Towards Enhancing the Endpoint Security using Moving Target Defense (Shuffle-based Approach) in Software Defined Networking

Muhammad Faraz Hyder
Department of Software Engineering
NED University of Engineering & Technology
Karachi, Pakistan
farazh@neduet.edu.pk

Waseemullah
Department of Computer Science and IT
NED University of Engineering & Technology
Karachi, Pakistan
waseemu@neduet.edu.pk

Muhammad Umer Farooq
Department of Computer Science and IT
NED University of Engineering & Technology
Karachi, Pakistan
umer@neduet.edu.pk

Usama Ahmed
Department of Computer Science and IT
NED University of Engineering & Technology
Karachi, Pakistan
ahmed.pg13900@cloud.neduet.edu.pk

Wajahat Raza
Department of Computer Science and IT
NED University of Engineering & Technology
Karachi, Pakistan
raza.pg13722@cloud.neduet.edu.pk

Abstract-Static IP addresses make the network vulnerable to different attacks and once the machines are compromised, any sensitive information within the network can be spoofed. Moving Target Defense (MTD) provides an efficient mechanism for proactive security by constantly changing different system attributes. Software Defined Networks (SDNs) provide greater flexibility in designing security solutions due to their centralized management and programming capabilities. In this paper, a mechanism for the protection of endpoint security is developed using IP address host shuffling. In the proposed approach, the real IP address of the host is masked and a virtual IP address is assigned. The virtual IPs are mined from the pool of unassigned IP addresses. The address pool is created using a pseudo-random number generator to guarantee high randomness. This approach helps in invalidating the intelligence gathered by the adversaries through the changes in the network configuration that will disturb attack execution, eventually leading to attack failure. Transparency is attained via preserving the actual IP intact and mapping a virtual IP to it. The proposed solution is implemented using the RYU Controller and Mininet. The efficient results obtained from the experiments substantiate the effectiveness of the MTD approach for enhancing endpoint security.

Keywords-IP shuffling; endpoint security; moving target defense; software defined networking; virtual IP

I. INTRODUCTION

Static networks are an easy target for attackers, even from a remote network. The attackers send probes using various scanning tools to random hosts in the network to identify the victims and attack them. However, dynamic allocation can be carried out using protocols like DHCP or NAT. Even though NAT and DHCP provide dynamic IP address assignment schemes for hosts, they cannot provide adequate security and can also be tracked. The Moving Target Defense (MTD) technique has been adopted to different domains including network security, cyber-physical systems security, Software Defined Network (SDN) security, etc. [1, 2]. However, to the best of our knowledge, MTD for endpoint security has not been covered in depth so far. Endpoint security is gaining importance, especially with the advent of technologies like the Internet of Things (IoT) [3].

In this paper, we exploited the RYU SDN controller [4] to design an MTD framework that can be used to provide dynamic IP assignments to end hosts periodically. Moreover, the solution ensures that all the communication inside the network is based upon virtual IP addresses assigned to different hosts. This provides a way to counter different threats generated against the endpoints. The solutions are scalable and programable due to the SDN architecture [5]. This approach of hiding the actual IPs of the endpoint/hosts substantially

Corresponding author: Muhammad Faraz Hyder enhances their security as the attackers are unable to get their correct information.

The purpose of MTD is to increase confusion and uncertainty for the attacker who is attempting to breach a system by exploiting vulnerabilities [6]. MTD changes the system/network configurations thus the information gathered by the attacker becomes invalid as the target attack surface is changing continuously. MTD techniques are categorized in terms of redundancy, shuffling, and diversity [7]. Shuffling-based MTD changes the system's configuration such as IP shuffling or service reconfiguration. MTD is mostly combined with SDN by using SDN controllers as a central controlling point for network topology shuffling-based MTD, or random host mutation using OpenFlow and port hopping. Diversity-based MTD gives the ability to implement various executions of same services of functionalities. This provides diversity in programming language to avoid code injection attacks and software stack diversity to enhance service provisions and network resilience. Redundancy-based MTD enhances the reliability of the system by generating many images of network components. The authors in [8, 9] used MTD for network security, especially in SDN networks. MTD has also been adopted for cloud security [10, 11]. The authors in [12] exploited MTD for the security of cyber-physical systems. There is a recent trend of the adaptation of MTD for wireless ad hoc networks as well [13, 14].

The main contributions of this paper are:

- The development of shuffle-based MTD mechanism in a distributed SDN environment.
- The MTD mechanism for the enhancement of endpoint security in the SDN environment.

II. THE PROPOSED SCHEME

This section will cover the way we have implemented and deployed the SDN based shuffling MTD. We have set up a MTD mechanism using RYU as a Python controller for the SDN which consists of 3 switches interconnected with 8 user hosts. Communication between Host one and Host two is being shown in Figure 1. The local networking of the system is:

1. In the first step, the packet will be coming into switch one with the real source address and a virtual destination address.
2. A packet will then go to the RYU controller.
3. The RYU controller will check for the attachments of the hosts in the IP pool if the switch one is directly connected to the destination address of the host.
4. The RYU controller will receive a negative response in a message.
5. The packet will go out of the RYU controller and will ask for a virtual IP address since it was detected with a real IP address.
6. The packet now will come to switch two with the virtual source address and virtual destination address.
7. The packet will then go to the RYU controller.

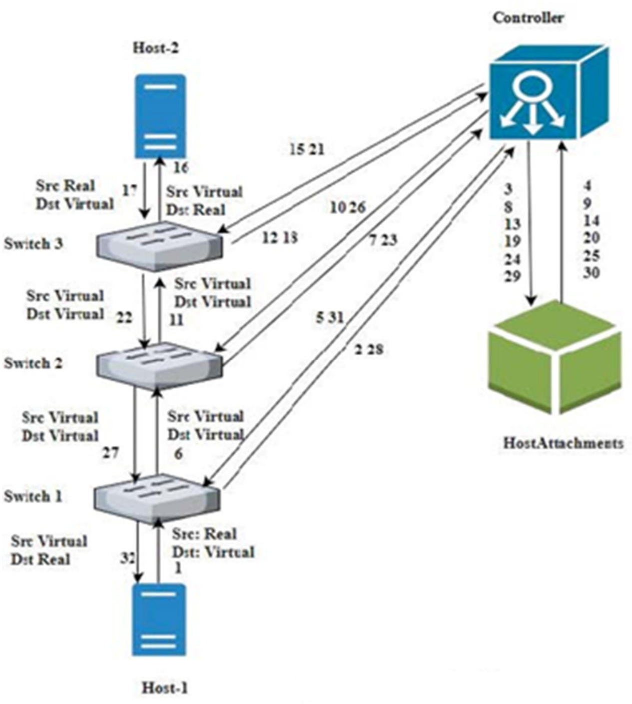


Fig. 1. Traffic flow sequence.

8. The RYU controller will now check for the attachments of the hosts in the IP pool if the switch two is associated directly to the destination address of the host.
9. The RYU controller will receive a negative response in a message.
10. A packet going out of the switch will now message to forward it to next available hopping.
11. The packet will come to switch three with the virtual source address and virtual destination address.
12. The packet will then go to the RYU controller.
13. The RYU controller will check for the attachments of the hosts in the IP pool if switch three is directly connected to the destination address of the host.
14. The RYU controller will receive a positive response in a message.
15. The packet going out of switch will now change to the real destination address.
16. The packet goes out to the destination host, i.e. host two.
17. Host two will now respond with the real source address and the virtual destination address.
18. The packet will go to the RYU controller.
19. The RYU controller will now check for the attachments of the hosts in the IP Pool if switch three is attached directly to the destination address of the host.
20. The RYU controller will receive a negative response in a message.

21. The packet going out of switch will now receive message to change into virtual source address.
22. The packet will come to switch two with the virtual source address and virtual destination address.
23. The packet will then go to the RYU controller.
24. The RYU controller will now check for the attachments of the hosts in the IP pool if switch two is directly connected to the destination address of the host.
25. The RYU controller will receive a negative response in a message.
26. A packet going out of the switch will now message to forward it to next available hopping.
27. The packet will come to switch one with the virtual source address and the virtual destination address.
28. The packet will then go to the RYU controller.
29. The RYU controller will now check for the attachments of the hosts in IP pool if switch one is attached directly to the destination address of the host.
30. The RYU controller will receive a positive response in a message.
31. A packet going out of switch will now message to change into the real destination address.
32. The packet mentioned in the first step has now reached host one, i.e. its destination.

In Figure 1, we have shown the working flow of the IP shuffling from real to virtual IP address. This workflow shows the architecture of the way our SDN environment containing MTD is implementing random host mutation using a timeout event. The timeout event is triggered every 30 seconds and then shuffles the IP address for every host. Figure 2 represents the real-virtual IP address life cycle. Virtual addresses are being shuffled in a random manner and are updated in every host. This system in general transforms real addresses to virtual addresses and vice versa. The real address is allocated the same way as before and nothing is changed on it, but all the communication is being done through virtual addresses inside the SDN network. At every timeout event triggering, the old flows are also deleted to be seemingly synchronized with packets and flow mappings.

III. EXPERIMENTAL SETUP

A. SDN Controller

We have used RYU as our SDN controller which is a Python-based controller and it will be responsible for controlling real and virtual IPs between switches and hosts using the random host mutation technique. It will also be controlling the communication between each host via virtual IPs and MAC addresses. It is a controller through which we execute and control both network configuration and real-time control of the network. Figure 3 indicates the experimental topology.

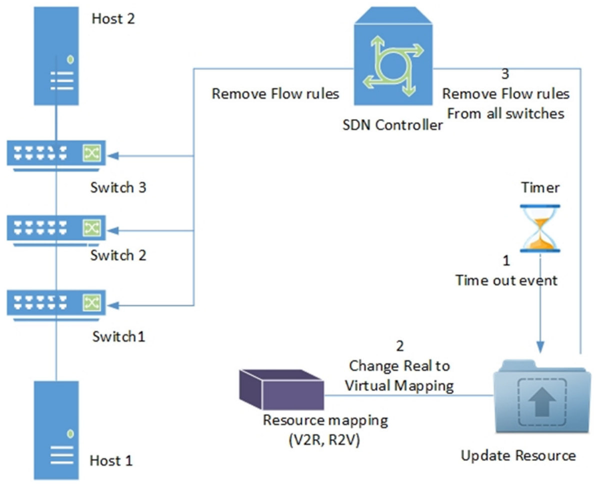


Fig. 2. Virtual-real IP address life cycle.

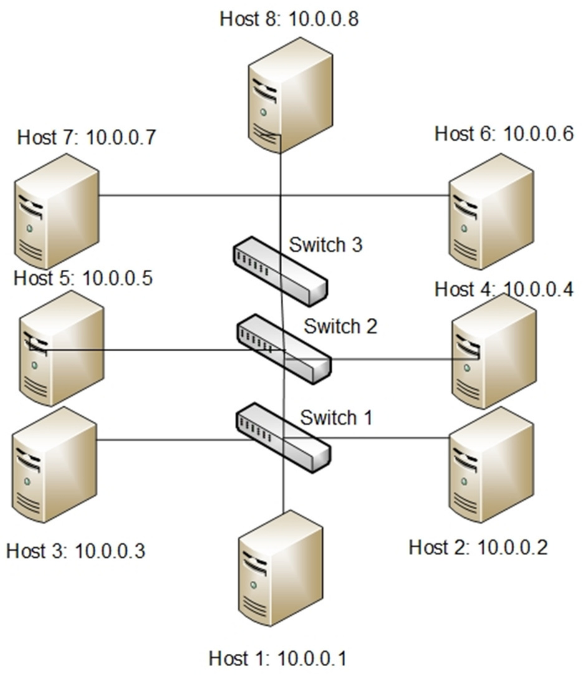


Fig. 3. Experimental topology.

B. Network Simulation

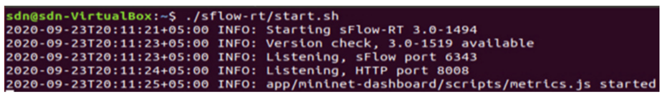
We have used Mininet [15] as our network simulation tool to deploy our network topology containing 3 switches and 8 hosts containing class A IP addresses. It is a network simulator that can generate virtual switches, hosts and attach these to the SDN controllers, in our case with the RYU controller. We emulated this via Python-based programming scripts where modifications are required at a certain level to grip the project with the required topology that we created.

C. Network Monitoring

We have used sFlow [16] as our network monitoring tool which is used to monitor our network devices which include switches and hosts. It will also be used for monitoring any kind of traffic and attacks on the network. We will also use sFlow for generating attack graphs. It is a monitoring tool through which we see the graphical interface of a topology and different shuffling of IPs and hosts in our environment. Also, we can identify the main and virtual IPs from its interface. Shuffling of IP addresses and MAC addresses and blocked source IPs can be identified.

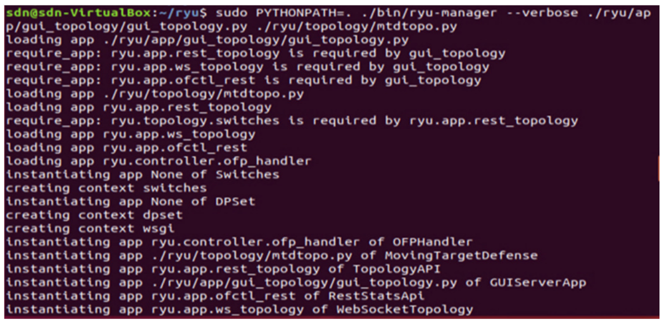
IV. RESULTS AND DISCUSSION

First, the sflow-rt is started as shown in Figure 4. Afterwards, the RYU controller starts along with our proposed algorithm and GUI topology as depicted in Figure 5. The network is created as presented in Figure 6. Now, the hosts try to ping each other through their real IP. Since our algorithm is activated, the packets drop on the real IP addresses as indicated in Figure 7. The controller is blocking this communication as presented in Figure 8. After finding the virtual IP address of the remote machine, host 2 is able to ping the host 8 successfully as indicated in Figure 9. The controller's log confirms this connectivity as depicted in Figure 10.



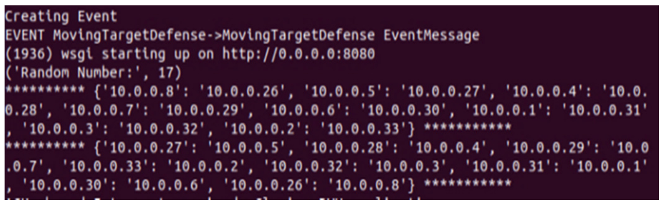
```
sdn@sdn-VirtualBox:~$ ./sflow-rt/start.sh
2020-09-23T20:11:21+05:00 INFO: Starting sFlow-RT 3.0-1494
2020-09-23T20:11:23+05:00 INFO: Version check, 3.0-1519 available
2020-09-23T20:11:23+05:00 INFO: Listening, sFlow port 6343
2020-09-23T20:11:24+05:00 INFO: Listening, HTTP port 8008
2020-09-23T20:11:25+05:00 INFO: app/mininet-dashboard/scripts/metrics.js started
```

Fig. 4. Initiation of the sFlow network monitoring tool.



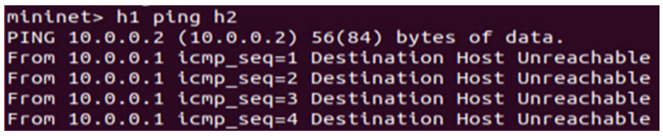
```
sdn@sdn-VirtualBox:~/ryu$ sudo PYTHONPATH=. ./bin/ryu-manager --verbose ./ryu/ap
p/gui_topology/gui_topology.py ./ryu/topology/mtdtopo.py
loading app ./ryu/app/gui_topology/gui_topology.py
require_app: ryu.app.rest_topology is required by gui_topology
require_app: ryu.app.ws_topology is required by gui_topology
require_app: ryu.app.ofctl_rest is required by gui_topology
loading app ./ryu/topology/mtdtopo.py
loading app ryu.app.rest_topology
require_app: ryu.topology.switches is required by ryu.app.rest_topology
loading app ryu.app.ws_topology
loading app ryu.app.ofctl_rest
loading app ryu.controller.ofp_handler
instantiating app None of Switches
creating context switches
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ./ryu/topology/mtdtopo.py of MovingTargetDefense
instantiating app ryu.app.rest_topology of TopologyAPI
instantiating app ./ryu/app/gui_topology/gui_topology.py of GUIServerApp
instantiating app ryu.app.ofctl_rest of RestStatsApi
instantiating app ryu.app.ws_topology of WebSocketTopology
```

Fig. 5. Initiation of RYU controller with GUI topology and our MTD algorithm.



```
Creating Event
EVENT MovingTargetDefense->MovingTargetDefense EventMessage
(1936) wsgi starting up on http://0.0.0.0:8080
('Random Number:', 17)
********** {'10.0.0.8': '10.0.0.26', '10.0.0.5': '10.0.0.27', '10.0.0.4': '10.0.
0.28', '10.0.0.7': '10.0.0.29', '10.0.0.6': '10.0.0.30', '10.0.0.1': '10.0.0.31'
, '10.0.0.3': '10.0.0.32', '10.0.0.2': '10.0.0.33'} ***********
********** {'10.0.0.27': '10.0.0.5', '10.0.0.28': '10.0.0.4', '10.0.0.29': '10.0
.0.7', '10.0.0.33': '10.0.0.2', '10.0.0.32': '10.0.0.3', '10.0.0.31': '10.0.0.1'
, '10.0.0.30': '10.0.0.6', '10.0.0.26': '10.0.0.8'} ***********
```

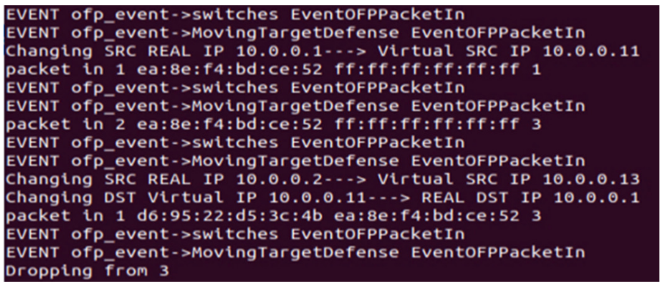
Fig. 6. Custom topology network creation.



```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
```

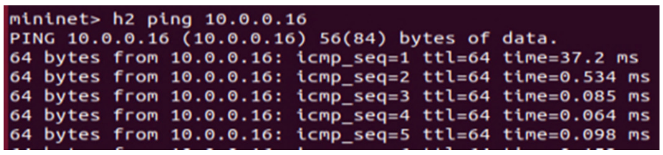
Fig. 7. Host 1 to host 2 ICMP communication through their real IP addresses.

In order to evaluate the effectiveness of our proposed scheme, we launch an attack on the end-hosts on their real and virtual IP address using hping [17].



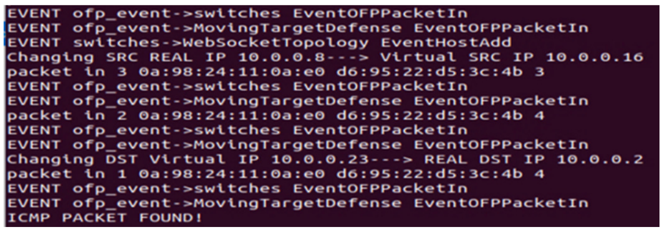
```
EVENT ofp_event->switches EventOFPPacketIn
EVENT ofp_event->MovingTargetDefense EventOFPPacketIn
Changing SRC REAL IP 10.0.0.1---> Virtual SRC IP 10.0.0.11
packet in 1 ea:8e:f4:bd:ce:52 ff:ff:ff:ff:ff:ff 1
EVENT ofp_event->switches EventOFPPacketIn
EVENT ofp_event->MovingTargetDefense EventOFPPacketIn
packet in 2 ea:8e:f4:bd:ce:52 ff:ff:ff:ff:ff:ff 3
EVENT ofp_event->switches EventOFPPacketIn
EVENT ofp_event->MovingTargetDefense EventOFPPacketIn
Changing SRC REAL IP 10.0.0.2---> Virtual SRC IP 10.0.0.13
Changing DST Virtual IP 10.0.0.11---> REAL DST IP 10.0.0.1
packet in 1 d6:95:22:d5:3c:4b ea:8e:f4:bd:ce:52 3
EVENT ofp_event->switches EventOFPPacketIn
EVENT ofp_event->MovingTargetDefense EventOFPPacketIn
Dropping from 3
```

Fig. 8. The controller is blocking the communication after identifying a real IP address.



```
mininet> h2 ping 10.0.0.16
PING 10.0.0.16 (10.0.0.16) 56(84) bytes of data.
64 bytes from 10.0.0.16: icmp_seq=1 ttl=64 time=37.2 ms
64 bytes from 10.0.0.16: icmp_seq=2 ttl=64 time=0.534 ms
64 bytes from 10.0.0.16: icmp_seq=3 ttl=64 time=0.085 ms
64 bytes from 10.0.0.16: icmp_seq=4 ttl=64 time=0.064 ms
64 bytes from 10.0.0.16: icmp_seq=5 ttl=64 time=0.098 ms
```

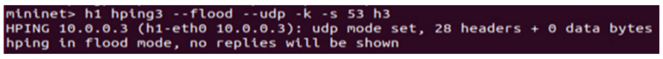
Fig. 9. Now we have first ping host 2 to host 8 through its virtual IP and MAC addresses.



```
EVENT ofp_event->switches EventOFPPacketIn
EVENT ofp_event->MovingTargetDefense EventOFPPacketIn
EVENT switches->WebSocketTopology EventHostAdd
Changing SRC REAL IP 10.0.0.8---> Virtual SRC IP 10.0.0.16
packet in 3 0a:98:24:11:0a:e0 d6:95:22:d5:3c:4b 3
EVENT ofp_event->switches EventOFPPacketIn
EVENT ofp_event->MovingTargetDefense EventOFPPacketIn
packet in 2 0a:98:24:11:0a:e0 d6:95:22:d5:3c:4b 4
EVENT ofp_event->switches EventOFPPacketIn
EVENT ofp_event->MovingTargetDefense EventOFPPacketIn
Changing DST Virtual IP 10.0.0.23---> REAL DST IP 10.0.0.2
packet in 1 0a:98:24:11:0a:e0 d6:95:22:d5:3c:4b 4
EVENT ofp_event->switches EventOFPPacketIn
EVENT ofp_event->MovingTargetDefense EventOFPPacketIn
ICMP PACKET FOUND!
```

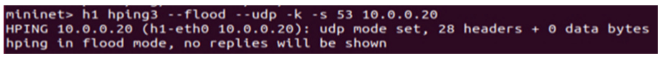
Fig. 10. The controller is allowing the communication after identifying a virtual IP address.

Figure 11 represents the hping based UDP flood from host 1 to host 3. Since it is based upon the real IP address, no traffic will be generated if the attacker is able to somehow find the virtual IP address of the host. Then the attack will be generated against the virtual IP as shown in Figure 12. However, since our algorithm is periodically generating random IPs after 30 seconds, the attack can be generated for a maximum duration of 30 seconds. Figure 13 indicates the flooding traffic passing through different switches.



```
mininet> h1 hping3 --flood --udp -k -s 53 h3
HPING 10.0.0.3 (h1-eth0 10.0.0.3): udp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

Fig. 11. hping flood from real host IP to another real host.



```
mininet> h1 hping3 --flood --udp -k -s 53 10.0.0.20
HPING 10.0.0.20 (h1-eth0 10.0.0.20): udp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

Fig. 12. hping flood from real host IP to the virtual IP of the host.

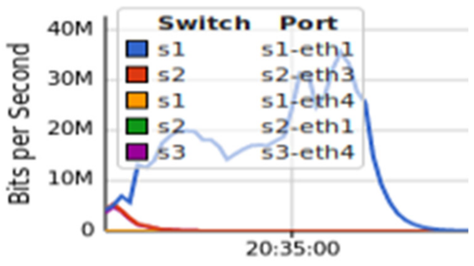


Fig. 13. Attack traffic generation across each switch.

Figure 14 shows that the attack traffic on virtual IP is reduced after 30 seconds as the virtual IP is changed. This approach ensures that DDoS attacks cannot proceed beyond the limit of the IP shuffling period. The solution is adaptable to change in the frequency of IP shuffling. It means that DDoS period can further be reduced with higher frequency of movement.

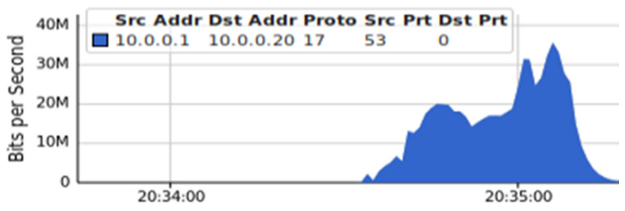


Fig. 14. The attack is generated on said hosts but due to the MTD algorithm, the connection will be terminated in 30 seconds.

Table I shows the comparison of the proposed approach with the existing solutions. The proposed scheme implemented endpoint protection using SDN and MTD approaches. Moreover, the proposed solution does not require any specific agent to be installed on the endpoint. This is a substantial attribute of our proposed solution and makes the proposed scheme flexible enough. The authors in [18] used the commercial tool "Fireeye" for the enhancement of endpoint security. Their solution is based on anti-virus based approach. The authors in [19] proposed a mechanism for endpoint security along with estimation of its effectiveness. The author in [20] proposed a mechanism for endpoint security improvement.

TABLE I. COMPARISON OF THE PROPOSED IMPLEMENTATION WITH EXISTING SOLUTIONS.

	SDN Enabled	MTD Enabled	Separate user-agent on the endpoint
Proposed	✓	✓	Not required
[18]	X	X	Required
[19]	X	X	Required
[20]	X	X	Required

V. CONCLUSION AND FUTURE WORK

In this work, we have implemented MTD by randomly metamorphosing IP addresses of endpoints using SDN. The fundamental aim of MTD is to thwart the discovery of machines by the probes generated by numerous worms and scanning tools. The actual IP addresses are hidden from both internal and external scanners to avoid attacks. The RYU controller was used to mutate the IP addresses. A host's real IP address is assigned with a virtual IP address at a high mutation rate from an unused resource pool of IP addresses by using a pseudo random number generator.

A brief description of the implementation of DNS response interception by the controller follows: When a host requests for the IP address of the destination host, the DNS server responds with the real IP address of the destination. The RYU controller intercepts the packet and alters the actual IP address to the virtual IP address before it reaches the requesting host. The implementation of authorized user functionality is as follows: An authorized user will be able to ping the destination host using a real IP address. The real IP detection within the network can be automated.

In the future, the proposed scheme can be used for the protection of critical services, especially those that are running on LANs including DNS, DHCP, etc. The proposed scheme can be used to divert the DNS and DHCP through real to virtual IP mapping along with port randomization for these services. This approach not only enhances security but can also be used for digital privacy enhancement.

REFERENCES

[1] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang, and S. Kambhampati, "A Survey of Moving Target Defenses for Network Security," *IEEE Communications Surveys Tutorials*, vol. 22, no. 3, pp. 1909–1941, 2020, https://doi.org/10.1109/COMST.2020.2982955.

[2] M. F. Hyder and M. A. Ismail, "INMTD: Intent-based Moving Target Defense Framework using Software Defined Networks," *Engineering, Technology & Applied Science Research*, vol. 10, no. 1, pp. 5142–5147, Feb. 2020, https://doi.org/10.48084/etasr.3266.

[3] S. Tedeschi, C. Emmanouilidis, J. Mehnen, and R. Roy, "A Design Approach to IoT Endpoint Security for Production Machinery Monitoring," *Sensors*, vol. 19, no. 10, Jan. 2019, Art. no. 2355, https://doi.org/10.3390/s19102355.

[4] T. Fujita, "Introduction to Ryu SDN framework," 2013, Accessed: Aug. 01, 2021. [Online]. Available: https://ryu-sdn.org/slides/ONS2013-april-ryu-intro.pdf.

[5] Y. Djeldjeli and M. Zoubir, "CP-SDN: A New Approach for the Control Operation of 5G Mobile Networks to Improve QoS," *Engineering, Technology & Applied Science Research*, vol. 11, no. 2, pp. 6857–6863, Apr. 2021, https://doi.org/10.48084/etasr.4016.

[6] R. Zhuang, S. A. DeLoach, and X. Ou, "Towards a Theory of Moving Target Defense," in *Proceedings of the First ACM Workshop on Moving Target Defense*, New York, NY, USA, Nov. 2014, pp. 31–40, https://doi.org/10.1145/2663474.2663479.

[7] J.-H. Cho *et al.*, "Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense," *arXiv:1909.08092 [cs]*, Sep. 2019, Accessed: Aug. 01, 2021. [Online]. Available: http://arxiv.org/abs/1909.08092.

[8] A. Chowdhary, A. Alshamrani, D. Huang, and H. Liang, "MTD Analysis and evaluation framework in Software Defined Network (MASON)," in *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, New York, NY, USA, Mar. 2018, pp. 43–48, https://doi.org/10.1145/3180465.3180473.

[9] A. Chowdhary, S. Pisharody, and D. Huang, "SDN based Scalable MTD solution in Cloud Network," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, New York, NY, USA, Oct. 2016, pp. 27–36, https://doi.org/10.1145/2995272.2995274.

[10] H. Alavizadeh, J. Jang-Jaccard, and D. S. Kim, "Evaluation for Combination of Shuffle and Diversity on Moving Target Defense Strategy for Cloud Computing," 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)*, Aug. 2018, pp. 573–578, https://doi.org/10.1109/TrustCom/BigDataSE.2018.00087.

[11] M. Torquato and M. Vieira, "Moving target defense in cloud computing: A systematic mapping study," *Computers & Security*, vol. 92, May 2020, Art. no. 101742, https://doi.org/10.1016/j.cose.2020.101742.

[12] B. Potteiger, Z. Zhang, and X. Koutsoukos, "Integrated moving target defense and control reconfiguration for securing Cyber-Physical systems," *Microprocessors and Microsystems*, vol. 73, Mar. 2020, Art. no. 102954, https://doi.org/10.1016/j.micpro.2019.102954.

[13] P. Wang, M. Zhou, and Z. Ding, "A Two-Layer IP Hopping-Based Moving Target Defense Approach to Enhancing the Security of Mobile

Ad-Hoc Networks," *Sensors*, vol. 21, no. 7, Jan. 2021, Art. no. 2355, https://doi.org/10.3390/s21072355.

[14] E. M. Ghourab and M. Azab, "Benign false-data injection as a moving-target defense to secure mobile wireless communications," *Ad Hoc Networks*, vol. 102, May 2020, Art. no. 102064, https://doi.org/10.1016/j.adhoc.2019.102064.

[15] R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using Mininet for emulation and prototyping Software-Defined Networks," in *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, Bogota, Colombia, Jun. 2014, pp. 1–6, https://doi.org/10.1109/ColComCon.2014.6860404.

[16] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Computer Networks*, vol. 62, pp. 122–136, Apr. 2014, https://doi.org/10.1016/j.bjp.2013.10.014.

[17] L. Liang, K. Zheng, Q. Sheng, and X. Huang, "A Denial of Service Attack Method for an IoT System," in *2016 8th International Conference on Information Technology in Medicine and Education (ITME)*, Fuzhou, China, Dec. 2016, pp. 360–364, https://doi.org/10.1109/ITME.2016.0087.

[18] M. Dujmić, D. Delija, G. Sirovatka, and M. Žagar, "Using FireEye Endpoint Security for educational purposes," in *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, Opatija, Croatia, Sep. 2020, pp. 1206–1211, https://doi.org/10.23919/MIPRO48935.2020.9245414.

[19] S. Chandel, S. Yu, T. Yitian, Z. Zhili, and H. Yusheng, "Endpoint Protection: Measuring the Effectiveness of Remediation Technologies and Methodologies for Insider Threat," in *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, Guilin, China, Oct. 2019, pp. 81–89, https://doi.org/10.1109/CyberC.2019.00023.

[20] S. J. Yoo, "Study on Improving Endpoint Security Technology," *Convergence Security Journal*, vol. 18, pp. 19–25, 2018.