Automação e Cibersegurança: Criando uma Ferramenta de Testes para Redes SDN

Ryan M. S. Leal[1], Johan K. E. Freitas[1], Francisco A. C. Albuquerque Júnior[1], Waslon T. A. Lopes[2], Fabrício B. S. Carvalho[2] e Iguatemi E. Fonseca[1]

[1]Centro de Informática – Universidade Federal da Paraíba (UFPB)
João Pessoa – PB – Brasil

[2]Centro de Energias Alternativas e Renováveis – Universidade Federal da Paraíba (UFPB)
João Pessoa, PB, Brasil.
ryanleal@cc.ci.ufpb.br, {johankevin33, facajx}@gmail.com, {waslon, fabricio}@cear.ufpb.br,iguatemi@ci.ufpb.br

Abstract. *With the widespread adoption of Software Defined Networks (SDN) in various current applications, increasingly more methods of vulnerability exploitation have been created targeting the data, control, and application planes. Due to its characteristic of centralizing control logic, SDN networks are more susceptible to denial-of-service attacks, thereby heightening the need to improve network protection methods. Aiming to contribute to the understanding of cyberattacks in SDN networks, this work demonstrates the development of an automated and modular testing execution tool, allowing for the customization of tests and ease of use in SDN network management routines.*

Resumo. *Com a ampla adoção das redes SDN (SDN - Software Defined Networks) em diversas aplicações atuais, mais métodos de exploração de vulnerabilidades têm sido criados tendo como alvos os planos de dados, controle e aplicação. Devido a sua característica de centralização da lógica de controle, as redes SDN se mostram mais vulneráveis a ataques de negação de serviço, aumentando a necessidade de aperfeiçoamento dos métodos de proteção das redes. Visando contribuir com o aprendizado de cyberataques em redes SDN, o presente trabalho demonstra o desenvolvimento de uma ferramenta de execução de testes automatizada e modularizada, permitindo a personalização dos testes e facilidade nas rotinas de gerenciamento de redes SDN.*

1. Introdução

A arquitetura de redes SDN (SDN – *Software Defined Networks*) tem sido cada vez mais adotada em diversas organizações. Sua natureza centralizada (por meio de um controlador de rede) e sua divisão da rede em planos de dados, controle e aplicação permitem que a manutenção e gerenciamento sejam feitos de forma muito mais eficiente e que os recursos sejam configurados de maneira dinâmica e flexível [Xia et al. 2015]. Além disso, sua natureza permite que sejam parte das rotinas de CI/CD (CI - *Continuous Integration*, CD - *Continuous Deployment*) e facilitem os esquemas de orquestração, permitindo a criação e manutenção de serviços automatizados e personalizados, por meio de sua programabilidade [Georgiev and Nikolova 2023]. Porém, com novas características, surgem novos pontos de vulnerabilidade, que podem ser explorados por atacantes. Os ataques podem gerar danos consideráveis à rede, causando prejuízos financeiros e, a depender da

aplicação, físicos aos usuários, principalmente sabendo da utilização dessa arquitetura em redes de quinta geração (5G) [Yao et al. 2019], as quais fornecem serviços simultâneos para diversas aplicações de diferentes tipos.

Uma das maneiras mais eficientes para melhorar a segurança da rede baseia-se em rotinas de testes de segurança, que levam os administradores da rede a conhecer seus pontos fortes e fracos, bem como reconhecer as medidas necessárias para mitigar em caso de ataques. Porém, caso executados por pessoas com pouco conhecimento na área de segurança, podem levar a erros de interpretação de resultados e, consequentemente, à má configuração da rede, sendo um dos maiores riscos de segurança, segundo dados da OWASP [Foundation 2023]. Além disso, é importante salientar o tempo gasto para executar cada cenário de teste, sendo necessário configurar cada ataque e analisar individualmente seus resultados, enfatizando a maior probabilidade de erros humanos durante a análise. Portanto, a criação de uma ferramenta de automação de testes pode facilitar a rotina dos administradores e ainda gerar mais segurança em seus resultados.

Ao pesquisar por artigos voltados para área de automação e segurança de redes SDN, a maioria dos trabalhos se refere a revisões da área [Yungaicela-Naula et al. 2022] ou visam estabelecer maneiras de identificar ataques [Yuan et al. 2024] [Smith-perrone and Sims 2017] na rede por meios automáticos, o objetivo deste trabalho é apresentar uma ferramenta de testes de segurança automatizada e modularizada, permitindo que os administradores da rede possam testá-la de maneira personalizada, fazendo com que ataques sejam adicionados e retirados com um simples processo de configuração, gerando logs de saída, possibilitando a análise dos resultados obtidos. Desta forma, facilita-se o processo de auditoria, monitoramento, manutenção e gerenciamento da rede, onde os administradores podem executar os testes e assim obter dados relevantes quanto a robustez da rede contra os ataques que foram escolhidos, isso de forma automática e integrada com os ciclos de CI/CD da organização. O restante deste artigo está organizado da seguinte maneira. A Seção 2 mostra uma breve descrição de redes SDN, bem como discute algumas de suas vulnerabilidades. A ferramenta de automatização de testes está apresentada na Seção 3, já a Seção 4 apresenta os resultados do experimentos. A Seção 5 mostra os comentários finais e trabalhos futuros.

2. Redes SDN

2.1. Visão Geral

As redes SDN trouxeram novas possibilidades para os ambientes de desenvolvimento, por meio da sua facilidade de manutenção e gerenciamento [Nsafoa-Yeboah et al. 2022]. Tais redes permitem que os recursos sejam alocados dinamicamente, o que é de grande utilidade para as aplicações modernas que trabalham com ambientes flexíveis, adequando conforme forem as necessidades, o que não era simples em redes tradicionais. A centralização do controle da rede, por meio de controladores programáveis, tornou as rotinas de monitoramento e gerenciamento muito mais práticas e automatizadas, possibilitando a personalização das regras de controle da rede e a criação de aplicações que usam informações da rede por meio das API's (API - *Application Programming Interface*) dos controladores [Guo et al. 2020].

As redes SDN são divididas em três planos: Plano de Dados (em que estão os dispositivos físicos da rede, como os switches e roteadores, responsáveis por encaminhar

os dados); Plano de Controle (no qual está localizado o controlador e estão sendo executadas as lógicas de tomada de decisão e configuração da rede); e Plano de Aplicação [Gopi et al. 2017] (no qual o usuário tem acesso aos serviços da rede - podendo ela ser programada para uso personalizado, utilizando informações obtidas por meio de API e enviando ações que geram as modificações necessárias; ou seja, é a camada em que os esquemas de gerenciamento e orquestração atuam).

Como meio de comunicação entre as camadas, são utilizados tecnologias como Openflow e P4 (P4 – *Programming Protocol-Independent Packet Processor*). O primeiro é um protocolo de comunicação e é um dos mais utilizados devido à sua praticidade, sendo já implementado em diversas ferramentas voltadas para SDN. Por sua vez, o P4 é uma linguagem de programação que independe de protocolos, o que tem se tornado mais atrativo, gerando mais uma camada de abstração e permitindo maior personalização [Liatifis et al. 2023]. Para fornecer mais velocidade nas operações de consulta à tabela de fluxos, as redes SDN utilizam em sua maioria a memória TCAM (TCAM - *Ternary Content Addressable Memory*), que são mais rápidas que a RAM (RAM - *Random Access Memory*) e CAM (CAM - *Content Addressable Memory*), sendo capaz de consultar utilizando três tipos distintos de entrada: “0”, “1”e “X”(representando uma informação de valor arbitrário), gerando melhorias nas consultas. Porém, como consequência é uma memória mais cara e com maiores custos energéticos, sendo utilizada de maneira limitada, pois gera maiores gastos na produção e manutenção do hardware da rede [Wen et al. 2016].

2.2. Vulnerabilidades em Redes SDN

A centralização da lógica de controle da rede e a divisão da rede em diferentes planos, promove os benefícios já mencionados, mas também traz novas ameaças [Sokappadu et al. 2019], [Nsafoa-Yeboah et al. 2022]. Ao afetar o controlador da rede, todas as redes que são controladas por ele também são afetadas. Ou seja, a utilização de um controle central cria oportunidades para explorar ataques de DoS (DoS – *Denial of Service*), os quais visam o esgotamento de recursos e a negação de serviço da rede, podendo ter como alvos os planos de dados, controle e aplicação [Qiu and Tang 2022], além de ainda poderem ser distribuídos para gerar o DDoS (DDoS – *Distributed Denial of Service*). Ataques DDoS podem ser de alta taxa ou baixa taxa, a depender da quantidade de tráfego gerado para afetar o dispositivo alvo. Os de alta taxa tendem a ser mais facilmente detectados devido aos picos de aumento do tráfego da rede, fazendo com que defesas sejam ativadas e mitiguem o ataque. Por sua vez, no caso dos ataques de baixa taxa, devido ao tráfego ser semelhante ao de um usuário comum, é mais difícil a detecção, visto que o atacante mantêm a conexão o suficiente para evitar o tempo limite [Rios et al. 2022].

Uma das vulnerabilidades exploradas pelos atacantes é a quantidade de memória TCAM limitada, que é utilizada em ataques como forma de gerar negações de serviço para usuário legítimos. Um exemplo desse tipo de ataque é o *Slow-TCAM* [Pascoal et al. 2017], um DDoS de baixa taxa, que usa uma *botnet* - ou seja, uma rede de dispositivos controlados pelo atacante para manter as conexões abertas até que a tabela de fluxo esteja cheia e gere negações de serviço para usuários legítimos. Isso pode ser mais agravado, evoluindo para o *Slow-Saturation* [Pascoal et al. 2020], caso o usuário crie dois subgrupos de *bots*, ambos gerando tráfego de baixa taxa constantemente e preenchendo a tabela de fluxos. Porém, um dos subgrupos em certos intervalos de tempo vai gerar um tráfego de alta taxa, sobrecarregando o *switch*, que começa a enviar pacotes in-

teiros em vez de somente o cabeçalho, gerando sobrecarga também na comunicação entre o controlador e os *switches*. Isto leva também ao esgotamento dos recursos do controlador, consequentemente gerando instabilidades e negação de serviço em todas as redes controladas por ele.

3. Desenvolvimento da Ferramenta

Para trazer uma solução para a problemática apresentada, a criação de uma ferramenta automatizada tornou-se atrativa, visto que dada uma certa configuração de cenário de testes de segurança como entrada, bastava executar a ferramenta e verificar seus *logs*, podendo ainda utilizá-los em algum sistema de gerenciamento de logs para obter uma saída mais interativa e agradável para o usuário, servindo assim de base para a tomada de decisão na manutenção da rede. A ferramenta pode inclusive ser adicionada aos ciclos de CI/CD, permitindo automatização e controle do ambiente de forma mais prática, participando até dos esquemas de orquestração. A depender das saídas, pode ser modificada a distribuição dos recursos da rede para previnir possíveis tentativas de ataques.

Outro ponto importante para a ferramenta é a capacidade de modularização dos ataques, em que é possível acoplar e desacoplar os módulos conforme sejam as necessidades dos testes, sendo então uma entrada para o arquivo de configuração quais ataques executar e quais seus alvos. Para fazer o desenvolvimento da ferramenta, optou-se pelo uso da linguaguem de programação Python 3.x e as bibliotecas *importlib* para realizar as importações de módulos dinamicamente. Com isso, pode-se importar os *scripts* dos ataques, *logging* para gerar os registros da execução dos testes por meio de mensagens de informação, debug e erro. Isso serve para gerar a saída da ferramenta, *subprocess* para executar comandos no sistema operacional, podendo assim executar ferramentas instaladas no sistema e permitindo também que as saídas do terminal sejam gravadas para processar e verificar resultados. Também utilizou-se a *scapy* que permite que pacotes de redes personalizados sejam gerados, útil inclusive para criar novos scripts de ataques.

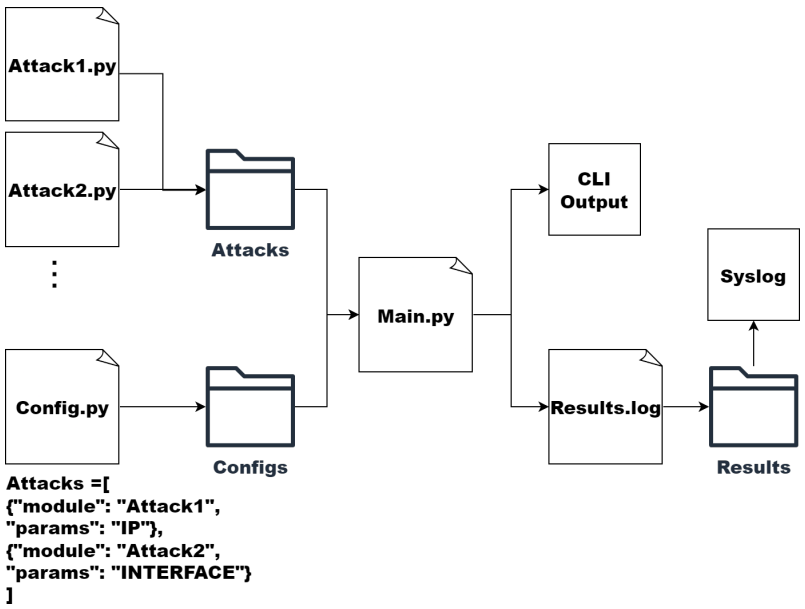

Figura 1. Arquitetura da ferramenta de automação de testes de segurança em redes SDN.

Na Figura 1 é apresentada a arquitetura de funcionamento da ferramenta de automação criada. Pode-se observar o passo-a-passo de execução, em que inicialmente o usuário configura a entrada no arquivo *config.py*, permitindo que ele selecione os parâmetros dos testes, como os ataques a serem executados (e têm seus scripts inseridos na pasta *Attacks*) e seus parâmetros, além da definição dos valores dos parâmetros utilizados. A ferramenta é então executada pelo arquivo principal *Main.py*, que irá realizar o processo de análise das configurações e execução dos ataques selecionados, possuindo um sistema de tratamento de erro em caso de configuração feita de forma errada. Tendo executado os ataques, a ferramenta retorna por meio da CLI (*Command Line Interface*), mostrando o passo-a-passo da execução em tempo real e também salvando as informações no arquivo *Result.log*, que pode ser utilizado por um sistema de *Syslog*, o que permite que a equipe de gerenciamento possa ter aplicações personalizadas para tratar a saída e gerar visualização, facilitando o processo de manutenção.

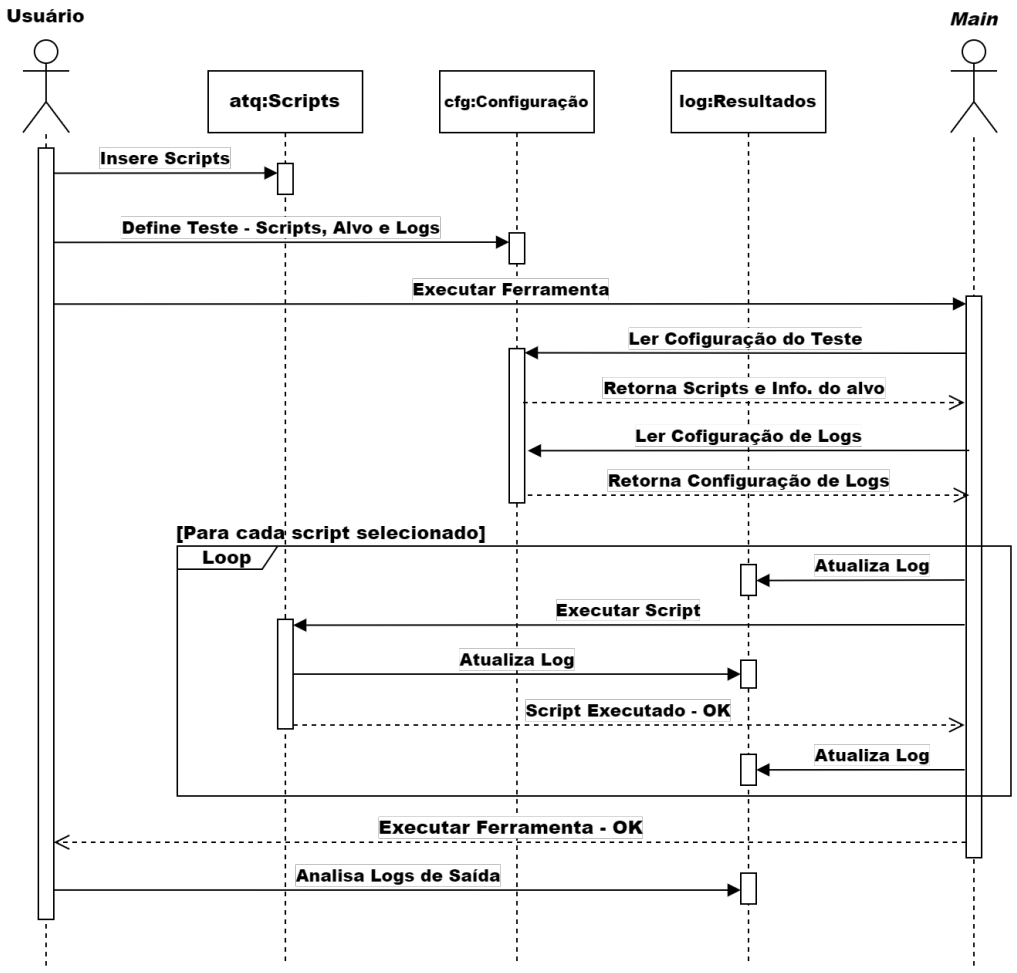

Figura 2. Diagrama de Sequência da ferramenta de automação de testes.

Na Figura 2 é demonstando o funcionamento da ferramenta por meio do diagrama de sequência, na qual o usuário inicialmente interage com as configurações base da ferramenta e só volta a interagir no final da execução do teste para observar os resultados obtidos. Todo o processo de execução dos scripts é feito pela ferramenta, atualizando o log sempre que um novo script é executado. Quando é finalizada sua execução, por essa

visualização é possível enxergar como os processos básicos da ferramenta foram executados. Devido a questões de privacidade da pesquisa, não será possível apresentar o código utilizado para os testes, mas por meio dos diagramas e explicações desta seção, é possível identificar, de maneira geral, o funcionamento e como se deram os testes.

4. Experimentos e Resultados

Concluído o processo de desenvolvimento base da ferramenta, foram realizados experimentos, executando testes de funcionalidades e análises de ganho de tempo com relação ao processo de execução manual do cenário de testes de segurança. Assim, nesta seção é feito o detalhamento desse processo.

4.1. Experimentos realizados

Para realizar o experimento, primeiramente foi criado um ambiente local de testes simulando o funcionamento de uma rede SDN real, para realizar esse processo, foram utilizadas duas máquinas virtuais, uma com o Xubuntu 22.04, a qual continha o controlador ONOS 3.0.0 [Foundation 2024], um dos principais controladores do mercado, sendo utilizado em diversas aplicações reais e também foi utilizada uma máquina Kali Linux 2023.4, a qual executou o emulador de redes SDN, Mininet [Contributors 2022], sendo utilizado o Openflow como protocolo de comunicação entre as camadas, foi feita a emulação de uma topologia de rede em árvore de profundidade 4 e que foi conectada com o ONOS para ser o controlador da rede, na Figura 3 é possível ver a topologia já sendo reconhecida pelo controlador ONOS em sua interface gráfica.

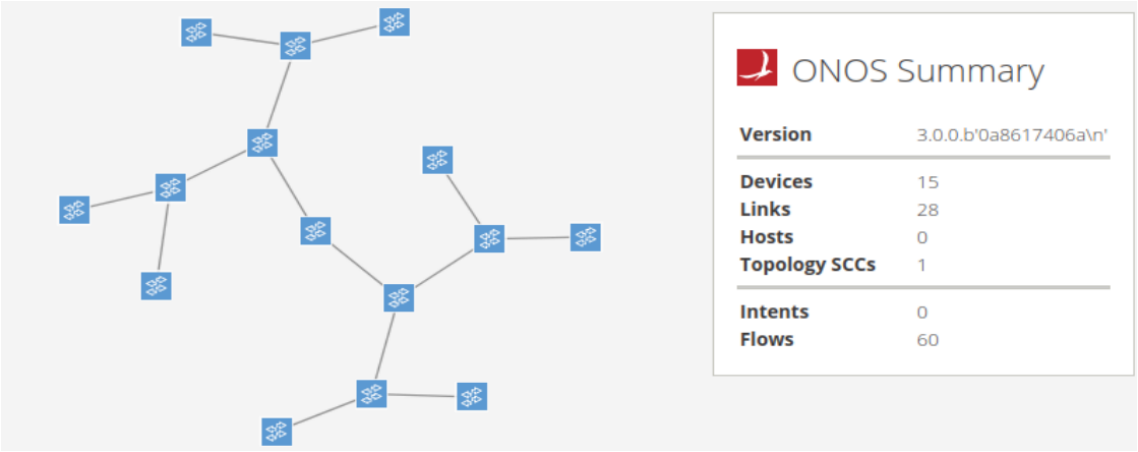

Figura 3. Topologia da rede emulada no Mininet apresentada na interface gráfica do ONOS.

Os seguintes ataques foram executados nos experimentos [Zhang et al. 2018]:

1. *MAC Address Flooding*: foi utilizada a ferramenta macof e ela executa uma inundação de pacotes ARP com endereços MAC (MAC – *Media Access Control*) aleatórios, o que além de congestionar a rede, também pode levar a tabela de endereços MAC do switch a ficar cheia e gerar comportamentos inesperados;
2. *TCP SYN Flood*: É um ataque de inundação que utiliza o pacote de início de conexão SYN do protocolo TCP (TCP – *Transmission Control Protocol*) para ocupar as portas do switch e gerar travamentos e negação de serviço, nesse trabalho, foi

utilizada tanto uma versão implementada utilizando Scapy quanto a versão disponível no Hping3, uma ferramenta que gera pacotes TCP/UDP/ICMP personalizados e permite também realizar inundações;

3. *UDP Flood*: Ataque de inundação que utiliza pacotes UDP (UDP – *User Datagram Protocol*) para sobrecarregar o switch, cada pacote UDP recebido, é normalmente respondido com um ping, caso sobrecarregado, será necessário que diversos pacotes de respostas sejam enviados, porém, maioria dos ataques modifica o IP de origem para gerar confusão na máquina alvo do ataque;
4. *ICMP Flood*: De maneira semelhante ao UDP flood, esse ataque também visa sobrecarregar o switch e gerar a negação de serviço, para isso, são enviadas múltiplas requisições ICMP (ICMP – *Internet Control Message Protocol*) do tipo *echo request*, as quais são respondidas com *echo response*, porém, com a sobrecarga no alvo, torna-se um desafio para responder uma quantidade imensa de requisições, gerando travamentos e a negação de serviço.

Foram modelados dois cenários de testes, tendo como alvo um switch conectado a um host arbitrário da rede emulada, o primeiro cenário executou 2 scripts de ataques, sendo eles o *macof* e o *Hping3*, ambos com 30 s de tempo limite. Para o segundo cenário, foram adicionados mais 3 scripts de ataques, criados utilizando a biblioteca *Scapy*, para gerar uma inundação de pacotes TCP, UDP e ICMP com 10 s de tempo limite para cada um, com isso, foi possível observar a diferença de tempo gerada conforme mais ataques são incluídos na ferramenta. A variação do ganho de tempo obtido ao realizar ataques mais demorados e mais rápidos, além de testar a funcionalidade da ferramenta para scripts que utilizam ferramentas de terceiros e também para os implementados manualmente pelo usuário utilizando bibliotecas como o *Scapy*, os arquivos de configuração do primeiro e segundo cenário são encontrados, respectivamente, nas Figuras 4 e 5. Cada um dos dois cenários foi executado com a ferramenta automatizada e também manualmente pelo terminal, executando os comandos com configuração padrão de uso de cada ferramenta e *script*, usando somente um tempo limite de 30s para delimitar a execução.

```
ATTACK_SCRIPTS = [
    {'module': 'macof_attack', 'params': {'interface': 'INTERFACE'}},
    {'module': 'hping_attack', 'params': {'target': 'IP', 'interface': 'INTERFACE'}},
    # Adicione mais ataques conforme necessário
]

TARGET_INFO = {
    'IP': '10.0.0.5',
    'INTERFACE': 'h5-eth0'
}
```

Figura 4. Arquivo de configuração utilizado no primeiro cenário de testes, com 2 scripts de ataque.

Para analisar o tempo de execução, foi utilizada a biblioteca *time* do Python, realizando os cálculos de tempo de execução individual (cada ataque) e geral dos scripts (todos os ataques juntos), sendo também cronometrado o tempo de execução manual para gerar o comparativo de tempo gasto em cada cenário. No processo de execução dos testes, foi selecionado um dos hosts de um nó folha da rede para simular o dispositivo atacante, a

```
ATTACK_SCRIPTS = [
    {'module': 'syn_flood_attack', 'params': {'target_ip': 'IP', 'target_port': 'PORT', 'duration': 'DURATION'}},
    {'module': 'icmp_flood_attack', 'params': {'target_ip': 'IP', 'duration': 'DURATION'}},
    {'module': 'udp_flood_attack', 'params': {'target_ip': 'IP', 'target_port': 'PORT', 'duration': 'DURATION'}},
    {'module': 'hping_attack', 'params': {'target_ip': 'IP', 'interface': 'INTERFACE'}},
    {'module': 'macof_attack', 'params': {'interface': 'INTERFACE'}}
]

TARGET_INFO = {
    'IP': '10.0.0.16',
    'PORT': '80',
    'DURATION': '10',
    'INTERFACE': 'h16-eth0'
}
```

Figura 5. Arquivo de configuração utilizado no segundo cenário de testes, com 5 scripts.

partir dele foi executada a ferramenta. Na Figura 6 temos uma captura de tela do arquivo de log gerado como saída para um dos cenários de testes, no qual é possível observar que a ferramenta gera informações de logs para diferentes níveis e realiza a análise de tempo de execução.

```
2024-06-18 21:21:22,444 - attack_scripts.udp_flood_attack - INFO - Iniciando ataque UDP Flood no IP 10.0.0.16, porta 80 por 10 segundos
2024-06-18 21:21:32,455 - attack_scripts.udp_flood_attack - INFO - Ataque UDP Flood concluído. 293 pacotes enviados.
2024-06-18 21:21:32,458 - __main__ - INFO - Ataque udp_flood_attack concluído com sucesso em 10.01 segundos
2024-06-18 21:21:32,458 - results.result_logger - INFO - Resultado do udp_flood_attack: Ataque UDP Flood concluído. 293 pacotes enviados.
2024-06-18 21:21:32,458 - __main__ - INFO - Executando ataque: hping_attack com parâmetros: {'target_ip': '10.0.0.16', 'interface': 'h16-eth0'}
2024-06-18 21:21:32,459 - attack_scripts.hping_attack - INFO - Executando Hping no IP 10.0.0.16 através da interface h16-eth0 com timeout de 30 segundos
2024-06-18 21:22:02,523 - attack_scripts.hping_attack - ERROR - Hping expirou após 30 segundos
2024-06-18 21:22:02,523 - __main__ - INFO - Ataque hping_attack concluído com sucesso em 30.06 segundos
```

Figura 6. Trecho do arquivo de log gerado na saída de um dos cenários de teste.

4.2. Resultados obtidos

Na Tabela 1 é possível observar os tempos obtidos tanto nas execuções manuais dos cenários de testes, quanto com o uso da ferramenta automatizada. Observa-se que no primeiro cenário, tivemos uma diferença de 35 s, caracterizando um ganho de 53,8% no tempo de execução. Para o caso com mais scripts, observou-se uma diferença de 1min45s, representando uma diferença de 111,7% no tempo de execução. Esta diferença percentual no ganho com mais ataques é explicada pela execução de mais ataques e, portanto, um cenário de testes mais complexo. Assim, quanto mais ataques são executados com a ferramenta, espera-se que os ganhos sejam maiores.

Nestes testes foram realizados os comparativos de tempo, verificando que a execução de forma automática de um teste de segurança é relevante em cenários em que diversos ataques precisariam ser executados, demonstrando inclusive que quanto mais *scripts* são incluídos, maiores os ganhos de tempo, porém, é claro que novas melhorias devem ser realizadas para obter uma ferramenta com mais impacto no ambiente de desenvolvimento, fornecendo um número base de *scripts* maior e mais facilidade na personalização dos cenários e do uso dos logs. Observando o ponto de inclusão em ambiente de desenvolvimento, a ferramenta pode ser incluída facilmente, devido a sua natureza de execução

e geração de logs simples, podendo incluí-la em diversos cenários, simplesmente por executar o código principal, utilizando os logs de saída em diferentes aplicações.

Tabela 1. Comparativo de tempo de execução manual e com a ferramenta.

	TEMPO	
	2 Scripts	**5 Scripts**
Manual	1min40s	3min19s
Ferramenta	1min05s	1min34s

5. Conclusão

Com base nos resultados obtidos, é possível concluir que os ganhos de tempo com o uso da ferramenta são relevantes e conseguem ser ainda maiores para cenários reais e com muito mais testes a serem realizados. Mesmo em um pequeno cenário, a facilidade de simplesmente executar a ferramenta, torna o processo menos cansativo e com menos chance de erros. Além disso, a organização e armazenamento das saídas por meio de um log torna o processo de análise de resultados mais simples e com a implementação de uma aplicação de visualização para o sistema de logs é possível obter um ótimo recurso para gerenciamento da rede, podendo ainda automatizar os processos de manutenção com base nos resultados obtidos nos teste. Ademais, com a integração com os ciclos de CI/CD, os ganhos de tempo conseguem ser maiores ainda. Para trabalhos futuros, almejam-se melhorias das funcionalidades base da ferramenta, a implementação de novos *scripts* de ataques, testes com integração em um sistema de syslog, podendo testar uma funcionalidade para geração de sugestões com base nos resultados obtidos pelos testes. Além disso, pretende-se utilizar a ferramenta em uma rede 5G OpenRAN, o que pode demonstrar sua utilidade em um cenário presente no mercado e inovador.

Referências

Contributors, M. P. (2022). Mininet - an instant virtual network on your laptop (or other pc). `https://mininet.org/` [Acessado: 05 de Junho de 2024].

Foundation, O. (2023). Top 10 web application security risks. `https://owasp.org/www-project-top-ten/` [Accessado: 10 de Junho de 2024].

Foundation, O. N. (2024). Open network operating system (onos) sdn controller. `https://opennetworking.org/onos/` [Accessado: 15 de Junho de 2024].

Georgiev, S. and Nikolova, K. (2023). Implementation of an agile sdlc ci/cd pipeline for managing a sdn vxlan-evpn fabric. In *2023 31st National Conference with International Participation (TELECOM)*, pages 1–4.

Gopi, D., Cheng, S., and Huck, R. (2017). Comparative analysis of sdn and conventional networks using routing protocols. In *2017 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pages 108–112.

Guo, C., Xie, D., Han, Y., Guo, J., and Wei, Z. (2020). Survey of software-defined network security issues. In Sun, X., Wang, J., and Bertino, E., editors, *Artificial Intelligence and Security*, pages 503–514, Singapore. Springer Singapore.

Liatifis, A., Sarigiannidis, P., Argyriou, V., and Lagkas, T. (2023). Advancing sdn from openflow to p4: A survey. *ACM Comput. Surv.*, 55(9).

Nsafoa-Yeboah, K., Tchao, E. T., Yeboah-Akowuah, B., Kommey, B., Agbemenu, A. S., Keelson, E., and Monirujjaman Khan, M. (2022). Software-defined networks for optical networks using flexible orchestration: Advances, challenges, and opportunities. *Journal of Computer Networks and Communications*, 2022(1):5037702.

Pascoal, T. A., Dantas, Y. G., Fonseca, I. E., and Nigam, V. (2017). Slow TCAM exhaustion DDoS attack. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 17–31. Springer.

Pascoal, T. A., Fonseca, I. E., and Nigam, V. (2020). Slow denial-of-service attacks on software defined networks. *Comput. Networks*, 173:107223.

Qiu, X. and Tang, Z. (2022). Research advanced in the security defence of software defined network. In *2022 International Conference on Electronics and Devices, Computational Science (ICEDCS)*, pages 380–384.

Rios, V. D. M., Inácio, P. R. M., Magoni, D., and Freire, M. M. (2022). Detection and mitigation of low-rate denial-of-service attacks: A survey. *IEEE Access*, 10:76648–76668.

Smith-perrone, J. and Sims, J. (2017). Securing cloud, sdn and large data network environments from emerging ddos attacks. In *2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence*, pages 466–469.

Sokappadu, B., Hardin, A., Mungur, A., and Armoogum, S. (2019). Software defined networks: Issues and challenges. In *2019 Conference on Next Generation Computing Applications (NextComp)*, pages 1–5.

Wen, X., Yang, B., Chen, Y., Li, L. E., Bu, K., Zheng, P., Yang, Y., and Hu, C. (2016). Ruletris: Minimizing rule update latency for tcam-based sdn switches. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 179–188.

Xia, W., Wen, Y., Foh, C. H., Niyato, D., and Xie, H. (2015). A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1):27–51.

Yao, J., Han, Z., Sohail, M., and Wang, L. (2019). A robust security architecture for sdn-based 5g networks. *Future Internet*, 11(4).

Yuan, B., Zhang, C., Ren, J., Chen, Q., Xu, B., Zhang, Q., Li, Z., Zou, D., Zhang, F., and Jin, H. (2024). Toward automated attack discovery in sdn controllers through formal verification. *IEEE Transactions on Network and Service Management*, 21(3):3636–3655.

Yungaicela-Naula, N. M., Vargas-Rosales, C., Pérez-Díaz, J. A., and Zareei, M. (2022). Towards security automation in software defined networks. *Computer Communications*, 183:64–82.

Zhang, M., Bi, J., Bai, J., and Li, G. (2018). Floodshield: Securing the sdn infrastructure against denial-of-service attacks. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 687–698.