LST 2.0: Testbed Emulado Baseado em Contêineres para Redes SDN Seguras

Alexandre M. Kaihara[1], Lucas Bondan[2], João J. C. Gondim[1], Gabriel S. Rodrigues[1], Marcelo A. Marotta[1], Genaína N. Rodrigues[1]

[1]Departamento de Ciência da Computação – Universidade de Brasília (UnB)

[2]Rede Nacional de Ensino e Pesquisa (RNP)

{alexandre.kaihara,siqueira.rodrigues}@aluno.unb.br,lucas.bondan@rnp.br

{gondim, marcelo.marotta, genaina}@unb.br

Abstract. *Not all emulated testbeds are suitable for security experimentation. Often security testbeds are restricted to their application context or are based on other technologies which have configurability and security application limitations (e.g., Mininet). While other proposals allow greater configurability, they do not focus on security applications or are not inserted in the context of SDN networks. To address the identified research gap, the Lightweight SDN Testbed (LST 2.0) is proposed. LST 2.0 is a lightweight tool capable of supporting different application contexts both for security and SDN networks programmatically and in real-time through Python. It is possible to monitor the network and collect metrics using Netflow, sFlow, IPFIX, or CICFlowMeter. In addition, pre-built Docker images are available for emulating both benign and malicious network flows.*

Resumo. *Nem todo testbed emulado é adequado para experimentações em segurança. Frequentemente os testbeds em segurança estão restritos ao contexto de aplicação para o qual foram desenvolvidos ou têm por base outras tecnologias que apresentam limitações de configurabilidade e de aplicações de segurança (e.g., Mininet), enquanto outras propostas de testbeds (e.g., DockSDN) permitem maior configurabilidade, mas não possuem foco em aplicações de segurança ou não estão inseridas no contexto de redes SDN. Visando atender à lacuna de pesquisa identificada, é proposta o Lighweight SDN Testbed (LST 2.0) - ferramenta leve capaz de atender a diversos contextos de aplicação tanto de segurança quanto de redes SDN programadas e em tempo real via Python. É possível monitorar a rede e coletar métricas fazendo uso de Netflow, sFlow, IPFIX ou CICFlowMeter. Ainda, imagens Docker pré-construídas são disponibilizadas para a emulação de fluxos de rede tanto benignos quanto maliciosos.*

1. Introdução

As Redes Definidas por Software (do inglês *Software-Defined Networking* — SDN), tornaram-se importantes arquiteturas de rede, permitindo maior flexibilidade na gestão e no monitoramento via separação do plano de controle do plano de dados. Tal característica permitiu o aprimoramento de mecanismos de segurança para a rede, como, por exemplo, no reforço de políticas de rede, na detecção e mitigação de ataques e na

proteção do próprio controlador de rede [D. Kreutz 2015]. Em contrapartida, a arquitetura SDN trouxe outras questões de segurança, tais como possíveis vulnerabilidades incorporadas ao controlador, ataques TCP/IP SYN, UDP *flooding*, *Man in the Middle*, entre outras [K. Raghunath 2018]. Assim, em face da diversidade de cenários de estudo, é possível beneficiar-se do uso de *testbeds* para a criação de ambientes experimentais adequados e na condução de experimentos em segurança que permitam validar as soluções propostas.

Em geral, a aquisição de equipamentos habilitados com SDN é custosa e sujeita às restrições de orçamento. Uma alternativa é recorrer às soluções de baixo custo, como, por exemplo, a emulação dos componentes de rede. Entretanto, nem todo *testbed* emulado é adequado para a condução de experimentos de segurança ou possui o nível de configurabilidade necessária para o estudo, como o Mininet, que compartilha o mesmo espaço de dados e não provê isolamento de recursos por nó [O. Flauzac 2019]. Ao mesmo tempo, *testbeds* virtualizados com máquinas virtuais consomem muitos recursos onerando o sistema hospedeiro, tornando difícil a experimentação com ataques de alto consumo de recurso, como ataques volumétricos (e.g., *Denial of Service* — DOS). Assim, os estudos em segurança exigem requisitos específicos da ferramenta para o correto uso e criação dos experimentos.

Em vários estudos de segurança têm-se o desenvolvimento de *testbeds* próprios. No entanto, muitas vezes, tais *testbeds* estão restritos ao contexto de aplicação ao qual foram desenvolvidos [R. Meena 2020, I. Sumantra 2020, M. Khan 2020], limitando o reuso da ferramenta. Embora outras propostas de *testbeds* permitam maior configurabilidade da ferramenta [E. Petersen 2020, X. Zhang 2020, R. Chadha 2016], estas não possuem foco em aplicações de segurança ou não estão inseridas no contexto de redes SDN, evidenciando a necessidade por *testbeds* que permitam explorar os múltiplos cenários de redes SDN, mas que também atendam às necessidades específicas em segurança.

Diante do exposto, é proposta uma nova versão do LST focado em segurança com base em virtualização leve com contêineres no contexto de redes SDN, denominado *Lightweight SDN Testbed* (LST 2.0) — disponível no repositório do GitHub[1] sob a licença GNU *General Public License*, versão três. O LST 2.0 objetiva a alta configurabilidade, sendo capaz de gerar diversas topologias programaticamente e em tempo real. Diferentemente da primeira versão, tais topologias podem ser compostas por quaisquer números de *switches*, *hosts* e controladores, além mudança na interface de uso de um arquivo JSON para um conjunto de métodos de linguagem de programação. Além disso, a ferramenta permite manipular diferentes configurações de rede e segurança dos nós de rede (*firewall*, *gateway*, entre outros). Visando necessidades específicas para experimentações em segurança, são disponibilizadas imagens pré-construídas para simular fluxos benignos e maliciosos, além de ferramentas para o monitoramento e a coleta de métricas na rede, facilitando o desenvolvimento de estudos de sistemas de detecção de intrusão em rede, entre outros.

O presente estudo está organizado da seguinte forma. Na Seção 2 são apresentados os trabalhos relacionados. Na Seção 3, a arquitetura da solução e principais funcionalidades da ferramenta são detalhadas. Em seguida, uma demonstração baseada na solução proposta é descrita na Seção 4. Por fim, a Seção 5 traz conclusões e perspectivas de trabalhos futuros.

[1] https://github.com/alexandrekaihara/lst2.0

2. Trabalhos Relacionados

O Mininet é uma ferramenta de emulação leve amplamente conhecida, que permite gerar topologias de redes SDN com grande facilidade. Embora o Mininet venha sendo utilizado por vários estudos de segurança [P. Maity 2020, R. Meena 2020], os nós de rede ali criados não possuem o isolamento necessário em relação ao hospedeiro principal por compartilhar o plano de dados de usuários e pelo lançamento de processos compartilhados. Tais características podem gerar resultados incorretos ou mascarar ataques sendo experimentados. Além disso, os nós de rede gerados precisam ser configurados um a um a cada execução, dificultando a criação de experimentos de maior complexidade [M. Ring 2017].

A utilização da tecnologia de contêineres tem sido explorada em *testbeds* [E. Petersen 2020, R. Chadha 2016, M. Khan 2020] devido à individualização do sistema de arquivos, das interfaces de rede e árvore de processos separada para cada contêiner, além de apresentarem consumo de CPU e memória mais eficientes em relação às máquinas virtuais. O NestedNed explora a construção de um *testbed* voltado para redes SDN hierárquicas, no qual é possível gerar topologias de rede com contêineres Docker aninhados conectados via *switches* Open vSwitch. No entanto, esse *testbed* possui limitações quanto à aplicabilidade em outros domínios, uma vez que os nós de rede são contêineres aninhados voltados para redes SDN hierárquicas.

O *testbed* CyberVAN [R. Chadha 2016] é proposto com foco em segurança, permitindo gerar diferentes cenários experimentais com nós tanto físicos quanto emulados (*e.g.*, contêineres) através de uma interface de usuário gráfica ou comandos no console. Entretanto, a fidelidade da reprodução de topologias é limitada devido ao uso de softwares de simulação de topologias de rede (*e.g.*, ns3, QualNet, entre outros) quando comparado às topologias emuladas. Além disso, o CyberVAN não provê suporte a geração de topologias programaticamente, não sendo possível beneficiar-se dos recursos e da automação proporcionada por linguagens de programação. Além disso, a configurabilidade da ferramenta está restrita às limitações das interfaces disponibilizadas, que podem não atender aos requisitos dos diversos estudos em segurança [X. Zhang 2020, R. Meena 2020, M. Ring 2017].

Soluções com múltiplos controladores têm sido propostas a fim de se contornar as limitações de escalabilidade e de segurança que a centralização do plano de controle de redes SDN trouxeram. Nesse sentido, visando ampliar a aplicabilidade, é essencial que *testbeds* permitam a criação de topologias com múltiplos controladores. O DockSDN [E. Petersen 2020] é um *testbed* com base em contêineres com suporte a geração de topologias personalizadas com múltiplos controladores. No entanto, apesar de toda configurabilidade que o DockSDN provê, não são disponibilizados *scripts* para a geração de ataques em rede, um recurso importante para permitir que pesquisadores foquem mais nas propostas do estudo e menos na elaboração de cenários experimentais.

O DCLab é um testbed com base no Mininet que possui uma API para a criação de experimentos, permitindo gerar topologias programaticamente. Voltado para o contexto de segurança, tanto [I. Sumantra 2020] quanto [R. Meena 2020] também são propostos como *testbeds* baseados no Mininet, montando vários cenários para avaliar a detecção e mitigação de ataques DDoS e técnicas de validação de endereço de origem, respectivamente. No entanto, o DCLab não possui foco em aplicações em segurança, ou seja, não provê scripts de ataque em rede e os estudos de [I. Sumantra 2020, R. Meena 2020] desenvolvem testbeds restritos ao contexto de aplicação para o qual foram desenvolvidos.

No melhor dos conhecimentos dos autores sobre *testbeds* emulados, faltam *testbeds* com alta configurabilidade para redes SDN e segurança capazes de gerar topologias programaticamente em tempo real, com suporte a múltiplos controladores. LST 2.0 trabalha disponibilizando imagens Docker pré-construídas que simulam fluxos de uma pequena organização e capaz de simular ataques como *Denial of Service* (DoS), *Port Scanning* e *Brute Force*. E ainda, a solução proposta monitora e coleta métricas da rede por meio do Netflow, sFlow e IPFIX e gera um relatório de métricas dos fluxos por meio do CICFlowMeter [H. Lashkari 2017], auxiliando na geração de *datasets* em segurança, no treinamento de algoritmos de detecção, no monitoramento de ataques em tempo-real, entre outros. Um resumo de todas as características explanadas, bem como o posicionamento do LST 2.0 frente à literatura é demonstrado na Tabela 1.

	Segurança	SDN	Multi Controlador	Topologia Confiurável	Gerar Topologia Programaticamente	Configuração em tempo-real	Geração de Relatório	Scripts de Ataques
Mininet		✓	✓	✓	✓			
CyberVAN	✓	✓	✓	✓		✓		✓
[M. Khan 2020]	✓			✓				✓
NestedNet		✓		✓		✓		
[M. Ring 2017]	✓	✓					✓	✓
DockSDN		✓	✓	✓	✓	✓		
DCLab		✓	✓	✓	✓	✓		
[I. Sumantra 2020]	✓	✓	✓	✓	✓			✓
[R. Meena 2020]	✓	✓	✓	✓	✓			✓
LST 2.0	✓	✓	✓	✓	✓	✓	✓	✓

Tabela 1. Características dos *Testbeds* Emulados em Relação às Características do LST 2.0

3. *Lightweight SDN Testbed 2.0*

O LST 2.0 foi desenvolvido com o objetivo de atender aos mais diversos cenários de estudos em redes SDN e segurança, via uma interface definida por métodos de linguagem de programação de fácil uso, mas que permitem alta flexibilidade na configuração e geração de topologias personalizadas. Nas seções seguintes têm-se a apresentação da arquitetura da ferramenta e suas principais funcionalidades.

3.1. Arquitetura da Ferramenta

O LST 2.0 é baseado principalmente em contêineres para a geração dos nós de rede e as configurações de comportamento e de rede são feitas programaticamente. A ferramenta está organizada por meio de um conjunto de métodos bem definidos pertencente a uma hierarquia de classes. A classe pai, nomeada como Node, possui todos os atributos e métodos mínimos para se instanciar, deletar e configurar um contêiner qualquer. Dessa forma, os usuários podem desenvolver classes próprias herdando os atributos e métodos da classe Node e focando-se apenas no desenvolvimento das configurações específicas para o estudo.

Por padrão, estão implementadas três classes que são especializações da classe Node como na Figura 1. A classe Host permite criar contêineres que serão nós comuns que consumirão serviços na rede. A classe Switch permite criar *switches* virtuais, cuja configuração de rede é específica para encaminhar pacotes entre portas de uma única interface de ponte (*bridge*) contida no contêiner e também para se conectar a um controlador em um IP e porta específicos. Caso nenhum controlador seja atribuído, o *switch* realizará apenas as funções básicas de camada de rede e também se nenhum IP for atribuído a ele,

será um *switch* de camada de enlace. A classe Controller permite criar contêineres que conseguem instanciar um ou mais controlares.

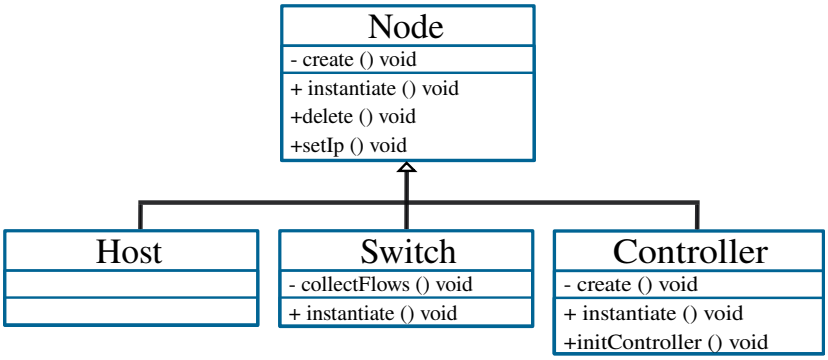

Figura 1. Hierarquia de Classes do LST 2.0

Cada nó de rede é representado por uma instância de alguma das especializações da classe Node. Cada instância, por sua vez, é capaz de criar, destruir e configurar o contêiner a qualquer momento, ficando sob responsabilidade do usuário manter as instâncias das classes e destruir os contêineres para finalizar o experimento.

No presente estudo, o LST 2.0 foi implementado utilizando Docker para o gerenciamento dos contêineres. Ainda, o OpenvSwitch foi utilizado como solução de switches virtuais. Os controladores, por sua vez, são implementados com Ryu[2]. Os controladores Ryu são utilizados por padrão para a classe Controller, os quais são capazes de configurar apenas o roteamento e encaminhamento comum dos *switches* na camada de Rede.

3.2. Principais Funcionalidades

Visando alta configurabilidade da topologia, o LST 2.0 permite gerar quaisquer números de *hosts*, *switches* virtuais e controladores. Os contêineres podem ser instanciados a partir de quaisquer imagens Docker, sejam as disponibilizadas por este estudo, uma imagem local ou uma imagem hospedada no DockerHub, permitindo a criação de novos comportamentos e perfis de rede nos experimentos. Para conectar os nós de rede, são criados pares virtuais Ethernet Linux inserindo cada ponta diretamente nos *namespaces* dos contêineres. Dessa forma, obtém-se maior configurabilidade de rede e evitam-se limitações de uso dos *drivers* de rede do Docker.

É possível instanciar diferentes tipos de controladores, tais como ONOS, Floodlight, POX, entre outros — o que permite explorar diferentes recursos providos pelos controladores (*i.e., northbound* API) ou controladores com melhores desempenhos em determinados cenários de ataques [A. Alashhab 2021]. Tais controladores permitirão avaliar diferentes aspectos da arquitetura SDN, podendo alterar as funções desempenhadas pelos *switches*, introduzir novas políticas na rede e até realizar detecção e mitigação de intrusões em rede. Assim, o LST 2.0 é capaz de avaliar soluções que mitigam a centralização do plano de controle da rede, tornando a arquitetura robusta a ataques do tipo DoS, que podem impactar no desempenho ou até bloquear totalmente o uso da rede.

São disponibilizadas imagens Docker para gerar topologias, entre as quais, permitem instanciar *Hosts* com o Sistema Operacional Ubuntu (versão 20:04), *switches* virtuais

[2]Controlador Ryu: https://ryu-sdn.org/

com o OpenvSwitch e controladores com o controlador Ryu. As demais imagens têm por base o estudo de [M. Ring 2017], que simulam fluxos de uma pequena organização. Tais imagens podem ser do tipo Servidor ou do tipo Cliente. Os contêineres do tipo Cliente consomem os serviços na rede, sendo possível atribuir comportamentos específicos para atender às especificidades de cada experimento.

Voltado para o contexto de segurança, os contêineres do tipo Cliente podem realizar ataques na rede como *Port Scan*, *Brute Force* e DoS, representando máquinas infectadas, sendo possível adicionar novos *scripts* de ataques. Os Clientes geram *logs* das execuções dos seus respectivos scripts, permitindo mapear quando uma ação ou ataque foi introduzida na rede.

O LST 2.0 permite que os *switches* Open vSwitch coletem métricas por meio do Netflow, sFlow ou IPFIX. Dessa forma, é possível fazer o monitoramento e coleta de dados da rede de forma local ou remota via *softwares* de monitoramento. Além disso, ao final da execução do experimento é possível gerar um relatório a partir do CICFlowMeter[3] contendo diversas métricas e estatísticas dos fluxos gerados como a duração, número de pacotes, total de *Bytes* transmitidos, comprimento máximo e mínimo dos pacotes, entre outros.

O LST 2.0 se difere de outros emuladores de rede por ser capaz configurar e instanciar os nós de rede programaticamente em tempo real — o que agiliza e torna mais dinâmica a verificação de erros e a validação das configurações durante o próprio desenvolvimento do experimento. Além disso, o LST 2.0 é capaz de gerar diversos perfis de serviços de rede, clientes e de ataques, permitindo o monitoramento e a coleta de dados da rede, trazendo uma ferramenta altamente configurável para os mais diversos cenários experimentais em SDN e segurança, seja isoladamente ou em ambas simultaneamente.

4. Demonstração

Foi utilizada uma imagem do Ubuntu Server versão 20.04.2 LTS para a demonstração planejada da ferramenta. A imagem foi instalada em uma máquina virtual (VM) do VirtualBox versão 6.1.26 e configurada com 15 GB de RAM, quatro núcleos de CPU e 32 GB de espaço de memória. Ainda que uma VM tenha sido utilizada, a mesma serve apenas para emular a máquina hospedeira do *testbed* com controle da quantidade de recursos dedicados. Ou seja, poderia ter sido utilizada uma máquina real em vez da VM para executar o LST 2.0.

A demonstração da ferramenta teve por base o experimento do estudo de [M. Ring 2017], o qual simula os fluxos de uma pequena organização e de máquinas espalhadas pela rede. Ao executar o experimento, é criada uma topologia composta por quatro sub-redes (Figura 2). A sub-rede dos servidores é composta por contêineres provedoras de serviços de *email*, *web*, de arquivos e *backup*. As demais sub-redes são compostas por uma impressora e contêineres do tipo Cliente. Os Clientes possuem comportamentos específicos da sub-rede ao qual pertencem. Parte desses clientes representam máquinas infectadas e geram fluxos maliciosos na rede. A topologia externa é composta por um servidor Seafile (i.e., repositório de arquivos compartilhados), um servidor *web* e várias máquinas do tipo Cliente, que geram tanto fluxos benignos quanto maliciosos na rede.

[3]https://www.unb.ca/cic/research/applications.html

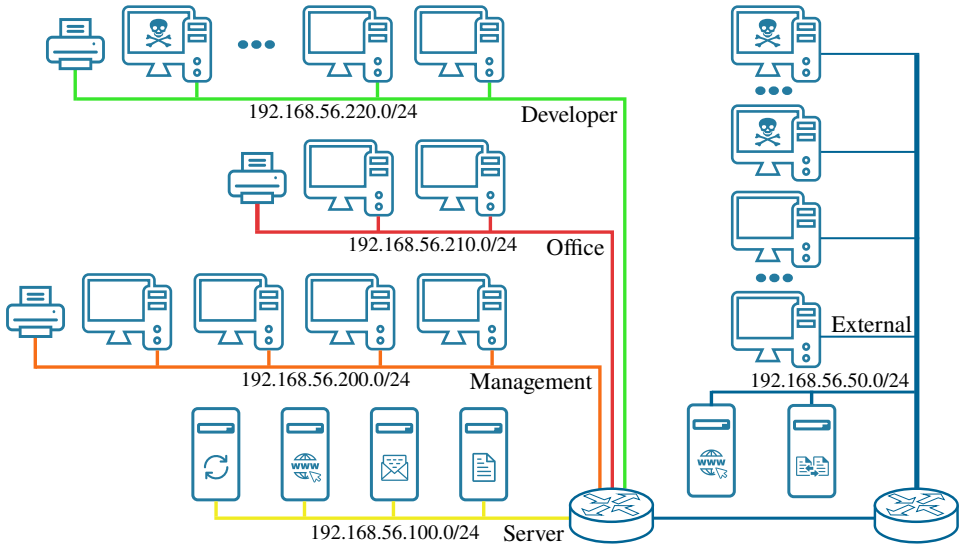

Figura 2. Topologia do Experimento Padrão

	Protocol	Duration	Total Fwd Packets	Total Bytes	Max Fwd Pkt Lenght	Min Fwd Pkt Length	Bytes/s	Packets/s	Outras Métricas
Flow 1	TCP	0,481	8	665	137	0	1382,53	16,63	···
Flow 2	TCP	0,074	4	102	39	0	1378,37	54,05	···
Flow 3	TCP	1,140	8	519	142	0	455,26	7,01	···
Flow 4	TCP	1,073	25	11.177	1282	0	10.416,58	23,29	···
Flow 5	UDP	0,362	2	155	47	47	428,17	5,52	···
Flow 6	UDP	0,419	3	156	28	28	372,31	7,15	···
Outros Fluxos	···	···	···	···	···	···	···	···	···

Tabela 2. Relatório de Execução do Experimento

Todos os Clientes instanciados geram *logs* das atividades na rede, detalhando o tipo de ação, quando foi realizada, e o status que indica se foi possível realizar ou não a ação. Ao fim da execução do experimento, a ferramenta criará um relatório com diversas métricas a partir dos fluxos gerados na rede, como demonstrado na Tabela 2.

5. Conclusão e Trabalhos futuros

No presente estudo propôs-se uma ferramenta que propicia alta configurabilidade para atender uma grande gama de aplicações em redes SDN voltadas para segurança denominada Lightweight SDN Testbed (LST 2.0). A partir da arquitetura proposta, a ferramenta foi desenvolvida com o uso de Docker e Python para a geração dos nós e a configuração dos experimentos, respectivamente. Com uma hierarquia de classes simples, é fácil a criação de novos nós com os mais diversos comportamentos e a capacidade de gerar topologias programaticamente, com alto grau de automação e configurabilidade.

Como trabalhos futuros, espera-se aprimorar a facilidade de utilização da ferramenta via interface gráfica de usuário, de modo que seja possível executar os experimentos localmente ou em um servidor remoto. Também pretende-se melhorar a coleta de métricas da rede e otimizar o uso de memória das máquinas das imagens pré-construídas que geram fluxos em rede.

Agradecimentos

O presente estudo se deu com o apoio financeiro do projeto PROFISSA (Programmable Future Internet for Secure Software Architectures), sob a bolsa n. 2020/05152-7, e da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

Referências

A. Alashhab, e. a. (2021). Experimenting and evaluating the impact of dos attacks on different sdn controllers. In *IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA*. IEEE.

D. Kreutz, e. a. (2015). Software-defined networking: A comprehensive survey. In *Proceedings of the IEEE*, volume 103. IEEE.

E. Petersen, M. A. (2020). Docksdn: A hybrid container-based sdn emulation tool. In *2020 IEEE Latin-American Conference on Communications (LATINCOM)*. IEEE.

H. Lashkari, e. a. (2017). Characterization of tor traffic using time based features. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - ICISSP*.

I. Sumantra, S. G. (2020). Ddos attack detection and mitigation in software defined networks. In *International Conference on System, Computation, Automation and Networking (ICSCAN)*. IEEE.

K. Raghunath, P. K. (2018). Towards a secure sdn architecture. In *9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE.

M. Khan, O. Rehman, I. R. (2020). Lightweight testbed for cybersecurity experiments in scada-based systems. In *020 International Conference on Computing and Information Technology 10 Sep. 2020*, volume 1. IEEE, Tabuk, Saudi Arabia.

M. Ring, e. a. (2017). Flowbased benchmark datasets for intrusion detection. In *Proceedings of the 16 th European Conference on Cyber Warfare and Security (ECCWS)*, page 361369. ACPI.

O. Flauzac, E. R. (2019). Is mininet the right solution for an sdn testbed? In *IEEE Global Communications Conference (GLOBECOM)*. IEEE.

P. Maity, e. a. (2020). An effective probabilistic technique for ddos detection in openflow controller. In *IEEE Systems Journal*, volume 16. IEEE.

R. Chadha, e. a. (2016). Cybervan: A cyber security virtual assured network testbed. In *MILCOM 2016 - 2016 IEEE Military Communications Conference*. IEEE.

R. Meena, e. a. (2020). Ryu sdn controller testbed for performance testing of source address validation techniques. In *3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE)*. IEEE.

X. Zhang, N. Prabhu, R. T. (2020). Nestednet: A container-based prototyping tool for hierarchical software defined networks. In *International Workshop on Rapid System Prototyping (RSP)*. IEEE.