ou ambientes.

3.3.4 Compatibilidade de controladores e protocolos

No documento "On the Performance of SDN Controllers: A Reality Check": [of Electrical et al., 2015], os autores abordam a compatibilidade de controladores e protocolos e apresentam como principais conclusões:

- Compatibilidade do controlador: Os controladores SDN selecionados, como Ryu, Pox, Nox, Floodlight e Beacon, suportam várias versões do protocolo OpenFlow. Essa compatibilidade garante que esses controladores possam comunicar efetivamente com os switches compativeis OpenFlow na rede.

- Avaliação de desempenho: O documento avalia o desempenho dos controladores SDN com base em seu gestão de mensagens OpenFlow e interações com os switches de rede. As diferentes métricas de desempenho como throughput, latência e equidade são avaliadas para determinar a eficácia de cada controlador em gerir redes baseadas em OpenFlow.

- Benchmarks do controlador: O estudo faz referência ao uso de ferramentas de *benchmark* como o Cbench, que são projetadas para avaliar o desempenho dos controladores SDN na gestão de mensagens OpenFlow. Essas ferramentas ajudam a medir a capacidade de resposta e escalabilidade do controlador sob diferentes condições de rede.

- Evolução do OpenFlow: O texto menciona que novos ramos derivados da ferramenta Cbench original visam adicionar suporte para as últimas especificações do OpenFlow. Esta inovação destaca a evolução contínua da tecnologia OpenFlow e a necessidade de ferramentas de *benchmarking* acompanharem esses avanços.

  De um modo geral, podemos afirmar que o OpenFlow desempenha um papel crucial ao permitir a comunicação entre os controladores SDN e os dispositivos de rede, e uma implementação eficaz é essencial para otimizar o desempenho e a eficiência das implantações SDN.

No contexto de compatibilidade de controladores e protocolos, o Ryu surge como a escolha mais popular para controladores SDN por várias razões:

- Suporte para a última versão do OpenFlow: O Ryu suporta a versão mais recente do OpenFlow. Isso garante compatibilidade com os recursos e aprimoramentos mais recentes na tecnologia SDN.
- Desenvolvimento ativo e comunidade: O Ryu possui uma comunidade de desenvolvimento vibrante e ativa, com o código sendo regularmente atualizado para suportar a última versão do OpenFlow. Isso garante que o Ryu permaneça atualizado com os padrões e tecnologias SDN em evolução.
- Documentação e testes abrangentes: O Ryu fornece testes unitários completos, documentos de desenvolvimento e verificações de compatibilidade com switches que utilizam o protocolo OpenFlow. Esse nível de documentação e testes ajuda na compreensão e utilização eficaz do controlador.
- Desempenho: Em avaliações de desempenho, o Ryu mostrou um desempenho competitivo, especialmente quando otimizado com tecnologias como PyPy. Com a capacidade de lidar com um alto volume de pacotes por segundo, pode ser adequado para vários cenários.

  Em resumo, a combinação de suporte para os padrões mais recentes, o desenvolvimento ativo, o bom desempenho e compatibilidade com outras tecnologias torna o Ryu uma escolha popular e acertada para implantações SDN.

3.3.5 Ataques cibernéticos

De uma forma geral, podemos afirmar que existem três tipos de mecanismos de ataques: [Sumantra and Gandhi, 2020]

1. Ataques baseados em volume: estes ataques inundam a rede com um grande volume de tráfego, sobrecarregando o sistema e tornando-o indisponível para utilizadores. Um exemplo de ataque baseado em volume é o ataque de *Ping flood*, no qual o *botnet* envia múltiplas solicitações de *ping* para a vítima [Sumantra and Gandhi, 2020] .

2. Ataques baseados em protocolo: estes ataques exploram vulnerabilidades nos protocolos de rede para interromper o sistema. Um exemplo de ataque baseado em protocolo é o ataque de *TCP SYN flood*, no qual o atacante explora o mecanismo de aperto de mão de três vias do TCP para tornar a conexão embrionária, não enviando o *ACK final* correspondente [Sumantra and Gandhi, 2020].

3. Ataques baseados em aplicações: estes ataques exploram vulnerabilidades na camada aplicacional para interromper o sistema. Um exemplo de ataque baseado em aplicações é o ataque de *Slow HTTP*, no qual os *botnet* enviam muitas sequências de solicitações HTTP incompletas para o servidor da web, sobrecarregando o servidor e causando a negação de serviço [Sumantra and Gandhi, 2020].

4. Ataques *HTTP/2 rapid reset*: Estes ataques exploram a funcionalidade do HTTP/2 para enviar uma série de frames *RST-STREAM* de forma rápida e repetida, forçando o encerramento prematuro das ligações. O ataque *HTTP/2 rapid reset* pode causar uma desestabilização significativa das redes ao interromper múltiplos *streams* simultaneamente, resultando em perda de performance e disponibilidade do serviço. A flexibilidade e o controle centralizado oferecidos pelas SDN são explorados para desenvolver estratégias eficazes de mitigação contra este tipo de ataque [Vidya, 2023].

3.3.6 Estratégias de Defesa e Mitigação

As estratégias de mitigação de ataques informáticos são componentes cruciais num plano de segurança cibernética robusto. Os métodos mais conhecidos são: [Cloudflare, 2022]

1. Proteção ativa - Esta abordagem envolve a implementação de soluções de mitigação de DDoS que estão sempre ativas, permitindo uma monitorização constante ao tráfego de rede para identificar e responder imediatamente a ataques DDoS. Vantagens: resposta rápida e contínua, minimizando o tempo de inatividade e o impacto do ataque. Desvantagens: custo mais elevado devido à necessidade de recursos constantes de monitorização e mitigação .

2. Mitigação sob demanda - Os serviços entram em ação apenas quando um ataque DDoS é detetado, redirecionando o tráfego para um serviço de *cloud* especializado em mitigação. Vantagens: redução dos custos operacionais e do uso de recursos. Desvantagens: potencial atraso na resposta ao ataque, permitindo que o tráfego malicioso afete o alvo por um período indefinido.

3. Reforço das táticas de proteção - Envolve a implementação de *firewalls*, sistemas IPS, e outras medidas de segurança para criar camadas adicionais de defesa. Vantagens: oferece uma defesa em profundidade, abordando diferentes aspetos e vetores de ataques. Desvantagens: custo elevado pois requer uma manutenção e atualização contínua para garantir eficácia.

Neste ponto, convém realçar a necessidade de existir um equilíbrio entre Segurança e Performance - A chave é encontrar um equilíbrio onde a segurança proteja efetivamente os sistemas de informação, mas sem tornar a aplicação tão complexa ou restritiva que afete negativamente a experiência do utilizador. A segurança deve ser uma facilitadora, não um obstáculo, para o uso eficiente e eficaz do sistema.

4. Redes de distribuição de conteúdo - tem como ponto central, o uso de CDNs e balanceamento de carga para distribuir o tráfego e mitigar ataques DDoS, mantendo a performance. Vantagens: ajuda a manter a disponibilidade e a performance do serviço mesmo quando existe um ataque. Desvantagens: insuficiente para ataques de grande escala ou sofisticados.

5. Proteção na borda - permite a mitigação de ataques DDoS com soluções baseadas na borda da rede, executando a deteção e a mitigação o mais próximo possível da origem do ataque. Vantagens: reduz o tempo de resposta ao ataque e minimiza o impacto no núcleo da rede. Desvantagens: está dependente da capacidade e da cobertura geográfica do provedor de serviços na borda.

6. *Honeypots* - A sua utilização é uma estratégia para mitigar ameaças de *botnet* e outros ataques, pois fornecem uma armadilha para o atacante, permitindo rastrear as atividades do invasor. O *honeypot* deve ser configurado para reproduzir as atividades de um sistema real e interagir com o atacante de maneira semelhante ao serviço real [Jantsch, 2016].

7. *Sandboxes* - São ambientes isolados que permitem a execução de código e a observação de seu comportamento sem riscos para o ambiente de produção. Vantagens: é possível identificar atividades maliciosas e impedir que isso afete o sistema real [Jantsch, 2016].

3.4 Resumo

Esta dissertação propõe uma metodologia inovadora para a deteção e mitigação de ataques de *HTTP/2 rapid reset* em ambientes de SDN. A abordagem central do estudo é a implementação de um IPS que utilize as capacidades avançadas das SDN para monitorizar e responder a ameaças em tempo real.

A componente prática deste projeto envolve a criação de um ambiente de testes utilizando o Mininet para simulação de redes e o controlador Ryu para gestão do tráfego de rede. O objetivo é desenvolver *scripts* específicos em Python que permitam a análise detalhada do tráfego HTTP/2, a identificação de padrões anómalos indicativos de ataques *rapid reset* e a implementação de respostas automatizadas para mitigar essas ameaças.

A revisão bibliográfica revelou que as técnicas de prevenção de *botnets* estão em constante evolução, com uma tendência crescente para a integração de soluções baseadas em SDN.

A gestão centralizada e a flexibilidade da SDN oferecem vantagens significativas na monitorização e resposta a ameaças de *botnets*. Além disso, a capacidade da SDN de reconfigurar dinamicamente a rede permite uma resposta rápida e eficaz a incidentes de segurança, o que é crucial na luta contra *botnets* cada vez mais sofisticados. Por fim, a implementação de mecanismos como P4-int e OpenFlow num ambiente SDN demonstrou ser eficaz na recolha da informação e análise do tráfego, que são essenciais para identificar padrões de tráfego anormais associados a *botnets*.

Capítulo 4

Metodologia e planeamento

A opção de utilizar SDN para aprimorar a deteção e prevenção de *botnets* representa um passo significativo na luta contra ameaças cibernéticas. Através da recolha eficiente de dados e da análise avançada, é possível não apenas identificar ameaças existentes, mas também adaptar-se rapidamente a novas táticas utilizadas por *botnets*.

Acredito que este projeto tem o potencial, não só de estabelecer novos padrões na segurança de redes e na proteção contra ameaças cibernéticas sofisticadas, como também de possibilitar o desenvolvimento de ferramentas de registo de dados em SDN. Permitirá ainda monitorizar e registar o tráfego de rede em vários pontos da rede, assegurando que o registo de dados seja abrangente e possa ser adaptado para focar em tipos específicos de tráfego ou comportamento de rede. Além disso, será possível integrar os dados registados com sistemas de deteção de anomalias para identificar padrões de tráfego de *botnets*, bem como, utilizar técnicas de aprendizado de máquina para melhorar a precisão da deteção de *botnets*.

A utilização da SDN permitirá também realizar testes em ambientes controlados que permitirão validar a eficácia dos mecanismos de registos de dados e implementar uma solução em ambientes reais para testar sua eficácia em condições dinâmicas e variadas. Será possível analisar o impacto do registo de dados no desempenho da rede e otimizar, minimizando a latência e o uso de recursos, ajustar os algoritmos com base nos resultados dos testes para equilibrar eficácia e eficiência. Por fim, esta opção também permitirá criar um IPS especificamente para mitigar o ataque *HTTP/2 rapid reset* , utilizando a linguagem Python e o controlador Ryu .

4.1 Caso de estudo e abordagem proposta

Neste capítulo, exploramos o objetivo principal deste trabalho: a mitigação do ataque *HTTP/2 rapid reset* por meio do desenvolvimento de um IPS eficiente e robusto. Esta abordagem visa especificamente combater as vulnerabilidades associadas ao protocolo HTTP/2, que podem ser exploradas por ataques de *HTTP/2 rapid reset*. Através da implementação de um IPS personalizado, pretendemos

não apenas neutralizar esses ataques em tempo real, mas também fortalecer a segurança da rede contra futuras ameaças similares, garantindo assim a integridade dos sistemas de comunicação baseados em HTTP/2.

4.1.1 HTTP/2 rapid reset

Este ataque pode ser caracterizado como inovador e de escala sem precedentes, contudo as proteções existentes da Cloudflare conseguiram absorver grande parte da sua carga. Numa fase inicial, verificou-se algum impacto no tráfego dos clientes, afetando cerca de 1 por cento das solicitações na primeira onda de ataques. No entanto, a Cloudflare, ao refinar os métodos de mitigação, conseguiu interromper o ataque a todos os seus clientes sem afetar os seus sistemas. Este ataque foi notável, pois partindo do uso de uma *botnet* relativamente pequena (cerca de 20.000 máquinas) conseguiu gerar um volume imenso de solicitações, explorando vulnerabilidades no protocolo HTTP/2 [Desgats and Pardue, 2023].

- Vulnerabilidade explorada: este ataque explorou características do protocolo HTTP/2 e detalhes de implementação do servidor, especificamente mencionados no CVE-2023-44487. A vulnerabilidade resulta da forma como o HTTP/2 lida com múltiplos fluxos e o uso de *frames RST-STREAM* para cancelar solicitações.

- Deteção e mitigação: as proteções existentes da Cloudflare foram, na sua maioria eficazes contra o ataque e refinaram os seus métodos de mitigação para proteger todos os clientes sem impactar os seus sistemas. A Cloudflare, juntamente com o Google e a AWS, coordenou a divulgação do ataque aos fornecedores afetados e aos provedores de infraestruturas críticas. A mitigação contra esse tipo de ataque envolve fechar a conexão TCP inteira quando o abuso é detetado. O HTTP/2 suporta a finalização das conexões usando o tipo de quadro *GOAWAY*. As mitigações para esse vetor de ataque podem assumir várias formas, mas geralmente concentram-se em rastrear estatísticas de conexão e usar vários sinais e lógica.

- Detalhe técnico: os atacantes utilizaram os recursos das 20.000 máquinas para gerar um grande número de solicitações e *rapid resets*, sobrecarregando os servidores e causando uma condição de negação de serviço. A exploração dessa vulnerabilidade revela uma fraqueza subjacente no protocolo HTTP/2, afetando potencialmente qualquer fornecedor que tenha implementado o mesmo.

- Impacto nos clientes: apesar da maioria das solicitações maliciosas ter sido bloqueada, o enorme tamanho dos ataques causou algum impacto, incluindo um aumento nos níveis de erros 502 nos *data centers* mais afetados.

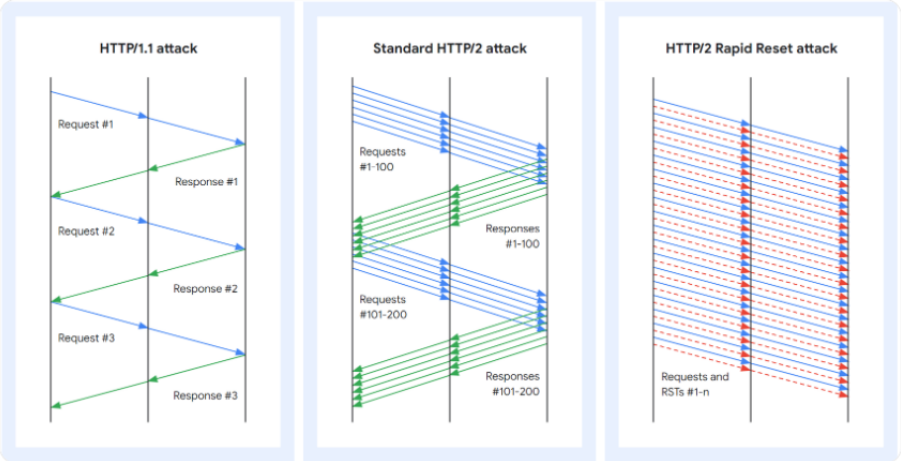


Figura 4.1: Rapid Reset
[Vidya, 2023]

4.1.2 Proposta de mitigação do ataque HTTP/2 rapid reset

Para criar um ambiente de testes eficaz que simule um ataque *HTTP/2 rapid reset* , adotaremos uma abordagem multifacetada, utilizando ferramentas e tecnologias específicas para cada etapa do processo. Para alcançar o objetivo de mitigar o ataque, realizaremos as seguintes etapas:

- **Configuração do Ambiente de Testes**

  VMware: Utilizaremos o VMware para criar Virtual Machine (VM) que hospedarão os componentes necessários para simular o ambiente de rede. Isso inclui a configuração de servidores e clientes que comunicam usando o protocolo HTTP/2.

  Isolamento e controlo: As VM proporcionam um ambiente isolado e controlado, ideal para simular ataques sem riscos para a rede real.

- **Controlador**

  O controlador Ryu: foi escolhido devido à sua flexibilidade e programabilidade, permitindo a criação de *scripts* para monitorizar e mitigar ataques *HTTP/2 rapid reset*. Com suporte ao protocolo OpenFlow, ampla adoção e uma comunidade ativa, Ryu facilita a implementação e manutenção de ambientes de teste. Suas capacidades de monitorização e análise detalhada de pacotes, junto com a escalabilidade e desempenho, são cruciais para simular ataques em larga escala. Além disso, sua integração com VMs no VMware e o emulador Mininet permite criar um ambiente de testes isolado e controlado. A facilidade de uso e implementação em Python, aliada ao histórico de sucesso em diversos estudos e projetos, reforça a escolha do Ryu como a solução ideal para a mitigação de ataques cibernéticos.

- **Protocolo**

  OpenFlow foi utilizado, dado que tem uma menor curva de aprendizagem e este protocolo é suportado por uma maior tipo de dispositivos. O P4 exige

hardware especifico e tem um ciclo de desenvolvimento associado que é complexo de manter.

- **Rede virtual**

  Emulador Mininet: permitirá criar uma rede virtual e possibilitará testar tipologias de rede.

- **Simular o ataque**

  Para simular o ataque *HTTP/2 rapid reset*, foram desenvolvidos dois *scripts*, um para o cliente e outro para o servidor. O *script* do cliente é responsável por iniciar uma ligação HTTP/2 com o servidor e enviar uma série de *frames RST-STREAM* rapidamente para interromper os *streams* estabelecidos. Este *script* foi projetado para gerar um volume significativo de tráfego de *reset*, simulando um ataque realista. Adicionalmente, a ferramenta hping3 foi utilizada para gerar tráfego de rede adicional e intensificar o ataque, incluindo tipos específicos de ataques como *SYN Flood*, *ACK Flood* e *FIN Flood*, tornando a simulação ainda mais próxima de um cenário real. A utilização do controlador Ryu, com suas capacidades avançadas de monitorização e controlo de tráfego, permite ajustar dinamicamente os parâmetros do ataque e implementar estratégias de mitigação em tempo real. A integração com VMs no VMware e o emulador Mininet proporciona um ambiente de teste isolado e controlado, garantindo que os ataques não afetem a rede real. Esta abordagem não só valida a eficácia dos mecanismos de mitigação, como também demonstrou a capacidade de mitigar eficazmente os ataques *SYN Flood*, *ACK Flood* e *FIN Flood* gerados pelo hping3, evidenciando a robustez do Ryu em gerir ataques cibernéticos complexos.

- **Análise de Pacotes com Wireshark**

  Wireshark: esta ferramenta de análise de rede será usada para capturar e analisar pacotes de rede. Com o Wireshark, podemos inspecionar detalhadamente os pacotes HTTP/2 para identificar padrões característicos do ataque *HTTP/2 rapid reset*.

  Diagnóstico e Compreensão do Ataque: A análise de pacotes permitirá entender melhor o ataque e verificar a eficácia das medidas de mitigação.

- **Desenvolvimento da estratégia de Mitigação**

  Utilizaremos Python para desenvolver a estratégia de mitigação, através de um *script* e recorrendo ao controlador Ryu.

  Resposta ao Ataque: O *script* monitorizará o tráfego na rede através do controlador Ryu. Na componente prática da minha dissertação, os padrões de tráfego anormais são detetados utilizando técnicas de comportamento histórico do tráfego para identificar desvios significativos. Especificamente, utilizamos algoritmos de clustering e deteção de anomalias para monitorar métricas como taxa de pacotes, volume de tráfego, e tempo de resposta. Quando um ataque é detetado, seja por padrões de tráfego anormais ou por alertas de uma API, o *script* bloqueia as ligações no controlador.

- **Implementação e Testes** Implementação da estratégia de mitigação: Após o desenvolvimento da estratégia, o *script* será avaliado e testado num ambiente de testes.

  Testes e Ajustes: Realizaremos testes para avaliar a eficácia do *script* na deteção e mitigação do ataque *HTTP/2 rapid reset*. Com base nos resultados recolhidos com o foco na segurança e performance, faremos os ajustes necessários para melhorar a resposta do sistema.

  Este ambiente de testes proporcionará uma visão abrangente da dinâmica do ataque *HTTP/2 rapid reset* e da eficácia das estratégias de mitigação. A combinação de VMware, Ryu, Wireshark e a programação em Python criará um ecossistema robusto para desenvolver, testar e aperfeiçoar um sistema de prevenção de intrusões eficaz contra este tipo específico de ataque.

4.2 Metodologia DSR

A metodologia DSR é uma abordagem de pesquisa utilizada principalmente em disciplinas como sistemas de informação e tecnologia da informação. Tem como principal objetivo o desenvolvimento e a avaliação de tecnologias, sistemas ou processos para resolver problemas identificados. A DSR é particularmente relevante para projetos que visam inovar ou melhorar práticas existentes através da criação de soluções práticas.

Os principais componentes da metodologia DSR incluem:

- Identificação do Problema e Motivação: Esta fase envolve a identificação clara do problema a ser resolvido e a justificação para a necessidade de uma solução. Neste ponto, é crucial entender o contexto e as necessidades dos utilizadores finais.

- Definição dos Objetivos da Solução: Após identificar o problema, são definidos os objetivos da solução. Estes objetivos devem ser mensuráveis e alcançáveis, e servem como elemento base para o desenvolvimento do artefacto.

- Design e Desenvolvimento: Nesta etapa, o artefacto (seja um processo, produto, modelo ou método) é criado. O design deve ser interativo, permitindo ajustes com base no *feedback* e nas descobertas feitas durante o processo de desenvolvimento.

- Demonstração: O artefacto desenvolvido é demonstrado num contexto que simula ou replica um ambiente real, pois só assim é possível mostrar como o artefacto resolve o problema.

- Avaliação: A avaliação envolve testar a eficácia do artefacto em responder aos objetivos definidos. A avaliação pode incluir métodos quantitativos e qualitativos para recolher dados sobre o desempenho e a utilidade do artefacto.

- Comunicação: Os resultados do projeto, incluindo o conhecimento gerado pelo desenvolvimento e avaliação do artefacto, são comunicados a públicos relevantes, como académicos e profissionais.

Várias atividades neste projeto da dissertação foram moldadas tendo por base os princípios da abordagem DSR:

- Recolha de Dados: A utilização de tecnologias SDN, juntamente com o controlador Ryu e protocolo openflow é crucial para a recolha eficiente de métricas e análise de tráfego de rede. Estas tecnologias permitem uma visão detalhada do tráfego, facilitando a identificação de padrões suspeitos.
- Análise e Investigação: O desenvolvimento de *scripts* em Python para analisar tráfego HTTP/2 é uma etapa vital. A análise de cabeçalhos de pacotes, frequência de requisições e tamanhos de payloads pode revelar atividades anormais ou maliciosas, que são indicativos de *botnets*.
- Implementação e Avaliação: Integrar a metodologia no controlador SDN Ryu e realizar avaliações experimentais é fundamental para testar a eficácia das soluções propostas. Isso permite ajustar e melhorar os modelos e estratégias de deteção e prevenção.
- Documentação e Comunicação: A documentação detalhada de todas as fases do projeto é essencial para a disseminação do conhecimento. O resultado dos dados contribui, não só para a comunidade académica, mas também para profissionais da área de segurança cibernética que podem aplicar esses conhecimentos em cenários reais.

A abordagem prevista no contexto deste trabalho é particularmente relevante no contexto atual da segurança cibernética, onde as *botnets* representam uma ameaça significativa. A utilização da SDN para deteção e prevenção de *botnets* oferece uma nova dimensão de flexibilidade e eficácia, permitindo uma resposta mais rápida e adaptável a ameaças emergentes.

Capítulo 5

Implementação do Ambiente de testes

5.1 Metodologia

Esta secção, descrevemos detalhadamente a metodologia e as etapas necessárias para a implementação do ambiente de testes. Exploramos a configuração do controlador Ryu, a integração de *script* de deteção e mitigação, e a validação das medidas implementadas..

5.1.1 Descrição do Ambiente de Testes

Ferramentas Utilizadas

Para a criação do ambiente de testes foram utilizadas as seguintes ferramentas:

- **Mininet**: Ferramenta de emulação de rede que permite a criação de uma rede virtual de switches, *hosts*, links e controladores em um único sistema [min, 2022].

  Aplicação no Ambiente de Testes: No ambiente de testes, Mininet foi utilizado para simular uma rede completa, possibilitando a criação de topologias complexas de rede sem a necessidade de hardware físico. Esta capacidade de emulação permitiu testar e validar diferentes cenários de ataque e defesa de forma eficiente e económica.

- **Ryu**: é um controlador de redes definido por software SDN que oferece uma plataforma para o desenvolvimento de aplicativos de rede flexíveis e programáveis [ryu].

  Aplicação no Ambiente de Testes: No ambiente de testes, Ryu foi utilizado como o controlador SDN principal. Ele gere o comportamento da rede em tempo real, implementando políticas de rede, roteamento e resposta a incidentes de segurança. A flexibilidade do Ryu permitiu a integração de

scripts personalizados para a mitigação de ataques e a monitorização do tráfego de rede.

- **Flask**: é um micro *framework* web programado em Python, projetado para ser leve e fácil de usar para o desenvolvimento de aplicações web [fla, 2024].

  Aplicação no Ambiente de Testes: Flask foi utilizado para criar uma interface gráfica para a monitorização e controlo do ambiente de testes. A interface desenvolvida com Flask permitiu a visualização em tempo real do estado da rede, dos eventos de segurança e das métricas de desempenho, além de possibilitar a interação com os controles de mitigação e resposta a incidentes.

Topologia da Rede Criada para os Testes

A topologia da rede criada consiste num único switch ligado a cinco *hosts*. Um dos *hosts* está configurado como Network Address Translation (NAT) e possui uma ligação direta à internet. Esta configuração é fundamental para testar eficazmente as funcionalidades e os mecanismos de segurança implementados.

Configuração do *Host* NAT:

O *host* configurado como NAT atua como um *gateway* para a internet, permitindo que os outros *hosts* da rede interna acessem recursos externos. Esta configuração cria um cenário realista em que um segmento da rede tem acesso à internet, similar a muitas implementações de rede do mundo real.

```
#!/usr/bin/env python

from mininet.net import Mininet
from mininet.node import Controller, OVSSwitch, RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.clean import Cleanup
import time

def clear_flows(net):
    """Clear flows from all switches in Mininet."""
    print("Clearing all flow entries...")
    for sw in net.switches:
        sw.cmd('ovs-ofctl del-flows %s' % sw.name)

def create_topology():
    """Create and start a Mininet topology."""
    net = Mininet(controller=RemoteController, switch=OVSSwitch, build=
    False)

    # Add Controller
    c0 = net.addController('c0', controller=RemoteController, ip='
    127.0.0.1', port=6633)

    # Add hosts and switch
    hosts = [net.addHost('h%d' % i) for i in range(1, 5)]
    switch = net.addSwitch('s1')

```

```
    # Add links
    for h in hosts:
        net.addLink(switch, h)

    net.build()
    net.addNAT().configDefault()
    net.start()

    return net

def test_network(net):
    """Run network tests such as ping and iperf."""
    print("Rapid Reset attack...")
    # Send TCP RST packets from h2 to h1
    h1, h2 = net.get('h1','h2')
    h1.cmd('python3 Tools/ServerRR.py &')

if __name__ == '__main__':
    setLogLevel('info')  # Set the log level to info to see outputs

    # Create and configure the network
    net = create_topology()

    test_network(net)

    # Open the CLI for interactive input
    CLI(net)

    # Stop the network
    net.stop()

    # Cleanup any remnants of previous Mininet runs
    Cleanup.cleanup()

    # Clear flows and run tests
    clear_flows(net)

    print("Mininet stopped.")
```

Listing 5.1: Mininet Script

Explicação do *script* Python

O *script* Python é estruturado em várias funções e métodos que juntos implementam as funcionalidades de monitorização, deteção e mitigação de ataques na rede. A seguir, detalhamos cada um desses componentes.

O *script* é composto pelos seguintes módulos e funções principais:

- Inicialização e configuração
- Gestão de conexão com Switches
- Manipulação de pacotes

- Funcionalidades de segurança
- Gestão de logs e regras de *firewall*

Explicação do Método __init__ e Variáveis Iniciais

O método __init__ inicializa as variáveis de instância necessárias para o funcionamento do controlador. Configura estruturas de dados para mapear endereços MAC a portas, contar pacotes IP por endereço IP, rastrear pacotes TCP RST e armazenar resultados de análise, através destes parâmetros podemos adaptar o tempo de resposta ao ataque dinamicamente consoante a tipologia de rede.

```
def __init__(self, *args, **kwargs):
    super(IntegratedSwitch, self).__init__(*args, **kwargs)
    self.mac_to_port = {}
    self.ip_packet_counts = defaultdict(lambda: defaultdict(int))
    self.tcp_rst_counts = defaultdict(lambda: defaultdict(lambda:
   defaultdict(list)))
    self.ddos_threshold = 40
    self.rapid_reset_threshold = 10
    self.logger.setLevel(logging.DEBUG)
    self.datapaths = {}
    self.blocked_sources = defaultdict(set)
    self.port_blacklist = {21, 23}
    self.honeypot_ip = '10.0.0.4'
    self.interface = 's1-eth4'
    self.sites_blacklist = {r"facebook.com"}
    self.results = {}
    self.http2_connections = {}
    self.rapid_reset_counts = defaultdict(list)
    self.reset_threshold = 5
    self.topology_data = {'links': []}
    self.topology_logs = {'nodes': []}
```

Listing 5.2: Script Mitigação - Variáveis

Configuração de *logging* e estruturas de dados

Neste método, o nível de *logging* é configurado para *DEBUG*, e são inicializadas várias estruturas de dados para armazenar informações sobre o tráfego e os ataques detetados.

5.1.2 Gestão de switches

O método switch_features_handler é chamado quando um novo switch se liga ao controlador. Ele instala um fluxo para lidar com pacotes desconhecidos e fluxos específicos para bloquear tráfego para portas em listas negras.

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
```

```
    parser = datapath.ofproto_parser

    match = parser.OFPMatch()
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)

    for port in self.port_blacklist:
        self.logger.info(f"Register blacklist ports: {port}")
        match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, ip_proto=in_proto.IPPROTO_TCP, tcp_dst=port)
        self.add_flow(datapath, 1, match, [])

        match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, ip_proto=in_proto.IPPROTO_UDP, udp_dst=port)
        self.add_flow(datapath, 1, match, [])

        self.update_fwrules("Any", "Any", "Any", port, "TCP/UDP", "Black List", "Drop", rule=True)
```

Listing 5.3: Script mitigação - gestão switches

5.1.3 Manipulação de pacotes

O método `detect_packet_type` é responsável por identificar o tipo de pacote recebido (IP, TCP, etc.), essencial para aplicar políticas de segurança adequadas.

- **Classificação de pacotes**: Em uma rede, diferentes tipos de pacotes (e.g., ARP, IPv4, TCP, UDP) circulam constantemente. Identificar o tipo de pacote permite que o sistema classifique e trate cada um de acordo com suas características e requisitos específicos.
- **Aplicação de políticas de segurança**: Cada tipo de pacote pode exigir políticas de segurança diferentes. Por exemplo, pacotes TCP podem ser verificados para detetar ataques de *rapid reset*, enquanto pacotes ARP podem ser monitorados para prevenir ataques de *spoofing*. Sem a identificação correta do tipo de pacote, seria impossível aplicar a política de segurança adequada.
- **Deteção de anomalias**: A deteção de padrões anómalos de tráfego depende da correta identificação do tipo de pacote. Por exemplo, um número anormalmente alto de pacotes RST (*Reset*) em conexões TCP pode indicar um ataque. Sem distinguir pacotes TCP de outros tipos, tais anomalias poderiam passar despercebidas.

```
def detect_packet_type(self, pkt):
    eth = pkt.get_protocols(ethernet.ethernet)[0]
    if eth.ethertype == ether.ETH_TYPE_ARP:
        return "ARP"
    elif eth.ethertype == ether.ETH_TYPE_IPV6:
        return "IPv6"
    elif eth.ethertype == ether.ETH_TYPE_MPLS:
```

```
        return "MPLS"
    elif eth.ethertype == ether.ETH_TYPE_LLDP:
        return "LLDP"
    elif eth.ethertype == ether.ETH_TYPE_IP:
        ip = pkt.get_protocol(ipv4.ipv4)
        if ip:
            if ip.proto == 6:
                return "TCP"
            elif ip.proto == 17:
                return "UDP"
            elif ip.proto == 1:
                return "ICMP"
            elif ip.proto == 2:
                return "IGMP"
            elif ip.proto == 132:
                return "SCTP"
            elif ip.proto == 2048:
                return "ARP2"
        return "IPv4"
    else:
        return "Unknown Ethernet Type"
```

Listing 5.4: Script mitigação - identificação pacotes

Mapeamento MAC-to-Port

O método packet_in_handler atualiza o mapeamento de endereços MAC para portas, essencial para a funcionalidade de comutação.

Identificação e bloqueio de sites/Ips em listas negras

O método _siteblocked verifica se o tráfego está direcionado a sites ou endereços ips em listas negras e bloqueia o tráfego conforme necessário. Este género de listas é essencial para manter o tempo de resposta do controlador otimizado.

```
def _siteblocked(self, pkt, msg, site):
    try:
        eth_pkt = pkt.get_protocol(ethernet.ethernet)
        ip_pkt = pkt.get_protocol(ipv4.ipv4)
        tcp_pkt = pkt.get_protocol(tcp.tcp)

        if eth_pkt and ip_pkt and tcp_pkt:
            tcp_header_length = tcp_pkt.offset * 4
            ethernet_header_length = 14
            ip_header_length = 20
            payload_start = ethernet_header_length + ip_header_length +
    tcp_header_length
            total_packet_length = len(msg.data)
            payload_length = total_packet_length - payload_start
            payload_data = msg.data[payload_start:payload_start +
   payload_length].decode('utf-8', errors='ignore')

            if re.search(site, payload_data, re.IGNORECASE):
                self.logger.info(f"Blocked traffic to {site}")
```

```
                return True
    except Exception as e:
        self.logger.error(f"Error processing packet: {e}")
    return False
```

Listing 5.5: Script mitigação - lista negra

5.1.4 Funcionalidades de Segurança

Deteção e Mitigação de Ataques DDoS

O método packet_in_handler detecta aumentos no número de pacotes IP de um determinado IP de origem para um IP de destino e bloqueia a fonte se o número exceder um limite definido.

```
if eth.ethertype == ether_types.ETH_TYPE_IP:
    ip_pkt = pkt.get_protocol(ipv4.ipv4)
    self.ip_packet_counts[ip_pkt.src][ip_pkt.dst] += 1
    self.update_fwrules(ip_pkt.src, "Any", ip_pkt.dst, "Any", protocol,
    "Any", "Drop", rule=False)

    for src_ip, dst_dict in self.ip_packet_counts.items():
        for dst_ip, count in dst_dict.items():
            if count > self.ddos_threshold:
                self.block_source(datapath.id, src_ip, dst_ip, "Any",
   protocol, "DDOS", is_rapid_reset=False)
```

Listing 5.6: Script mitigação - número pacotes

Deteção de *resets* rápidos de TCP

O método detect_rapid_reset é responsável pela deteção de *rapid reset* em conexões TCP, não se limitando ao HTTP/2. Alguns pontos chave desta função:

Processamento de pacotes TCP:

Analisa pacotes TCP para identificar *flags* de reset (RST). Monitoriza *resets* em diferentes portas e endereços IP.

Contagem de *resets*:

Mantém contadores de *resets* por endereço IP de origem e destino e por porta de destino. Avalia se o número de *resets* em um curto período indica um comportamento anormal.

Deteção de padrões suspeitos:

Deteta padrões de *resets* que podem indicar tentativas de ataque, como múltiplos *resets* em portas distintas ou combinações suspeitas de *flags* (como RST+SYN).

Ação e mitigação:

Toma medidas como bloquear o IP de origem ou redirecionar o tráfego suspeito para um *honeypot*. Ajusta políticas de rede com base na deteção de padrões suspeitos.

```
1     def detect_rapid_reset(self, tcp_pkt, ip_src, ip_dst, dst_port,
    protocol, datapath_id):
2         """Detects rapid TCP resets."""
3         flags = tcp_pkt.bits
4         rst = flags & tcp.TCP_RST
5         ack = flags & tcp.TCP_ACK
6         syn = flags & tcp.TCP_SYN
7         threshold_count = 2  # Number of RSTs to consider before taking
     action
8         threshold_time = 30  # Time window (in seconds) to consider for
     rapid resets
9         current_time = time.time()  # Current time for timestamping RST
     packets
10
11        # Record timestamp before executing the defense mechanism
12        start_time = time.time()
13
14        # Detect unusual RST packets
15        if rst:
16            # Append the time of the RST packet to the appropriate list
17            self.tcp_rst_counts[ip_src][ip_dst][dst_port].append(
    current_time)
18
19            # Evaluate whether multiple distinct ports are targeted by
    checking port count over threshold
20            ports_over_threshold = {port: times for port, times in self
    .tcp_rst_counts[ip_src][ip_dst].items() if len(times) >
    threshold_count}
21
22            # If multiple distinct ports are involved
23            if len(ports_over_threshold) > 3:
24                formatted_ports = ', '.join(str(port) for port in
    ports_over_threshold.keys())
25                self.logger.warning(f"Rapid reset attack likely from {
    ip_src} to {ip_dst} on multiple ports: {formatted_ports}")
26                self.block_source(datapath_id, ip_src, ip_dst, "Any",
    protocol, "RST-PORTS", is_rapid_reset=False)
27                execution_time = time.time() - start_time
28                self.append_to_json_file('analise.json', {"RST-
    PORTS_Execution_Time": execution_time})
29                self.logger.info(f"Defense mechanism execution time: {
    execution_time} seconds")
30
31            # Check the specific port's activity
```

```
32              if len(self.tcp_rst_counts[ip_src][ip_dst][dst_port]) >
    self.rapid_reset_threshold:
33                  self.logger.warning(f"Rapid reset attack likely from {
    ip_src} to {ip_dst}:{dst_port}: {len(self.tcp_rst_counts[ip_src][
    ip_dst][dst_port])} RSTs in last {threshold_time} seconds")
34                  self.block_source(datapath_id, ip_src, ip_dst, dst_port
    , protocol, "RST-COUNT", is_rapid_reset=True)
35                  execution_time = time.time() - start_time
36                  if len(self.tcp_rst_counts[ip_src][ip_dst][dst_port]) <
     15:
37                      self.append_to_json_file('analise.json', {"RST-
    COUNT-HONEYPOT_Execution_Time": execution_time})
38                  else:
39                      self.append_to_json_file('analise.json', {"RST-
    COUNT-15_Execution_Time": execution_time})
40                  self.logger.info(f"Defense mechanism execution time: {
    execution_time} seconds")
41
42              # Detect specific patterns of malicious resets
43              if rst and ack and not syn:
44                  # An RST+ACK packet without SYN could indicate an
    attempt to disrupt an established connection
45                  self.logger.warning(f"RST+ACK without SYN detected from
     {ip_src}")
46                  self.block_source(datapath_id, ip_src, ip_dst, dst_port
    , protocol, "RST-ACK", is_rapid_reset=False)
47                  execution_time = time.time() - start_time
48                  self.append_to_json_file('analise.json', {"RST-
    ACK_Execution_Time": execution_time})
49                  self.logger.info(f"Defense mechanism execution time: {
    execution_time} seconds")
50
51              if rst and syn:
52                  # A packet with both RST and SYN flags could be part of
     a scanning or spoofing attack
53                  self.logger.warning(f"Suspicious RST+SYN packet
    detected from {ip_src}")
54                  self.block_source(datapath_id, ip_src, ip_dst, dst_port
    , protocol, "RST-SYN", is_rapid_reset=False)
55                  execution_time = time.time() - start_time
56                  self.append_to_json_file('analise.json', {"RST-
    SYN_Execution_Time": execution_time})
57                  self.logger.info(f"Defense mechanism execution time: {
    execution_time} seconds")
```

Listing 5.7: Script mitigação - bloqueio

Bloqueio ou limitação de tráfego suspeito

O método block_source bloqueia a fonte de tráfego suspeito ou redireciona tráfego para um *honeypot* consoante o numero de ataques.

```
1 def block_source(self, dpid, src_ip, ip_dst, dst_port, protocol,
    details, is_rapid_reset=False):
2     if dpid not in self.datapaths:
3         self.logger.error(f"Datapath {dpid} not found. Cannot block or
    redirect source: {src_ip}:{dst_port}")
```

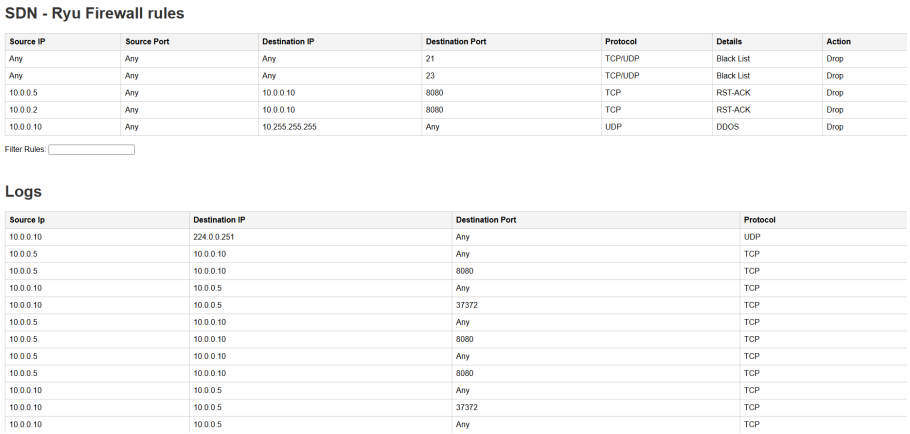
```
        return
    priority = 1000
    datapath = self.datapaths[dpid]
    parser = datapath.ofproto_parser
    ofproto = datapath.ofproto

    if dst_port is not None and dst_port not in self.port_blacklist and
    is_rapid_reset:
        if len(self.tcp_rst_counts[src_ip][ip_dst][dst_port]) < 15:
            self.create_apply_meter(datapath, rate=100)

            match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
    ipv4_src=src_ip, ip_proto=in_proto.IPPROTO_TCP, tcp_dst=dst_port)
            actions = [
                parser.OFPActionSetField(ipv4_dst=self.honeypot_ip),
                parser.OFPActionOutput(ofproto.OFPP_CONTROLLER)
            ]
            inst = [
                parser.OFPInstructionActions(ofproto.
    OFPIT_APPLY_ACTIONS, actions),
                parser.OFPInstructionMeter(meter_id=1)
            ]
            mod = parser.OFPFlowMod(datapath=datapath, priority=
    priority, match=match, instructions=inst)
            datapath.send_msg(mod)
            self.update_fwrules(src_ip, "Any", self.honeypot_ip,
    dst_port, protocol, details, "Speed 100kbs Redirect", rule=True)
        elif len(self.tcp_rst_counts[src_ip][ip_dst][dst_port]) > 15:
            self.logger.info(f"Blocking traffic from {src_ip}:{dst_port
    }")
            priority = 1001
            match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
    ipv4_src=src_ip)
            actions = []
            self.add_flow(datapath, priority, match, actions)
            self.update_fwrules(src_ip, "Any", self.honeypot_ip,
    dst_port, protocol, details, "Drop", rule=True)
            self.blocked_sources[dpid].add((src_ip, dst_port))
    else:
        match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,
    ipv4_src=src_ip, ipv4_dst=ip_dst)
        actions = []
        self.add_flow(datapath, priority, match, actions)
        self.update_fwrules(src_ip, "Any", ip_dst, dst_port, protocol,
    details, "Drop", rule=True)
```

Listing 5.8: Script mitigação - limitação velocidade e honeypot

5.1.5 Gestão de *logs* e regras de *firewall*

Atualização de regras de *firewall*

O método update_fwrules atualiza as regras de *firewall* e regista eventos significativos e atualizações, como podemos verificar na figura 5.1.



SDN - Ryu Firewall rules

Source IP	Source Port	Destination IP	Destination Port	Protocol	Details	Action
Any	Any	Any	21	TCP/UDP	Black List	Drop
Any	Any	Any	23	TCP/UDP	Black List	Drop
10.0.0.5	Any	10.0.0.10	8080	TCP	RST-ACK	Drop
10.0.0.2	Any	10.0.0.10	8080	TCP	RST-ACK	Drop
10.0.0.10	Any	10.255.255.255	Any	UDP	DDOS	Drop

Filter Rules:

Logs

Source Ip	Destination IP	Destination Port	Protocol
10.0.0.10	224.0.0.251	Any	UDP
10.0.0.5	10.0.0.10	Any	TCP
10.0.0.5	10.0.0.10	8080	TCP
10.0.0.10	10.0.0.5	Any	TCP
10.0.0.10	10.0.0.5	37372	TCP
10.0.0.5	10.0.0.10	Any	TCP
10.0.0.5	10.0.0.10	8080	TCP
10.0.0.5	10.0.0.10	Any	TCP
10.0.0.5	10.0.0.10	8080	TCP
10.0.0.10	10.0.0.5	Any	TCP
10.0.0.10	10.0.0.5	37372	TCP
10.0.0.10	10.0.0.5	Any	TCP

Figura 5.1: Firewall

```python
def update_fwrules(self, src_ip, src_port, dst_ip, dst_port, protocol, site, action, rule=False):
    if not rule:
        node_ids = set((node['Source Ip'], node['Destination Ip'], node['Destination Port'], node['Protocol'])
                       for node in self.topology_logs['nodes'])
        node_key = (src_ip, dst_ip)
        self.topology_logs['nodes'].append({'Source Ip': src_ip, 'Destination Ip': dst_ip,
                                            'Destination Port': dst_port, 'Protocol': protocol})
        node_ids.add(node_key)

        with open('logs.json', 'w') as f:
            json.dump(self.topology_logs, f)

    existing_links = set(
        (link['Source Ip'], link['Source Port'], link['Destination Ip'], link['Destination Port'], link['Protocol'])
        for link in self.topology_data['links'])

    new_link = (src_ip, src_port, dst_ip, dst_port, protocol)

    if new_link not in existing_links and rule:
        self.topology_data['links'].append({
            'Source Ip': src_ip,
            'Source Port': src_port,
            'Destination Ip': dst_ip,
            'Destination Port': dst_port,
            'Protocol': protocol,
            'Site': site,
            'Action': action
        })
        self.logger.info(f"Added new link: {new_link}")

    with open('fw_rules.json', 'w') as f:
        json.dump(self.topology_data, f)
```

Listing 5.9: Script mitigação - utualiza regras e eventos

Log de eventos e atualizações significativas

Os *logs* são essenciais para monitorizar e diagnosticar o comportamento da rede. O método update_fwrules inclui a funcionalidade de registo de ações para a consola ou para o ambiente gráfico para registar eventos significativos e atualizações de regras.

5.2 Deteção de ataques HTTP/2 rapid reset

A função http2_rr foi desenvolvida especificamente para lidar com pacotes HTTP/2 e detetar ataques de *rapid reset* no contexto de ligações HTTP/2. Esta função é crucial para identificar rapidamente os padrões de tráfego maliciosos associados a este tipo de ataque, permitindo a aplicação de medidas de mitigação em tempo real.

Processamento de pacotes HTTP/2:

Extrai e processa o *payload* dos pacotes TCP para identificar eventos específicos do protocolo HTTP/2. Utiliza a biblioteca h2 para interpretar *frames* HTTP/2.

Deteção de eventos HTTP/2:

Deteta eventos como *StreamReset*, *StreamEnded*, *DataReceived*, *RequestReceived*, e *ConnectionTerminated*. Para cada evento, realiza ações específicas, como registar o evento ou aumentar contadores de *rapid reset*.

Contagem e avaliação de *rapid reset*:

Mantém um registo de tempos de *rapid reset* para avaliar se há um ataque ocorrendo. Se o número de *resets* em um curto período exceder um limite definido, é identificado como um possível ataque.

Ação e mitigação:

Em caso de deteção de um possível ataque, pode tomar medidas de mitigação, como registar o evento e ajustar políticas de rede.

```
def http2_rr(self, msg):
        """Handles HTTP/2 rapid resets."""
        dp = msg.datapath
        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocol(ethernet.ethernet)
        start_time = time.time()
        if eth.ethertype != 0x0800:  # Not IPv4
            return
```

```
9
10          ip = pkt.get_protocol(ipv4.ipv4)
11          if ip.proto != 6:  # Not TCP
12              return
13
14          tcp_pkt = pkt.get_protocol(tcp.tcp)
15          if not tcp_pkt:
16              return
17
18          ip_header_length = ip.header_length * 4
19          tcp_header_length = tcp_pkt.offset * 4
20          tcp_payload_start = 14 + ip_header_length + tcp_header_length
21          tcp_payload = msg.data[tcp_payload_start:]
22
23          connection_id = (dp.id, ip.src, ip.dst, tcp_pkt.src_port,
    tcp_pkt.dst_port)
24
25          if connection_id not in self.http2_connections:
26              self.http2_connections[connection_id] = H2Connection()
27              self.http2_connections[connection_id].initiate_connection()
28
29          h2_conn = self.http2_connections[connection_id]
30
31          try:
32              events = h2_conn.receive_data(tcp_payload)
33              h2_conn.clear_outbound_data_buffer()
34
35              event_occurred = False  # Flag to check if any event
    occurred
36
37              for event in events:
38                  if isinstance(event, h2.events.StreamReset):
39                      self.logger.error(f"Stream {event.stream_id} reset:
     Error code {event.error_code}")
40
41                      current_time = time.time()
42                      self.rapid_reset_counts[connection_id].append(
    current_time)
43                      self.rapid_reset_counts[connection_id] = [
44                          t for t in self.rapid_reset_counts[
    connection_id] if current_time - t < self.rapid_reset_threshold
45                      ]
46
47                      if len(self.rapid_reset_counts[connection_id]) >
    self.reset_threshold:
48                          self.logger.warning(f"Potential rapid reset
    attack detected for connection {connection_id}")
49                          self.block_source(dp.id, ip.src, ip.dst,
    tcp_pkt.dst_port, "TCP", "Rapid Reset Detected", is_rapid_reset=
    True)
50                      event_occurred = True
51
52                  elif isinstance(event, h2.events.StreamEnded):
53                      self.logger.info(f"HTTP/2 stream {event.stream_id}
    ended on {connection_id}")
54                      h2_conn.end_stream(event.stream_id)
55                      event_occurred = True
56
```

```
57                 elif isinstance(event, h2.events.DataReceived):
58                     self.logger.info(f"Data received on stream {event.
   stream_id}: {len(event.data)} bytes")
59                     event_occurred = True
60
61                 elif isinstance(event, h2.events.RequestReceived):
62                     self.logger.info(f"Request received on stream {
   event.stream_id}: {event.headers}")
63                     event_occurred = True
64
65                 elif isinstance(event, h2.events.ConnectionTerminated):
66                     self.logger.info(f"Connection {connection_id}
   terminated, error code {event.error_code}")
67                     if connection_id in self.http2_connections:
68                         del self.http2_connections[connection_id]
69                     event_occurred = True
70
71             if event_occurred:
72                 execution_time = time.time() - start_time  # Calculate
   execution time
73                 self.append_to_json_file('analise.json', {"
   HTTP2_RR_Execution_Time_Events": execution_time})
74
75         except h2.exceptions.ProtocolError as e:
76             self.logger.error(f"HTTP/2 protocol error: {e}, Connection
   ID: {connection_id}")
77             self.logger.error(f"Error details: {e.error_code} in state
   {h2_conn.state_machine.state}")
78
79             current_time = time.time()
80             self.rapid_reset_counts[connection_id].append(current_time)
81             self.rapid_reset_counts[connection_id] = [
82                 t for t in self.rapid_reset_counts[connection_id] if
   current_time - t < 5
83             ]
84             if len(self.rapid_reset_counts[connection_id]) > self.
   reset_threshold:
85                 self.logger.warning(f"Potential rapid reset attack
   detected for connection {connection_id}")
86                 self.block_source(dp.id, ip.src, ip.dst, tcp_pkt.
   dst_port, "TCP", "Rapid Reset Detected", is_rapid_reset=True)
87             execution_time = time.time() - start_time  # Calculate
   execution time
88             self.append_to_json_file('analise.json', {"
   HTTP2_RR_Execution_Time_Error": execution_time})
89
90         except Exception as e:
91             self.logger.error(f"Error processing HTTP/2 frames: {e},
   Connection ID: {connection_id}")
92
93             execution_time = time.time() - start_time  # Calculate
   execution time
94             self.append_to_json_file('analise.json', {"
   HTTP2_RR_Execution_Time_Exeption": execution_time})
```

Listing 5.10: Script mitigação - http2 rapid reset

5.3 Resumo

O capítulo 5 detalha a implementação de um ambiente de testes para deteção e mitigação de ataques cibernéticos, com um foco especial na mitigação de ataques *HTTP/2 rapid reset*. Através de uma explicação abrangente das funções e métodos utilizados nos *scripts* Python, foram abordadas as técnicas práticas para a deteção e mitigação de ataques utilizando SDN. As funcionalidades de segurança implementadas concentram-se em métodos de bloqueio e limitação de tráfego, assegurando a proteção da rede contra diversas ameaças. Este capítulo fornece uma visão clara de como as tecnologias SDN, juntamente com ferramentas como o Ryu, podem ser utilizadas para criar uma infraestrutura de rede robusta e resiliente, capaz de identificar e responder rapidamente a ataques cibernético

Capítulo 6

Ambiente de Testes para Deteção e Mitigação de Ataques

Neste capítulo, abordaremos a criação de um ambiente de testes robusto e eficiente para a deteção e mitigação de ameaças cibernéticas. O objetivo principal é garantir elevados níveis de segurança e resiliência na rede através de análise em tempo real e gestão adaptativa de ameaças. Vamos detalhar as ferramentas utilizadas, a configuração do ambiente, e os testes realizados para validar as metodologias de segurança implementadas.

6.1 Objetivo

O objetivo principal é criar uma estrutura robusta que garanta níveis elevados de segurança e resiliência na rede. Através da análise em tempo real e da gestão adaptativa de ameaças, este sistema visa ser uma ferramenta crucial na luta contra ameaças cibernéticas complexas, proporcionando um ambiente seguro para operações de rede. A Figura 6.1 ilustra a topologia do sistema, destacando a interconexão entre o controlador Ryu, o switch Mininet e os *hosts* , proporcionando uma visão clara de como os componentes se integram para formar uma defesa coesa e eficiente.

6.2 Funcionalidades principais

Os requisitos funcionais, traduzem-se nas seguintes funcionalidades:

- **Deteção Avançada de *botnets***: Utiliza capacidades da SDN para monitorização do tráfego de rede em tempo real, detetando potenciais ameaças de atividades de *botnet*.
- **Mitigação de DDoS**: Utilização *scripts* Python para respostas dinâmicas a ameaças DDoS, garantindo tempo mínimo de inatividade da rede.

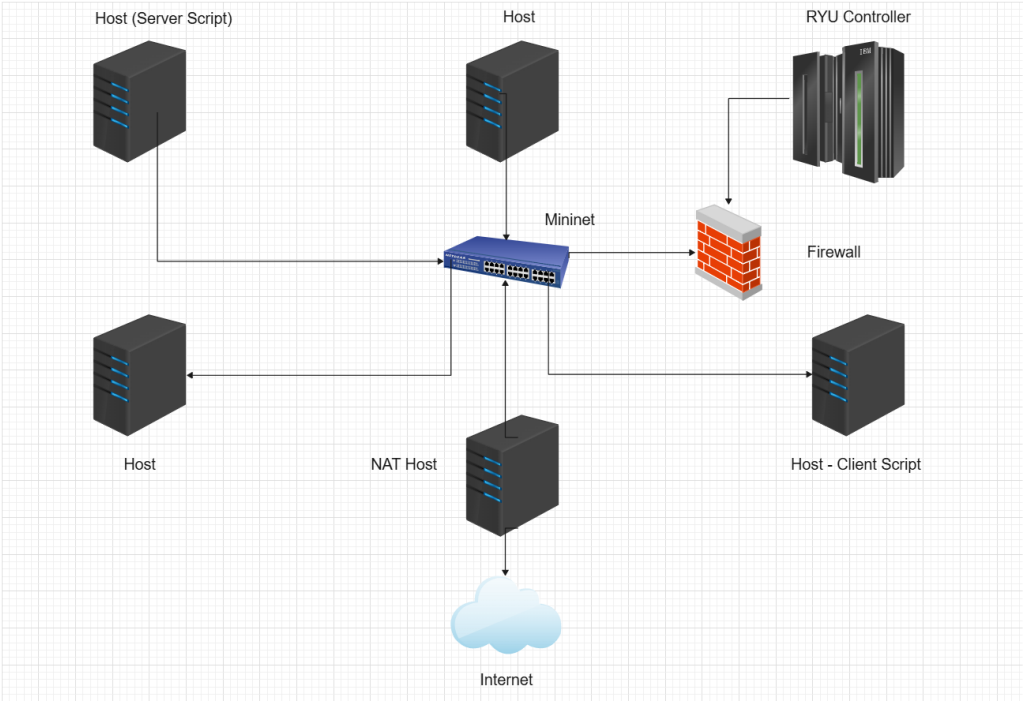


Figura 6.1: Diagrama Sistema

- **Prevenção de Ataques rapid reset**: Integra mecanismos de última geração para prevenir interrupções causadas por inundações de pacotes *rapid reset*.[1]

- **Políticas de Segurança Personalizáveis**: Oferece configurações flexíveis através de uma interface intuitiva, permitindo aos utilizadores personalizar medidas de segurança conforme suas necessidades. Por exemplo, os utilizadores podem definir regras específicas para filtrar tráfego malicioso, ajustar os níveis de sensibilidade dos algoritmos de deteção de anomalias para minimizar falsos positivos, e configurar alertas em tempo real para atividades suspeitas. Além disso, é possível integrar políticas de segurança personalizadas que se adaptam automaticamente a mudanças no padrão de tráfego da rede. como por exemplo o numero de ligações

6.3 Projeto Ryu Integrated Switch

O repositório contém o *IntegratedSwitch*, uma aplicação baseada no controlador Ryu e projetada para gestão de redes, incluindo funcionalidades como redirecionamento de pacotes, deteção de DDoS, deteção de ataques *rapid reset* e gestão do tráfego usando OpenFlow na versão 1.3. Este projeto é compatível com ferramentas de simulação de rede como Mininet, facilitando os testes em ambientes de rede simulados.

[1] A capacidade de adaptação dinâmica e a resposta em tempo real são características que os diferenciam das soluções tradicionais.

6.4 Passo a passo

Os principais passos incluem:

1. Inicialização e Configuração, onde são feitas as necessárias inicializações dos componentes de software
2. Gestão de ligações dos Switches, onde se identifica o tipo de topologia, o seu estado, as diversas ligações, entre outros.

No primeiro passo são realizadas as seguintes fases:

- Inicializar a aplicação com as variáveis necessárias.
- Configurar o *logging* e as estruturas de dados.

No segundo passo a gestão de ligações dos switches:

- Deteção de novas ligações de switches.
- Instalação de fluxos para lidar com pacotes desconhecidos.
- Instalação fluxos para bloquear tráfego para portas em listas negras.
- Gestão desconexões de switches.

O terceiro passo e final a manipulação de pacotes:

- Deteção de tipos de pacotes (IP, TCP, etc.).
- Gestão mapeamentos MAC-para-porta.
- Identificação e bloqueio de sites ou endereços de ips em listas negras.
- Manipulação de tipos específicos de pacotes como IPv6, ARP ou protocolos IP específicos, para facilitar a analise de *logs*.

Funcionalidades de segurança

- **Deteção e manipulação de potenciais ataques DDoS**: O *script* monitoriza o tráfego de rede em tempo real, utilizando contadores para registar o número de pacotes IP enviados por cada endereço IP de origem. Quando o número de pacotes excede um limiar definido (por exemplo, 40 pacotes), o *script* identifica essa atividade como um possível ataque DDoS. Em resposta, o *script* pode bloquear o endereço IP de origem ou aplicar outras medidas de mitigação para proteger a rede.

- **Deteção e manipulação de *rapid resets***: O *script* rastreia pacotes TCP-RST para identificar padrões de tráfego anormais que indicam ataques de *rapid reset*. Utilizando con'tadores e monitorização de tempos, ele deteta atividades suspeitas, como múltiplos *resets* em um curto período. Quando um ataque é detetado, o *script* pode redirecionar o tráfego suspeito para um *honeypot* ou bloquear o endereço IP de origem para mitigar o ataque.

- **Bloqueio ou limitação do tráfego suspeito**: O *script* permite bloquear tráfego de fontes suspeitas ou limitar a largura de banda para certos tipos de tráfego. Utilizando regras de fluxo e medidores, o *script* pode aplicar ações como descartar pacotes ou redirecionar o tráfego para um *honeypot*, bem como limitar a taxa de transferência de dados para minimizar o impacto dos ataques na rede.

- **Gestão de fontes e portas bloqueadas**: O *script* mantém uma lista de endereços IP e portas bloqueadas, atualizando dinamicamente essas listas com base em atividades suspeitas detetadas. Ele também gere as regras da *firewall* para garantir que o tráfego para portas críticas, como 21 e 23, seja bloqueado automaticamente. Além disso, o *script* oferece funcionalidades para registar e analisar eventos de bloqueio, permitindo ajustes contínuos nas políticas de segurança.

Gestão de *logs* e regras

- Atualização das regras de *firewall*.

- Registo de eventos e atualizações significativas.

6.5 Preliminares do desenvolvimento

6.5.1 Pré-requisitos

- Python 3.5 ou superior

- Framework Ryu 4.34

- Mininet 2.3.0

- Flask 3 ou superior

- hping3 3.a2.ds2-10

- iperf 3.xx-1

6.5.2 Instalação

Após a instalação de todos os requisitos, pode-se proceder à instalação da solução. Para tal sugere-se a criação de uma ambiente Python para o desenvolvimento.

Ambiente Python opcional

```
sudo apt install python3.9-venv
python -m venv /path/to/new/virtual/environment
source myvenv/bin/activate
```

Instalação do Ryu

```
pip install Ryu
```

Instalação do mininet

```
sudo apt install mininet
```

Instalação do flask

```
pip install flask
```

Clonar o repositório

```
git clone https://github.com/paulopais/SDN_Project.git
cd Ryu-integrated-switch
```

6.6 Executando a aplicação Ryu

Para iniciar o controlador Ryu:

```
Ryu-manager integrated_switch.py
```

Para iniciar o Flask:

```
sudo python3 gui.py
```

Para iniciar os arquivos *py* do Mininet:

```
sudo python3 mininet_rapidresettest.py
```

2

```
chmod +x mininet.sh
./mininet.sh
```

6.7 Resumo

Em resumo, o Capítulo 6 fornece uma visão de todas etapas e ferramentas necessárias para criar um ambiente de testes eficaz para a deteção e mitigação de

[2]Certifique-se de ter as permissões necessárias para executar o *script*

ameaças cibernéticas, destacando a importância da segurança adaptativa e a gestão em tempo real de incidentes de segurança .

Capítulo 7

Simulação do ataque

7.1 Realização de ataques e deteção

Este capítulo descreve os tipos de testes realizados para detetar e mitigar ataques no sistema de segurança de rede, bem como a eficácia das metodologias implementadas. Os testes foram conduzidos com o objetivo de validar a robustez e a eficiência do sistema em identificar e neutralizar ameaças cibernéticas.

Simulação de Ataques DDoS: Utilizando ferramentas como hping3, foram enviados pacotes em massa para um único destino para o sobrecarregar.

Simulação de Ataques de *rapid resets*: Foram utilizados *scripts* para enviar pacotes TCP com *flags RST* repetidamente.

Simulação de *Scans*: Foram realizados *scans* de portas utilizando pacotes SYN.

Simulação de Acesso a Sites e ips Bloqueados: Foram realizados testes de navegação para sites/ips na lista de bloqueio.

Simulação de Ataques de *HTTP/2 rapid reset*: Utilizando o *script* do cliente que é responsável por simular um ataque de *rapid resets* através de uma série de pedidos HTTP/2 e o *script* do servidor que é responsável por receber os pedidos HTTP/2 e enviar *frames* de *reset* como resposta, simulando um comportamento de servidor vulnerável.

7.1.1 *Script* do Cliente

O *script* do cliente é responsável por simular um ataque de *rapid resets* através de uma série de pedidos HTTP/2. Explicação do código do cliente.

```
1 import socket
2 import time
3 from h2.config import H2Configuration
4 from h2.connection import H2Connection
5 from h2.events import ResponseReceived, DataReceived, StreamReset,
      StreamEnded
6 import h2.exceptions
```

```
7
8 HOST = '10.0.0.5'
9 PORT = 8080
10
11 def send_http2_request(connection, stream_id):
12     headers = [
13         (':method', 'GET'),
14         (':authority', f'{HOST}:{PORT}'),
15         (':scheme', 'http'),
16         (':path', '/')
17     ]
18     connection.send_headers(stream_id, headers, end_stream=True)
19     return connection.data_to_send()
20
21 def trigger_rapid_resets(connection, sock, stream_id):
22     for _ in range(10):
23         try:
24             data_to_send = send_http2_request(connection, stream_id)
25             sock.sendall(data_to_send)
26             time.sleep(0.1)  # delay to simulate rr attacks
27             connection.reset_stream(stream_id)  #  RST_STREAM
28             sock.sendall(connection.data_to_send())
29             stream_id += 2  # Next id
30         except Exception as e:
31             print(f'Error: {e}')
32
33 def handle_http2_response(connection, conn):
34     while True:
35         try:
36             data = conn.recv(65535)
37             if not data:
38                 break
39
40             events = connection.receive_data(data)
41             for event in events:
42                 if isinstance(event, ResponseReceived):
43                     print(f'Received response on stream {event.
   stream_id}')
44                 elif isinstance(event, DataReceived):
45                     print(f'Received data on stream {event.stream_id}')
46                     if event.stream_ended:
47                         print(f'Stream {event.stream_id} ended')
48                         connection.end_stream(event.stream_id)  # Marca
    o fim do fluxo
49                 elif isinstance(event, StreamReset):
50                     print(f'Stream {event.stream_id} was reset by the
   server')
51                 elif isinstance(event, StreamEnded):
52                     print(f'Stream {event.stream_id} ended normally')
53
54             conn.sendall(connection.data_to_send())
55         except h2.exceptions.ProtocolError as e:
56             print(f'HTTP/2 protocol error: {e}, error code: {e.
   error_code}')
57             break
58         except Exception as e:
59             print(f'Error processing HTTP/2 frames: {e}')
60             break
```

```
61
62 def main():
63     print(f'Attempting to connect to {HOST}:{PORT}')
64     try:
65         sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
66         sock.connect((HOST, PORT))
67         print(f'Connected to {HOST}:{PORT}')
68
69         h2_config = H2Configuration(client_side=True)
70         h2_connection = H2Connection(config=h2_config)
71         h2_connection.initiate_connection()
72
73         sock.sendall(h2_connection.data_to_send())
74
75         stream_id = 1
76         trigger_rapid_resets(h2_connection, sock, stream_id)
77
78         handle_http2_response(h2_connection, sock)
79         sock.close()
80     except ConnectionRefusedError as e:
81         print(f'Connection refused: {e}')
82     except Exception as e:
83         print(f'Error: {e}')
84
85 if __name__ == '__main__':
86     main()
```

Listing 7.1: Script cliente - http2 rapid reset

7.1.2 *Script* do servidor

O *script* do servidor é responsável por receber os pedidos HTTP/2 e enviar *frames* de *reset* como resposta, simulando um comportamento de servidor vulnerável a ataques *rapid reset*.

```
1 import socket
2 from h2.config import H2Configuration
3 from h2.connection import H2Connection
4 from h2.events import RequestReceived, DataReceived,
     RemoteSettingsChanged, StreamReset
5 from h2.exceptions import NoSuchStreamError
6
7 HOST = '10.0.0.5'
8 PORT = 8080
9
10 def send_reset_frame(connection, stream_id):
11     """
12     Send 1 reset frame to the client.
13     """
14     try:
15         print(f'Sending Reset - {stream_id}')
16         connection.reset_stream(stream_id)
17         return connection.data_to_send()
18     except NoSuchStreamError:
19         print(f"Attempted to reset a non-existent or closed stream: {
     stream_id}")
```

```
        return None

def handle_http2_connection(conn):
    """
    New conenction HTTP2.
    """
    h2_config = H2Configuration(client_side=False)
    h2_connection = H2Connection(config=h2_config)
    h2_connection.initiate_connection()

    conn.sendall(h2_connection.data_to_send())

    while True:
        try:
            data = conn.recv(65535)
            if not data:
                break

            events = h2_connection.receive_data(data)
            for event in events:
                if isinstance(event, RequestReceived):
                    print(f'Received request on stream {event.stream_id}')
                    data_to_send = send_reset_frame(h2_connection, event.stream_id)
                    if data_to_send:
                        conn.sendall(data_to_send)
                elif isinstance(event, DataReceived):
                    if event.stream_ended:
                        h2_connection.end_stream(event.stream_id)
                elif isinstance(event, RemoteSettingsChanged):
                    pass
                elif isinstance(event, StreamReset):
                    print(f"Stream Reset Received for stream ID: {event.stream_id}")

            conn.sendall(h2_connection.data_to_send())
        except Exception as e:
            print(f'Error processing HTTP/2 frames: {e}')
            break

    conn.close()

def main():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind((HOST, PORT))
    sock.listen(5)

    print(f'Serving on http://{HOST}:{PORT}')

    while True:
        conn, _ = sock.accept()
        handle_http2_connection(conn)

if __name__ == '__main__':
    main()
```

Listing 7.2: Script servidor - http2 rapid reset

7.1.3 Execução dos *scripts*

Para executar os *scripts*, siga os seguintes passos:

1. Inicie o servidor primeiro, executando o *script* do servidor.
2. Em seguida, execute o *script* do cliente para iniciar os ataques de *rapid reset*.

Execução do *Script* do servidor

No terminal Mininet execute o seguinte comando:

```
h1 python server_script.py &
```

Execução do *script* do cliente

No mesmo terminal execute o seguinte comando:

```
h2 python client_script.py
```

7.1.4 Ataque RST-PORTS

O ataque "RST-PORTS"consiste em enviar pacotes `RST` para múltiplas portas de destino, simulando um ataque de `rapid reset` em várias portas simultaneamente. Para realizar este ataque, utilizamos os seguintes comandos:

```
h3 hping3 -p 86 -c 3 --rst 10.0.0.1 &
h3 hping3 -p 87 -c 3 --rst 10.0.0.1 &
h3 hping3 -p 88 -c 3 --rst 10.0.0.1 &
h3 hping3 -p 89 -c 3 --rst 10.0.0.1 &
```

Estes comandos enviam pacotes `RST` para as portas 86, 87, 88 e 89 do IP 10.0.0.1, com três pacotes por porta, simulando um ataque distribuído.

7.1.5 Ataque RST-COUNT

O ataque "RST-COUNT"foca em enviar um número elevado de pacotes `RST` para uma única porta de destino, simulando um ataque de `rapid reset`. O comando utilizado é:

```
h2 hping3 -p 81 -c 12 --rst 10.0.0.1
```

Este comando envia 12 pacotes `RST` para a porta 81 do IP 10.0.0.1, testando a capacidade do sistema em detetar e mitigar ataques rápidos e repetitivos na mesma porta.

7.1.6 Ataque RST-ACK

O ataque "RST-ACK"envolve o envio de pacotes com *flags* `RST` e `ACK`, que podem ser usados para testar a resiliência do sistema contra pacotes de *reset* que também confirmam a receção de dados. O comando utilizado é:

```
h2 hping3 -V -S -p 70 10.0.0.1
```

Este comando envia pacotes TCP *SYN* para a porta 70 do IP 10.0.0.1 com *verbose mode* ativado, simulando um cenário onde pacotes maliciosos tentam estabelecer e imediatamente cancelar as ligações.

7.1.7 Análise dos resultados

Os resultados dos ataques serão registados e processados pelo controlador Ryu, que utilizará os métodos implementados para detetar e mitigar os ataques. O método `detect_rapid_reset` será particularmente importante para identificar os ataques *rapid reset* e tomar medidas apropriadas.

7.2 Resumo

Este capítulo detalhou os passos necessários para realizar ataques `rapid reset` e detetá-los utilizando os *scripts* de cliente e servidor fornecidos. Estes testes são fundamentais para validar a eficácia das metodologias de deteção e mitigação implementadas no controlador Ryu. As simulações realizadas permitiram avaliar a robustez do sistema contra diferentes tipos de ataques, confirmando que o controlador Ryu é capaz de identificar e mitigar ataques complexos de forma eficiente. Cada comando hping3 foi projetado para testar uma parte específica do sistema, garantindo a deteção e a mitigação eficazes das ameaças cibernéticas.

Capítulo 8

Análise dos tempos de execução dos mecanismos de defesa

8.1 Introdução

Neste capítulo, analisamos os tempos de execução de vários mecanismos de defesa implementados no nosso sistema de segurança. Os dados foram recolhidos a partir do ficheiro `analise.json`, que regista o tempo de execução de cada tipo de mecanismo de defesa quando ocorrem eventos específicos na rede. Compreender os tempos de execução é crucial para avaliar o desempenho e a eficiência das nossas estratégias de defesa.

8.2 Funcionamento

Os tempos de execução são medidos em segundos e são registados sempre que um mecanismo de defesa é ativado. Estes tempos fornecem uma visão sobre a sobrecarga computacional associada a cada ação de defesa, ajudando-nos a identificar possíveis congestionamentos e a otimizar o nosso sistema para um melhor desempenho.

8.3 Resultados

O ficheiro `analise.json` contém os diversos tempos de execução para vários mecanismos de defesa como podemos visualizar na figura 8.1:


```
[
    {
        "DDOS": 0.0028471946716308594
    },
    {
        "RST-PORTS_Execution_Time": 0.22593235969543457
    },
```

```
    {
        "HTTP2_RR_Execution_Time_Error": 0.00025773048400878906
    },
    {
        "BlackList_Execution_Time": 0.2556948661804199
    },
    {
        "RST-BADREP_Execution_Time": 0.4888155460357666
    },
    {
        "RST-ACK_Execution_Time": 0.0011324882507324219
    },
    {
        "RST-COUNT-HONEYPOT_Execution_Time": 0.0006833076477050781
    },
    {
        "RST-COUNT-15_Execution_Time": 0.00010585784912109375
    }
]
```


Listing 8.1: Tempos de mitigação

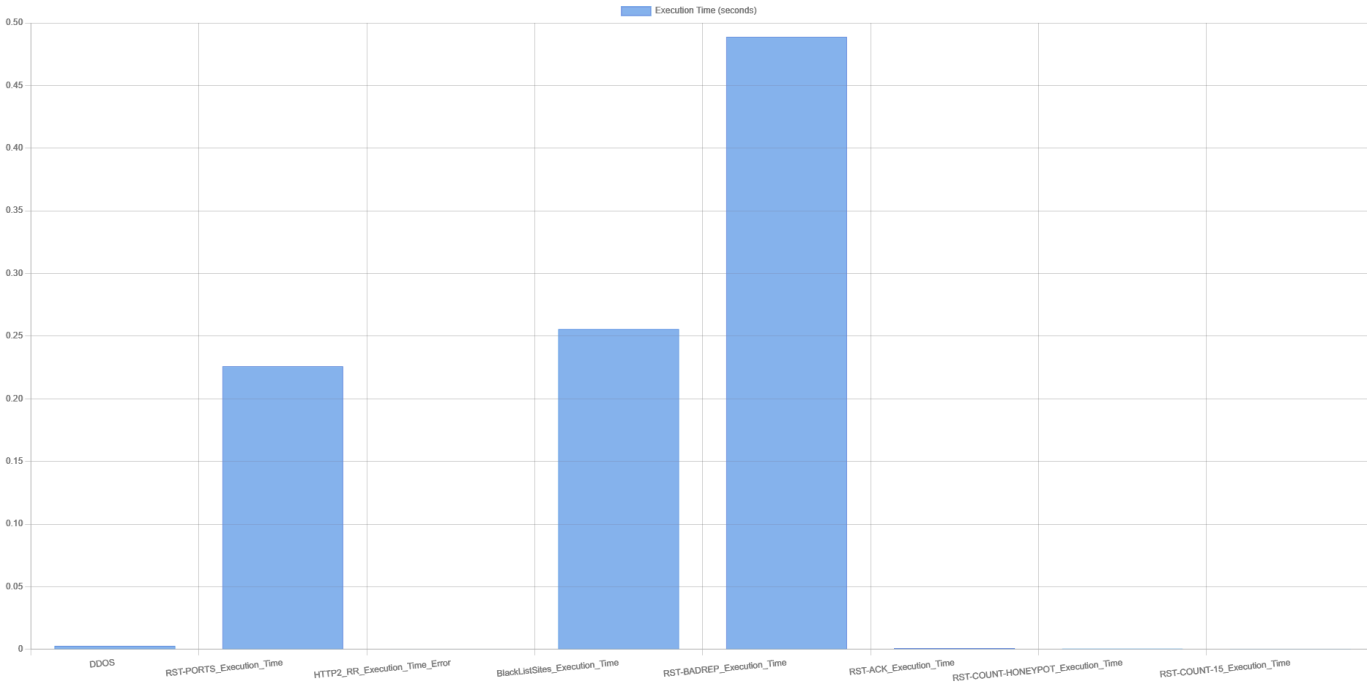


Figura 8.1: Tempos de mitigação de ataques

Para medir os tempos de cada estratégia de mitigação, utilizamos a função `time.time()` do Python para capturar o tempo que demorou a mitigação. Os dados registados foram armazenados em arquivos JSON, usando a função append_to_json_file para análise posterior.

8.4 Análise

8.4.1 Deteção de DDoS (`DDOS`)

- **Tempo de execução**: 0.00284 segundos
- O mecanismo de deteção de DDoS é altamente eficiente, com um tempo de execução de aproximadamente 2.85 milissegundos. Esta resposta rápida é crucial para mitigar ataques em larga escala que podem sobrecarregar todo nosso sistema.

8.4.2 Deteção de *rapid reset* utilizando diversas Portas

- **Tempo de execução**: 0.225 segundos
- A deteção de ataques de *rapid reset* em várias portas demora aproximadamente 225.93 milissegundos. Este tempo relativamente mais alto indica um processo de deteção mais complexo.

8.4.3 Deteção de *HTTP/2 rapid reset* com Erro

- **Tempo de execução**: 0.000257 segundos
- O mecanismo de deteção de *HTTP/2 rapid reset*, quando encontra erros, executa em aproximadamente 0.26 milissegundos, indicando uma sobrecarga mínima.

8.4.4 Deteção de sites/ips na lista negra

- **Tempo de Execução**: 0.255 segundos
- O tempo de execução para a deteção de tráfego para sites/ips na lista negra é de cerca de 255.69 milissegundos. Este mecanismo envolve a inspeção dos dados dos pacotes, o que se torna mais intensivo em termos de processamento.

8.4.5 Deteção de má reputação

- **Tempo de Execução**: 0.4888155460357666 segundos
- A deteção de *resets* de fontes com má reputação demora cerca de 488.82 milissegundos. Este tempo de execução mais alto resulta de ter que se ligar a sistemas externos para realizar a verificação.

8.4.6 Deteção de *rapid reset* com *ACK*

- **Tempo de Execução**: 0.0011324882507324219 segundos
- A deteção de *rapid reset* com *flags* ACK executa em aproximadamente 1.13 milissegundos, indicando um tempo de resposta rápido.

8.4.7 Deteção de *reset* reencaminhando para o *honeypot*

- **Tempo de Execução**: 0.0006833076477050781 segundos
- Este mecanismo, que redireciona o tráfego para um *honeypot* para uma inspeção mais detalhada, executa em cerca de 0.68 milissegundos, mostrando uma utilização de processamento muito baixa.

8.4.8 Deteção de *resets* por tempo

- **Tempo de execução**: 0.00010585784912109375 segundos
- A deteção de *resets* frequentes num curto espaço de tempo é extremamente rápida, com um tempo de execução de aproximadamente 0.11 milissegundos.

8.5 Discussão

A análise dos tempos de execução revela os seguintes pontos:

- **Eficiência**: A maioria dos mecanismos de defesa é executado muito rapidamente, dentro de poucos milissegundos, garantindo que não exista um impacto significativo no desempenho.
- **Complexidade**: Mecanismos que envolvem verificações mais complexas, como a pontuação de reputação ou a inspeção de múltiplas portas, apresentam tempos de execução mais altos. Isto é esperado devido ao processamento adicional necessário.
- **Potencial de Otimização**: Os tempos de execução relativamente mais altos para a deteção de sites/ips na lista negra e *resets* de má reputação sugerem áreas onde a otimização pode ser benéfica. Técnicas como o *cache* de resultados ou a simplificação das verificações podem reduzir estes tempos.

8.6 Resumo

Os mecanismos de defesa implementados no sistema de segurança de rede apresentam, em geral, tempos de execução baixos, assegurando uma resposta rápida

a potenciais ameaças. No entanto, existem áreas onde a otimização pode melhorar ainda mais o desempenho. A monitorização contínua e a análise destes tempos de execução ajudarão a manter um sistema de defesa eficiente e robusto contra ameaças em evolução.

Este capítulo proporciona uma compreensão detalhada do desempenho dos mecanismos de defesa, destacando seus pontos fortes e áreas para melhoria. Os *insights* obtidos orientarão futuras melhorias para garantir que a rede se mantenha segura e resiliente contra ameaças em constante evolução.

Capítulo 9

Conclusão

Esta dissertação apresenta uma abordagem inovadora e abrangente na área da cibersegurança, focando-se na mitigação de ataques que exploram a vulnerabilidade *HTTP/2 rapid reset* em SDN. A pesquisa incluiu uma análise detalhada dos conceitos fundamentais, bem como uma revisão do estado da arte na deteção de ataques originários de botnets, proporcionando uma base sólida para o desenvolvimento de estratégias eficazes de mitigação.

A interoperabilidade de algoritmos em ambientes SDN mostrou-se essencial, permitindo a aplicação de soluções de segurança adaptáveis e dinâmicas em redes complexas.

No âmbito prático, implementámos e testámos diversos mecanismos de defesa no controlador Ryu, avaliando os seus tempos de execução e a eficácia na deteção e mitigação de ataques. Os resultados mostraram que a maioria dos mecanismos de defesa tem tempos de execução baixos, assegurando uma resposta rápida a potenciais ameaças. Identificaram-se, no entanto, áreas para otimização, particularmente nos mecanismos de deteção de sites na lista negra e na API de reputação, onde a utilização de *cache* de resultados ou a simplificação das verificações poderiam reduzir os tempos de execução.

A implementação de algoritmos que não apenas detetam, mas também respondem proativamente a ameaças, mostrou ser essencial para a criação de sistemas de segurança robustos. Esta abordagem permite não só a identificação de atividades maliciosas, mas também a implementação imediata de medidas defensivas para prevenir danos e garantir a integridade do sistema.

Este trabalho contribui significativamente para o campo da cibersegurança ao explorar a vulnerabilidade *HTTP/2 rapid reset* e propor soluções práticas para a sua mitigação.A análise contínua dos tempos de execução dos mecanismos de defesa são fundamentais para manter um sistema de defesa eficiente e robusto contra ameaças em evolução.

Futuramente, espera-se que este projeto estabeleça novos padrões na segurança de redes, possibilitando o desenvolvimento de ferramentas avançadas de registo e análise de dados em SDN. A capacidade de monitorizar e registar o tráfego de rede em vários pontos, integrada com sistemas de deteção de anomalias, melho-

rará a precisão na identificação de padrões de tráfego de botnets. Além disso, a criação de um IPS específico para mitigar ataques *HTTP/2 rapid reset*, utilizando a linguagem Python e o controlador Ryu, representa um avanço significativo na proteção contra ameaças cibernéticas emergentes.

Em conclusão, esta dissertação não só oferece uma solução robusta para a mitigação de ataques *HTTP/2 rapid reset*, mas também abre caminho para futuras pesquisas e desenvolvimentos na área da cibersegurança, garantindo redes mais seguras.

Referências

Ryu sdn framework. URL https://ryu-sdn.org/. Accessed: (2024-03-17).

Mininet overview - mininet, 2022. URL https://mininet.org/overview/. Accessed: (2024-03-11).

Flask (web framework) - wikipedia, 2024. URL https://en.wikipedia.org/wiki/Flask_(web_framework). Accessed: (2024-05-17).

Cloudflare. Cinco boas práticas para mitigar ataques DDoS. 2022.

Miguel Borges de Freitas, Luis Rosa, Tiago Cruz, and Paulo Simões. SDN-Enabled Virtual Data Diode. In Sokratis K. Katsikas, Frédéric Cuppens, Nora Cuppens, Costas Lambrinoudakis, Annie Antón, Stefanos Gritzalis, John Mylopoulos, and Christos Kalloniatis, editors, *Computer Security*, pages 102–118, Cham, 2019. Springer International Publishing. ISBN 978-3-030-12786-2. doi: 10.1007/978-3-030-12786-2_7.

Julien Desgats and Lucas Pardue. HTTP/2 rapid reset: deconstructing the record-breaking attack, 2023. URL https://blog.cloudflare.com/technical-breakdown-http2-rapid-reset-ddos-attack/. Accessed: (2024-01-30).

Zbigniew Dziong, Jean-Charles Gregoire, Jacek Rak, IEEE Communications Society, Institute of Electrical, and Electronics Engineers. *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks) : conference proceedings : September 26-28, 2016, Montreal, Canada*. IEEE, 2016. ISBN 9781467389914.

Gatefy. Botnet: o que é, como funciona e como prevenir - gatefy, 2021. URL https://gatefy.com/pt-br/blog/o-que-sao-botnets/. Accessed: (2023-09-15).

Stefanini Group. O que é SDN? entenda o conceito de software defined networkk - stefanini brasil, 2018. URL https://stefanini.com/pt-br/insights/artigos/entenda-o-conceito-de-software-defined-network. Accessed: (2024-02-15).

Fei Hu, Qi Hao, and Ke Bao. A survey on software-defined network and openflow: From concept to implementation, 4 2014. ISSN 1553877X.

Rodrigo Jantsch. Universidade federal do rio grande do sul instituto de informÁtica curso de ciência da computaçÃo, 2016.

kaspersky. O que é botnet?, 2018. URL `https://www.kaspersky.com.br/resource-center/threats/botnet-attacks`.

Diego Kreutz, Fernando M V Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, Senior Member, Steve Uhlig, D Kreutz, and F Ramos. Software-defined networking: A comprehensive survey, 2014.

Jomilė Nakutavičiūtė, 2020. URL `https://nordvpn.com/pt-br/blog/what-is-quic-protocol/`. Accessed: 2023-12-10.

Institute of Electrical, Electronics Engineers, Yimeng Zhao, Luigi Iannone, and Michel Riguidel. *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on : date, 18-21 Nov. 2015*. IEEE, 2015. ISBN 9781467368841.

Gonçalo Pereira, José Silva, and Pedro Sousa. Estudo comparativo de controladores software-defined networking (sdn) comparative study of software-defined networking (sdn) traffic controllers, 2019.

R. Sanjeetha, Monica R. Mundada, and G. S. Vaibhavi. Botnet forensic analysis in software defined networks using ensemble based classifier. pages 462–467. Institute of Electrical and Electronics Engineers Inc., 2022. ISBN 9798350397475. doi: 10.1109/I4C57141.2022.10057656.

Renato Yoshio Soma. Introdução a SDN e OpenFlow, 2019.

I. Sumantra and S. Indira Gandhi. DDoS attack detection and mitigation in software defined networks, 2020.

Algar Telecom. O que é SDN (software defined network) e como funciona, 2018. URL `https://blog.algartelecom.com.br/geral/o-que-e-sdn-software-defined-network-e-como-funciona/`. Accessed: (2023-11-02).

Ngoc Thinh Tran, Tan Long Le, and Minh Anh Tuan Tran. ODL-ANTIFLOOD: a comprehensive solution for securing opendaylight controller. pages 14–21. Institute of Electrical and Electronics Engineers Inc., 12 2018. ISBN 9781538691861. doi: 10.1109/ACOMP.2018.00011.

Ryhan Uddin and Md Fahad Monir. Evaluation of four SDN controllers with firewall modules. Association for Computing Machinery, 1 2020. ISBN 9781450377782. doi: 10.1145/3377049.3377050.

M A Vidya, 2023. URL `https://www.anoopcnair.com/cve-2023-44487-http-2-rapid-reset-attack/`. Accessed: 2023-12-09.

Changhoon Yoon, Seungwon Shin, Phillip Porras, Vinod Yegneswaran, Heedo Kang, Martin Fong, Brian O'Connor, and Thomas Vachuska. A security-mode for carrier-grade SDN controllers. volume Part F132521, pages 461–473. Association for Computing Machinery, 12 2017. ISBN 9781450353458. doi: 10.1145/3134600.3134603.