Attack Graph-Based Moving Target Defense in Software-Defined Networks

Seunghyun Yoon, *Student Member, IEEE*, Jin-Hee Cho, *Senior Member, IEEE*, Dong Seong Kim, *Senior Member, IEEE*, Terrence J. Moore, *Member, IEEE*, Frederica Free-Nelson, and Hyuk Lim, *Member, IEEE*

***Abstract*—Moving target defense (MTD) has emerged as a proactive defense mechanism aiming to thwart a potential attacker. The key underlying idea of MTD is to increase uncertainty and confusion for attackers by changing the attack surface (i.e., system or network configurations) that can invalidate the intelligence collected by the attackers and interrupt attack execution; ultimately leading to attack failure. Recently, the significant advance of software-defined networking (SDN) technology has enabled several complex system operations to be highly flexible and robust; particularly in terms of programmability and controllability with the help of SDN controllers. Accordingly, many security operations have utilized this capability to be optimally deployed in a complex network using the SDN functionalities. In this paper, by leveraging the advanced SDN technology, we developed an attack graph-based MTD technique that shuffles a host's network configurations (e.g., MAC/IP/port addresses) based on its criticality, which is highly exploitable by attackers when the host is on the attack path(s). To this end, we developed a hierarchical attack graph model that provides a network's vulnerability and network topology, which can be utilized for the MTD shuffling decisions in selecting highly exploitable hosts in a given network, and determining the frequency of shuffling the hosts' network configurations. The MTD shuffling with a high priority on more exploitable, critical hosts contributes to providing adaptive, proactive, and affordable defense services aiming to minimize attack success probability with minimum MTD cost. We validated the out performance of the proposed MTD in attack success probability and MTD cost via both simulation and real SDN testbed experiments.**

***Index Terms*—Moving target defense, proactive/adaptive defense, asset criticality, hierarchical attack graph, attack path prediction, network address shuffling, software-defined networking.**

I. Introduction

MOVING target defense (MTD) has emerged as a proactive defense technique to thwart and confuse potential attackers aiming to penetrate a system by exploiting system vulnerabilities [11]. MTD has been studied as one of several network obfuscation approaches designed to mislead attackers by changing the attack surface (i.e., the system configurations) by using IP shuffling or randomization [20], [26], packet header randomization [36], network topology shuffling [5], or migration of system platforms [14]. The recent advance of software-defined networking (SDN) technology has been leveraged to effectively and efficiently deploy various types of MTD techniques. The principal merit of the SDN technology is to decouple the network control plane from the data-forwarding plane to enhance flexibility, robustness, and programmability to a networked system by using an SDN controller that can readily deploy MTD techniques.

In this paper, we leverage the SDN technology to propose an MTD framework that can provide a solution to determine how often each host's network configuration can be shuffled to provide adaptive, proactive, and affordable security services. To provide highly cost-effective security services, the approach proposed in this paper focuses on shuffling network configurations of highly critical, vulnerable hosts that can significantly attract attackers aiming to exploit the vulnerabilities on the attack paths with the hosts.

The fundamental underlying idea is that given each host with a different level of asset criticality if more critical hosts are protected with high priority, then it will significantly contribute to building highly secure and dependable systems with low defense cost. To this end, we develop an exploitability prediction algorithm for each attack path and estimate an expected attack success probability (i.e., a likelihood of an attacker successfully compromising a critical target host). When a host is detected as being on a highly vulnerable, exploitable attack path by the proposed exploitability prediction algorithm, hosts on the attack path

Manuscript received September 17, 2019; revised February 19, 2020; accepted April 6, 2020. Date of publication April 10, 2020; date of current version September 9, 2020. This work was partially supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No. 2017-0-00421), and by the U.S. Army Combat Capabilities Development Command (CCDC) International Technology Center - Pacific (ITC-PAC) and CCDC Army Research Laboratory (CCDC-ARL) under Cooperative Agreement (FA5209-19-P-A056). The associate editor coordinating the review of this article and approving it for publication was C. Fung. *(Corresponding author: Hyuk Lim.)*

Seunghyun Yoon and Hyuk Lim are with the School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology, Gwangju 61005, South Korea (e-mail: seunghyunyoon@gist.ac.kr; hlim@gist.ac.kr).

Jin-Hee Cho is with the Department of Computer Science, Virginia Tech, Falls Church, VA 22043 USA (e-mail: jicho@vt.edu).

Dong Seong Kim is with the School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, QLD 4072, Australia (e-mail: dan.kim@uq.edu.au).

Terrence J. Moore and Frederica Free-Nelson are with the Network Science Division, U.S. Army Research Laboratory, Adelphi, MD 20783 USA (e-mail: terrence.j.moore.civ@mail.mil, frederica.f.nelson.civ@mail.mil).

Digital Object Identifier 10.1109/TNSM.2020.2987085

have a priority to be shuffled over other hosts, which can provide adaptive, proactive, and low-cost defense services. Note that the frequency of triggering an MTD operation in each host (i.e., changing the host's network configurations such as MAC/IP/port addresses) is directly related to both the security and performance. Thus, executing MTD operations more frequently is more likely to increase a security protection but incurs more MTD cost due to more frequent changes in the network configurations of hosts. The **key contributions** of this work are:

- We present a new graphical model for attack graphs called the *three-tier attack graph (TAG) model*. This model is developed to estimate the compromise probability of a host by using the host's vulnerability information. We perform the host's exploitability analysis using the TAG model.
- We construct an *asset criticality-aware attack graph* based on the TAG model to identify the critical components of a system. We propose an *attack path prediction method* to protect critical assets by selecting k most vulnerable paths among all the possible attack paths for each host. We also develop two novel asset criticality metrics, called *role-based criticality* (RC) and *influence-based criticality* (IC), as a measure of a host's criticality to identify the more critical hosts that require a higher priority protection.
- We propose an *overhead-controllable address shuffling* method that changes a host's network configurations by selecting a single host based on the asset criticality at every shuffling interval. This approach allows the security administrator to operate MTD without affecting system performance by controlling the shuffling interval.
- We consider various advanced attack techniques such as reconnaissance attacks, forensic attacks [6], and topology poisoning attacks [18]. Our proposed SDN-based MTD technique can defeat the attackers using network forensic techniques such as flow rule reconstruction and IP/MAC address revealing techniques because it does not expose the real address to the attackers and can manipulate the header and payload information appropriately by using SDN functionalities.
- We implement the proposed MTD technique in SDN and demonstrate its out performance in terms of attack success probability and MTD shuffling cost in both Mininet-based emulation and SDN testbed experiments.

The remainder of this paper is organized as follows. Sections II and III give an overview of the related work and background of security metrics and SDN-based MTD techniques. Section IV describes our system model, network model, attack model, exploitability metric, and its analysis methodology. Sections V and VI provide the details of the proposed asset criticality-aware MTD in SDNs. Section VII demonstrates experimental results to validate the performance of the proposed MTD. Finally, Section VIII summarizes our key findings.

II. Related Work

MTD aims to increase uncertainty and confusion for attackers attempting to penetrate into a system by identifying system vulnerabilities. The main function of MTD is to change the attack surface (i.e., the system/network configurations), consequently invalidating the intelligence gathered by the attackers and wasting their resources (e.g., time and cost). To provide a brief overview of the state-of-the-art MTD approaches, we categorize MTD techniques in terms of shuffling, diversity, and redundancy; following the classification discussed in [11].

Shuffling-based MTD is the most common MTD that rearranges or randomizes system configurations such as IP shuffling or randomization [9], [20], [26], [33], packet header randomization [36], virtual machine or proxy migration [30], or software/service reconfiguration [10], [35]. Some recent studies have introduced topology shuffling-based MTD techniques by creating a network topology using honeypots [5] or diversifying routing paths [17]. However, these works did not consider different levels of vulnerabilities of attack paths with highly critical nodes, which are discussed in this paper.

Diversity-based MTD provides the capability to deploy different implementations of the same functionalities or services. The examples include software stack diversity [19] to enhance network resilience and service provisions, and programming language diversity [34] to avoid code injection attacks.

Lastly, *redundancy-based MTD* improves system reliability by creating multiple replicas of network components such as redundancy of network sessions in cyber-physical systems [23]. Redundancy is used in conjunction with the shuffling or diversity techniques to implement the MTD.

Some of the research on MTD mentioned above leveraged the SDN technology by utilizing SDN controllers such as network topology shuffling-based MTD [17], an OpenFlow random host mutation (OF-RHM) architecture [20], and port-hopping MTD [12]. However, our proposed method differs from these SDN-based MTD techniques in that we validated our MTD based on a three-tier attack graph security model aiming to minimize attack success and defense cost.

III. Background

In this section, we provide backgrounds on system vulnerabilities, attack graphs, and asset criticality in networks.

A. Security Vulnerabilities

Vulnerability refers to the weakness of a computer system that can be exposed under the event of a threat. The *national vulnerability database* (NVD) [21] is the vulnerability data repository owned by the U.S. government, where the security content automation protocol represents the data. The NVD contains security checklist references, security-related software flaws, misconfigurations, product names, and impact metrics. The NVD analyzes entries on the *common vulnerabilities and exposures* (CVE) [21] by aggregating data points from various reference articles in the public domain.

Common vulnerability scoring system (CVSS) [21] is the most popular vulnerability scoring metric to assess the severity of computer system security vulnerabilities based on the CVE information. The CVSS score indicates only the severity of a vulnerability on a given host; not the severity of the entire system. CVSS comprises three metric groups: the *base metric* group, the *temporal metric* group, and the *environmental metric* group. A score is mainly obtained using a base metric (i.e., [0, 10] with 10 being the most severe), and can be refined by considering temporal and environmental metrics.

To quantify the severity of vulnerabilities, an exploitability metric has been introduced to ascertain the probability of a vulnerability being successfully exploited by an attacker. The exploitability can be estimated by the following two methods: (1) *CVSS-based exploitability estimation* using the CVSS scores and the corresponding exploitability of the vulnerabilities [21]; and (2) *time-to-compromise (TTC)-based exploitability estimation* using the concept of the number of attack trials until reaching attack success. Longer TTC indicates higher security in the system (e.g., survivability) while shorter TTC indicates high system vulnerability, as a defender's metric [22]. On the other hand, as an attacker's metric, TTC is also used to measure the strength of attackers; whereby shorter TTC indicates a stronger attack.

B. Attack Graph

An attack graph (AG) illustrates the relationships among various vulnerabilities exploitable by an attacker and the privileges obtainable by the attacker. Depending on the representations of nodes and edges, different AGs can be generated. Typically, in an AG, a node represents a state (e.g., host, privilege, and vulnerability), and a directed edge denotes to a state transition (i.e., node *A* to node *B*, remote shell to *sftp*). In the AG, an attacker's exploit of a vulnerability often leads to privilege escalation on hosts (e.g., acquiring root access).

Reference [31] proposed an AG concept in which each node contains five fields: user level (e.g., none, guest, privileged, root, or admin), machine (e.g., single host, subnet, or multiple subnets), vulnerabilities, capabilities (e.g., read, write, install a virus), and state(s). Each edge contains two fields: an action and a condition. Although this model is not scalable, it allows the modeling of dynamic aspects of the network. Reference [29] introduced an exploit dependency graph, in which each node is a privileged node, a pre-condition node (i.e., vulnerability), or a post-condition node (i.e., exploit), and each edge represents a state transition. Reference [25] introduced the concept of a Bayesian AG that models potential attack paths by using a Bayesian network. Given each node representing a state, a single node is represented by multiple nodes with different user privileges or states. On edges, the conditional probability tables are assigned to infer the exploitability using the Bayesian model.

However, conventional AGs have shown some limitations in terms of scalability [16] and dynamic reconfigurations of AGs [7]. In large-scale networks, the complexity of finding all attack paths in the AG is known to be exponential. Previous efforts to mitigate the complexity are mostly based on heuristics [32]. In addition, inherent dynamics derived from changing network topology or configurations can naturally affect the attack graphs, which has not been addressed well in the literature [7].

IV. System Model

This section describes our network model, attack model, three-tier attack graph (TAG), and exploitability metric.

A. Network Model

We utilize an SDN technology to deploy the MTD technique proposed in this paper. However, implementing a shuffling-based MTD technique with the SDN technology requires a set of components and protocols to make shuffling decisions and perform address translations. In this section, we describe the architecture of our proposed shuffling-based MTD and the communication protocols between an SDN controller and SDN-enabled switches to map virtual addresses to real addresses or vice-versa. The SDN controller manages the switches in the network to control packet forwarding decisions while SDN-enabled switches only deal with forwarding packets. The SDN-enabled switches are configured to encapsulate packets that have no exact matching flow rules in flow tables, and the encapsulated packets, called "OFPT_PACKET_IN" packets in OpenFlow (OF) protocol, are forwarded to the SDN controller for the handling of the flow. Note that OF is a standard protocol for the communication between SDN-enabled switches and an SDN controller. In this work, we assume that the SDN controller and control channel are trusted; the case of the SDN controller or the control plane of the SDN being compromised by the attacker is out of the scope of our paper.

In our proposed architecture, only the SDN controller knows the real IP and media access control (MAC) addresses and active port numbers of hosts, while other hosts in the network only use virtual addresses to communicate with each other. After the SDN controller receives new packets from SDN-enabled switches, it determines the process for mapping from virtual to real addresses and sends "FLOW_MOD" packets to install appropriate flow rules in the SDN-enabled switches. Each switch uses the flow rules to convert the real address into a virtual address or vice versa in the packet header. At the switch right before a packet is delivered to the destination host, the flow rules reconvert only the virtual address of the destination host in a packet header to its real address. Therefore, both the source host and destination host do not know each other's real address. However, each host is aware of its own real address; upon a host being compromised, the attacker may find the real address of the compromised host. Hence, the underlying idea of MTD techniques is to prevent potential outside attackers from compromising legitimate hosts by dynamically changing the legitimate host's IP/MAC/port addresses. After a host is compromised, it is treated as an inside attacker, which should be detected by an IDS placed in a given defense system.

The SDN controller mainly deals with the following tasks: (1) making shuffling decisions of each host; (2) selecting virtual addresses and port numbers that do not overlap;

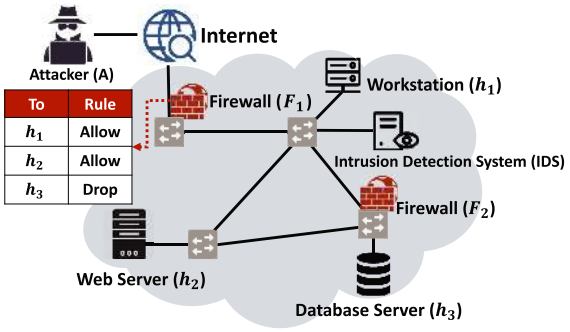


Fig. 1. An example of network topology.

TABLE I
VULNERABILITIES OF AN EXAMPLED NETWORK

Host	Vul.	CVE ID	Cause of vul.	Access	CVSS
h_1	v_1	CVE-2002-1644	Remote shell	Remote	6.5
	v_2	CVE-2007-5616	Privilege escalation	Remote	7.2
	v_3	CVE-2006-2421	Buffer overflow	Remote	6.5
	v_4	CVE-2007-5616	Privilege escalation	Local	7.2
h_2	v_5	CVE-2015-4108	Remote code exec.	Remote	6.8
	v_6	CVE-2018-9843	Remote code exec.	Remote	9.8
	v_7	CVE-2008-2384	SQL injection	Remote	7.5
	v_8	CVE-2017-6516	Privilege escalation	Local	7.2
h_3	v_9	CVE-2007-6304	Denial-of-Service	Remote	5.0
	v_{10}	CVE-2008-3234	Remote shell	Remote	6.5
	v_{11}	CVE-2001-1180	Privilege escalation	Local	7.2

(3) mapping from real address to virtual address during communication, and (4) ensuring zero-performance degradation by allowing connection with the previous IP for a certain period of time. The detailed procedure of shuffling-based MTD in SDN is discussed in Section VI.

Fig. 1 shows an example of network topology, in which the workstation (WS) handles user requests and the database (DB) server houses critical data such as personal credentials or enterprise business plans. In addition, there are a few network security functions, such as firewalls and an IDS in the network. We assumed the use of a firewall to implement traffic-blocking policies and an IDS to inspect network traffic flows.

B. Attack Model

We consider the following attack behaviors in this work:

- *Identification and analysis of the target system by scanning attacks:* Attackers can leverage various scanning tools to identify the target system's information [8]. Moreover, attackers may conduct reconnaissance attacks and privilege escalation attacks to identify vulnerable components and set their attack goals. A different attack goal may require the attacker to access other systems inside the target network.
- *Forensic attacks:* Attackers can know whether or not the victim network is an SDN by fingerprinting techniques [13], and conduct flow rule reconstruction attacks with several probing packets (e.g., ARP or ICMP packets [6]) to obtain network configuration information.
- *Topology poisoning attacks:* After successfully compromising one of the hosts in a target system, adversaries can perform topology poisoning attacks, such as link layer discovery protocol (LLDP) spoofing attacks, to mislead the topology view of an SDN controller. The compromised hosts are usually exploited to capture the packets destined to other hosts to find the next victim candidates in the network [18]. In [18], a topology update verification was proposed to defend against the topology poisoning attacks.
- *Data exfiltration attacks:* The attacker iteratively repeats a sequence of exploiting steps (i.e., identifying a target system and components inside the target and compromising them to achieve its goal) and extracts confidential data, which can lead to the breach of data integrity and confidentiality. To gain the desired privilege to access a target host, an attacker exploits single or multiple vulnerabilities on each host.

As shown in Fig. 1, active hosts in the example network are divided into two types: hosts that can be accessed by outside entities (denoted by H_{ex}) and hosts that can be connected by only internal entities (indicated by H_{in}). A H_{ex} type host can be exploited and accessed directly by remote attackers through initial vulnerabilities on the host. This implies that an attacker is unauthorized at the outset but can compromise its direct neighbors without any access rights to their settings or files. By contrast, the attacker cannot reach a H_{in} type host directly without first compromising a host that can directly communicate with it. In Fig. 1, h_1 and h_2 are H_{ex} type hosts because the firewall (F_1) rule allows the attacker to reach only the hosts h_1 and h_2. Conversely, h_3 is a H_{in} type host because the firewall rule blocks the access by the remote attacker.

A legitimate user can log in to any host by using two types of accounts: a user privilege and root privilege. However, an attacker must acquire an appropriate privilege of a target host to exploit its vulnerability. For simplicity, in this work, we assumed that attackers could exploit a target host's vulnerability. Moreover, in a given network, active hosts consist of both H_{ex}'s and H_{in}'s. In the example network provided in Fig. 1, we assumed that an attacker is initially located outside the network. The primary goal of the attacker is to exploit vulnerable components of the target system and obtain valuable information from the DB server. Further, we assumed that the hosts have inherent vulnerabilities which are summarized in Table I and indexed by a CVE number with a CVSS severity score. Specifically, the attacker may exploit vulnerability CVE-2002-1644 in h_1 for launching a privilege escalation attack in the first round of the attack. h_1, as H_{ex}, is remotely accessible by anonymous users on the Internet; thus it exposes its vulnerabilities that allow a write operation on its home directory. An exploitable command shell assigned to h_2 (as H_{ex}) and h_3 (as H_{in}) can be exploited by a remote code execution attack via system vulnerabilities, such as CVE-2015-4108 and CVE-2018-9843. The attacker finally gains root privilege of the DB server h_3 by exploiting the system vulnerability CVE-2001-1180.

C. Three-Tier Attack Graph (TAG)

We propose a new graphical model for an AG in which the network topology information and vulnerability information

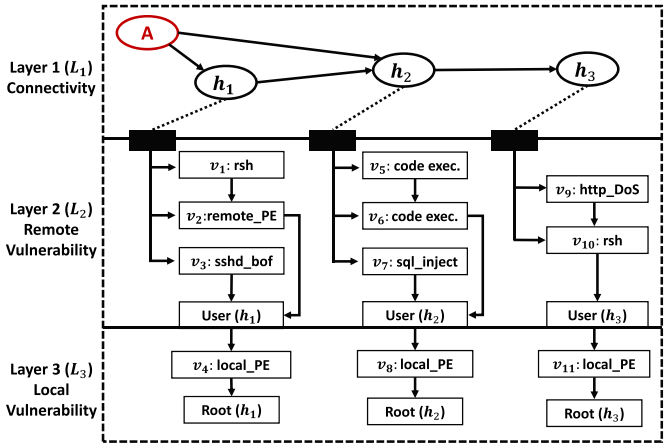


Fig. 2. Three-tier attack graph (TAG).

of network hosts can be considered as separate layers. The key benefit of separating network topology information from system vulnerability information on an AG is lowering the complexity of the AG generation and eliminating the dependency of security assessments on various network reachability constraints such as firewall configurations and network routing policies. In general, an AG has the form of a directed graph with loops because the hosts in the network are interconnected; however, the loops in the AG make it harder to find attack paths. In the proposed TAG model, if a given AG with loops is cyclic, we can convert it to a directed acyclic graph (DAG) by applying either the cycle handling method [24] or *d*-separation algorithm [15] without any loss of integrity.

In our proposed model, each host exposes a set of its own vulnerabilities. Moreover, we generate a TAG based on the type and characteristics of each host's vulnerabilities. As described in Fig. 2, we propose a TAG model with three layers, consisting of connectivity, remote vulnerability, and local vulnerability layers. Each layer is detailed as follows:

- *Layer 1 (L_1) with network connectivity:* This layer is determined by the network topology and firewall configuration, which provides reachability information.
- *Layer 2 (L_2) with remote vulnerability:* This layer shows vulnerabilities exposed by remote hosts that can directly access hosts in a target network. In this layer, the listed vulnerabilities are bound to the network stack and can be exploited by remote entities that are one or more hops away from a target host without any permission from the host.
- *Layer 3 (L_3) with local vulnerability:* This local layer shows vulnerabilities exposed by internal hosts whose compromise requires obtaining a local root privilege. In this layer, since the vulnerable components are not tied to the network stack, an attacker must gain the root privilege of a target host to exploit its vulnerabilities.

Fig. 2 depicts the TAG generated based on the network topology in Fig. 1 and Table I. L_1 provides information describing connectivity between network hosts and the attacker. Given that each host has some vulnerabilities, L_2 represents the relationships of the remote vulnerabilities which can be exploited by the remote attacker. For example, the host h_1 has three possible remote vulnerabilities, along with the combinations of the vulnerabilities. By using those vulnerabilities, the potential attack paths are $(v_1 \rightarrow v_2 \rightarrow User\ (h_1))$, $(v2 \rightarrow User\ (h_2))$, and $(v_3 \rightarrow User\ (h_2))$. The attacker can get the user privilege of the host h_1 by exploiting one of these attack paths. L_3 denotes the local vulnerabilities that can be exploited to obtain a host's root privilege.

To address the scalability problem of AG, the two-layer (TL-AG) attack graph has been proposed and shown to improve the scalability in AGs [16]. The key idea of TL-AG is to separate reachability information from vulnerability information for their better controllability on AGs. We take the merit of this approach in a similar manner but added an additional layer by dividing the vulnerability information into two layers: remote vulnerability and local vulnerability. This allows us to handle both user privilege and root privilege compromise for capturing attack paths in a given network.

D. Exploitability Metric

In this section, we explain how to calculate the probability that an attacker successfully exploits a given host's vulnerability, called *exploitability*. Given that the attack goal is to compromise a host, we define the exploitability as the probability of an attacker exploiting a host's single vulnerability to compromise the host successfully. Denote the probability of successfully exploiting the vulnerability v of a host (i.e., exploitability) by $P_e(v)$. For example, if $P_e(v_1)$ is greater than $P_e(v_2)$, an attacker can more easily exploit v_1 than v_2. We estimate $P_e(v)$ based on the CVSS severity score by:

$$P_e(v) = \frac{\mu(v)}{\mathrm{VS}_{\max}}, \tag{1}$$

where $\mu(v)$ is the severity score of vulnerability v on the CVSS scale and $\mathrm{VS}_{\max}$ is the maximum severity score of vulnerabilities (i.e., 10 in the CVSS score [28]).

Some other terms used in this paper to measure exploitability are compromise probability and attack success probability.

- *Compromise probability* refers to the probability of the user or root privilege of a single host being successfully compromised by an attacker. Depending on the attacker's purpose, it can exploit several vulnerabilities of the host to obtain a proper privilege.
- *Attack success probability* is the probability that an attacker's given goal is successfully achieved. For example, it refers to the probability that an attacker can access a target host in a given network. Note that this probability can be measured as a generic metric with a different definition of attack success.

To compromise a target host, an attacker needs to compromise all the hosts on the path to the target host. We assumed that an attacker does not have prior knowledge about the vulnerabilities of all hosts in a network and the entire network topology. Further, we assumed an intelligent attacker will aim to select a next host with low attack complexity but high exploitability vulnerability based on CVSS information to reach the target host.

E. Estimating Per-Host Exploitability Based on Vulnerabilities

In this section, we describe how to estimate the per-host exploitability (i.e., compromise probability) based on the vulnerability information of a host. Due to the complexity of analyzing vulnerability information based on an AG, we generalized how exploitability of a vulnerability is analyzed to determine the attack success probability of a node. We measure the following three types of host's exploitability associated with the vulnerabilities to respective layers: *exploitable vulnerabilities* in L_1 (connectivity), *user privilege vulnerability* in L_2 (remote vulnerability), and *root privilege vulnerability* in L_3 (local vulnerability).

1) Exploitable Vulnerabilities in L_1: A host *h* can have two sets of adjacent hosts: hosts with out-degree, denoted by $\mathcal{H}^{out}(h)$, and hosts with in-degree, denoted by $\mathcal{H}^{in}(h)$. For example, $\mathcal{H}^{out}(A) = \{h_1, h_2\}$ and $\mathcal{H}^{in}(h_2) = \{A, h_1\}$ in Fig. 2. Let $\mathcal{V}_r(h)$ denote a set of remotely accessible vulnerabilities in the host *h*. Then, $\mathcal{V}_r(h_1) = \{v_1, v_2, v_3\}$, $\mathcal{V}_r(h_2) = \{v_5, v_6, v_7\}$, and $\mathcal{V}_r(h_3) = \{v_9, v_{10}\}$. Because an attacker can obtain the information about vulnerabilities of its adjacent hosts, it will select one of the vulnerabilities as its next vulnerability to exploit. In Fig. 2, the attacker *A* attempts to exploit a vulnerability in a set of $\mathcal{V}_r(h_1) \cup \mathcal{V}_r(h_2)$ for $\mathcal{H}^{out}(A) = \{h_1, h_2\}$. We assume that each attacker chooses the next vulnerability to exploit, with a probability in proportional to the exploitabilities of the next vulnerabilities accessible remotely. Then, the probability of host *h* selecting the next victim host $h_j \in \mathcal{H}^{out}(h)$, $P^{AE}_{h_j}(h)$, is obtained by:

$$P^{AE}_{h_j}(h) = \frac{\sum_{v \in \mathcal{V}_r(h_j)} P_e(v)}{\sum_{h_i \in \mathcal{H}^{out}(h)} \sum_{v \in \mathcal{V}_r(h_i)} P_e(v)}. \quad (2)$$

2) Exploitability of User Privilege Vulnerabilities in L_2: We denote the probability of the host h_j's user privilege being exploited by an attacker (called the *user privilege exploitability*) by $P_{UE}(h_j)$. $P_{UE}(h_j)$ refers to the probability that an attacker successfully exploits the remote vulnerabilities of host h_j to gain h_j's user privilege. Usually, attackers compromise a user privilege by exploiting the file transfer protocol (FTP) or remote shell vulnerabilities, called *remote command execution vulnerabilities*. Host h_j's user privilege exploitability, $P_{UE}(h_j)$, is estimated by:

$$P_{UE}(h_j) = 1 - \prod_{v \in \mathcal{V}_r(h_j)} \left[1 - \prod_{u \in APV(v,\, U(h_j))} P_e(u) \right], \quad (3)$$

where $APV(v, R(h_j))$ is the set of vulnerabilities existing in all possible attack paths from vulnerability *v* to host h_j's user privilege and $P_e(u)$ is the exploitability of vulnerability *u* on the attack path $APV(v, U(h_j))$. In Fig. 2, host h_1 has three remotely accessible vulnerabilities; $\mathcal{V}_r(h_1) = \{v_1, v_2, v_3\}$ and $P_{UE}(h_1) = 1-(1-P_e(v_1)P_e(v_2))(1-P_e(v_2))(1-P_e(v_3))$.

While $P_{UE}(h_j)$ in (3) is the exploitability for each individual host h_j, it is needed to obtain the compromise probability that takes into account the attacker's attack paths. The probability that the host h_j's user privilege is successfully compromised on the attack paths, $P_{UC}(h_j)$, is obtained by:

$$P_{UC}(h_j) = P_{UE}(h_j) \sum_{h \in \mathcal{H}^{in}(h_j)} \Big[P_{UC}(h)\ P^{AE}_{h_j}(h) \Big]. \quad (4)$$

Note that $P_{UC}(h_j)$ in (4) is dependent on the other user privilege compromise probabilities of previous hosts, which are the in-degree hosts of h_j. The computation of $P_{UC}(h_j)$ can be sequentially done on the attack paths from the H_{ex} nodes to the H_{in} nodes.

3) Exploitability of Root Privilege Vulnerabilities in L_3: If an attacker is legitimately authorized with privileges obtained by successfully exploiting internally accessible vulnerabilities that provide an administrative control over a host, it can eventually affect the kernel settings and system files of the host. Let $P_R(h_i)$ denote the probability of the host h_i's root privilege being compromised by successfully exploiting all internally accessible vulnerabilities. $P_R(h_i)$ is given by:

$$P_R(h_i) = P_{UC}(h_i) \sum_{v \in \mathcal{V}_{rt}(h_i)} \prod_{u \in APV(v, R(h_j))} P_e(u), \quad (5)$$

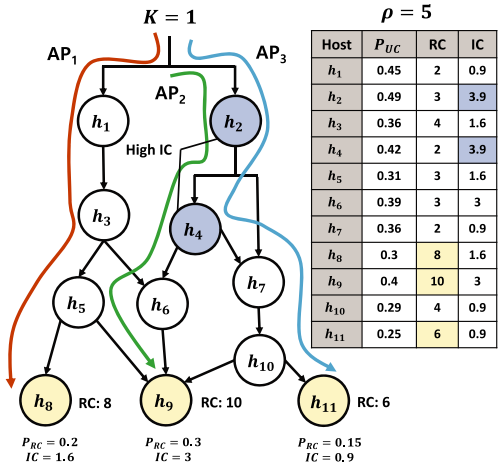
where $\mathcal{V}_{rt}(h_i)$ refers to a set of vulnerabilities associated with the host h_i's root privilege, and $APV(v, R(h_j))$ is the set of vulnerabilities existing in all possible attack paths from vulnerability *v* to host h_j's root privilege. To avoid any path explosion problem upon the existence of multiple possible attack paths to compromise the root privilege of a host, we will use backward attack path (BAP) prediction algorithm, which considers a host with multiple vulnerabilities over a host with a single vulnerability. The probabilities of a host's user privilege in (4) and a host's root privilege in (5) being successfully compromised indicate the cumulative effects by attack steps, showing how a series of individual exploits can allow an attacker to achieve its final attack goal.

V. Asset Criticality-Aware MTD

After the per-host exploitability analysis with the TAG model, we construct an asset criticality (AC)-aware AG using the L_1 connectivity information of the TAG to predict the possible vulnerable attack paths. In this section, we describe the proposed asset criticality metrics, asset criticality-based shuffling, attack path prediction method (i.e., BAP prediction algorithm), and the deployment procedure of the asset criticality-aware MTD.

A. Asset Criticality

1) Role-Based Criticality (RC): When a system (or network) is designed to provide certain services, each host is given its own embedded capability. This capability can be utilized to determine the host's criticality based on its role. In this paper, we call it a host's *role-based criticality*. For example, a networked system consists of several types of assets, such as DB servers, authentication servers, and Web servers. These servers are usually considered more important than other typical user hosts; thus, they require higher levels of security



Host	P_{UC}	RC	IC
h_1	0.45	2	0.9
h_2	0.49	3	3.9
h_3	0.36	4	1.6
h_4	0.42	2	3.9
h_5	0.31	3	1.6
h_6	0.39	3	3
h_7	0.36	2	0.9
h_8	0.3	8	1.6
h_9	0.4	10	3
h_{10}	0.29	4	0.9
h_{11}	0.25	6	0.9

Fig. 3. Asset criticality-aware attack graph.

protection and performance maintenance in order to seamlessly provide normal, reliable services without interruptions and without being compromised.

We devise a role criticality (RC) metric, representing a host's criticality level in terms of its role in a given system. We initialize each host's RC level, which is an integer ranging in [1, 10] and denoted by RC_{h_i} (for the host h_i). For example, in a cloud data center, one of the most critical assets is the DB server. Hence, we assumed that the DB server's RC is higher than those of other host virtual machines (VMs), i.e., $RC_{DB} > RC_{VM}$. In Fig. 3, the three nodes h_8, h_9, and h_{10} are the servers to be protected with high AC levels; for example, $RC_{h_8} = 8, RC_{h_9} = 10$, and $RC_{h_{10}} = 6$, respectively.

To estimate an RC (the RC_{h_i} for host h_i), we identify attack sequences starting from externally accessible hosts (i.e., H_{ex}) to hosts with high RC by using the information from L_1 and L_2 in the TAG model. Then, if a host is on the paths to hosts with higher RC, the network configurations for the host (i.e., MAC/IP/port addresses) should be shuffled more frequently. For example, in Fig. 3, if the attacker successfully compromises h_1, the attacker has a chance to attack h_8 and h_9. Therefore, if a host is on the path to other critical assets (e.g., h_4), the host's IP address should be shuffled more frequently. In this work, we define the attack success probability as the probability of a critical asset's root privilege being successfully compromised by an attacker. Furthermore, attackers are assumed to exploit the user privileges of other remaining hosts, called *intermediate hosts*, which are not critical assets. For simplicity, we define a critical asset with $RC_{h_i} > \rho$, where ρ is a threshold to determine highly critical assets.

2) Influence-Based Criticality (IC): A host h_i's role-based criticality, RC_{h_i}, is determined based on its role in terms of its service provision. However, RC_{h_i} does not reflect h_i's influence in terms of its location in a network. For example, some hosts are involved with multiple and more attack paths while others are not. Thus, depending on how many attack paths the host is involved with, its influence on exposing path vulnerabilities to attacks varies. To consider a host's network influence on the degree of vulnerabilities exposed to attacks, we devise an *influence-based criticality* metric; denoted by IC_{h_i} for host h_i. We calculate IC_{h_i} for all highly critical assets, h_i's (with $RC_{h_i} > \rho$), based on the identified attack paths they are on. In Fig. 3, since ρ is set to five, only hosts h_8, h_9, and h_{11} are considered as critical assets.

Given that *n* is the number of highly critical hosts and *k* is the number of the most vulnerable attack paths for each critical host identified by the BAP prediction algorithm (discussed below in Section V-B), IC_{h_i} is obtained as follows:

$$IC_{h_i} = \sum_{p \in AP_{BAP}(h_i)} RC_{eh(p)} \cdot P_R(eh(p)), \qquad (6)$$

where $AP_{BAP}(h_i)$ is a set of paths passing through the host h_i obtained from the BAP prediction algorithm, *eh*(*p*) is the end host that is finally targeted (i.e., critical assets) by the attack path *p*, RC_h is host *h*'s RC (which is given), and $P_R(h)$ is the probability of the host *h*'s root privilege being compromised in (5). In Fig. 3, IC_{h_2} (which is shown with a blue-colored host) is 3.9 because the attack paths passing through h_2 are AP_2 and AP_3. In addition, IC_{h_9} and $IC_{h_{11}}$ are 3 and 0.9, respectively, where h_9 and h_{11} are target hosts on AP_2 and AP_3.

B. Prediction of Attack Paths to Target Hosts

Asset criticality-aware MTD operation should be based on the accurate prediction of the exploitability of an attack path. In this section, we discuss two attack path prediction algorithms: the brute-force (BF) search and the proposed backward attack path (BAP) prediction algorithms. BF is used to identify the optimal solution(s) and to estimate the prediction accuracy of the BAP over BF. To show the benefit of the BAP for scalability, we will compare the BF and BAP in terms of running time in Section VII.

An attack path is a sequence of attacks that can be identified on a TAG, starting from an externally accessible host (i.e., H_{ex}) to an internally connectable host (i.e., H_{in}) with high AC level (i.e., a victim host). Utilizing the TAG model, we compute h_i's exploitability, $P_{UC}(h_i)$ for all h_i's in (4) and $P_R(h_i)$ for potential victim hosts in (5). Fig. 3 shows an asset criticality-aware AG, wherein h_1 and h_2 are the externally accessible hosts, meaning there are no in-degree edges to the hosts, whereas, h_8, h_9, and h_{10} are the target servers with high AC level (i.e., potential victim hosts). In Fig. 3, the table in the figure summarizes the compromise probability of each host's user/root privilege, and the RC/IC values of a host. We assumed that an attacker's ultimate goal is to compromise the root privilege of the critical assets. In this case, one of the attack paths to h_8 is AP_1 (i.e., $h_1 \rightarrow h_3 \rightarrow h_5 \rightarrow h_8$ as shown in an orange-colored curve in Fig. 3).

1) Brute-Force Search: Finding all attack paths in an AG has exponential time complexity. One common way to find attack paths with highly critical asset nodes on the AG is a depth-first search (DFS) and topological sorting (TS)-based brute-force (BF) search method. TS is a linear ordering of vertices of a DAG for every directed edge on the graph. The BF-based attack path identification method finds all the attack paths for all H_{ex} and critical asset pairs on an AG. A valid sequence of an attack path can be obtained by applying the

Algorithm 1 Backward Attack Path (BAP) Prediction

Input: TAG, $A \leftarrow$ a set of critical assets ($\rho = 5$)
Output: k attack paths for each asset

1: **for each** asset $a_i \in A$ **do**
2: **for** $t \leftarrow 1$ to k **do**
3: $h(a_i, t) \leftarrow \text{greedyBacktrack}(a_i, t)$ ▷ $h(a_i, t)$ is the set of hosts included in the t-th attack path targeting a_i.
4: **while** $h(a_i, t).last \notin S(H_{ex}))$ **do**
5: $h_j \leftarrow \text{findMaxNeighbor}(h(a_i, t).last)$
6: **add** host h_j to host set $h(a_i, t)$
7: **end while**
8: **if** $h_j \in S(H_{ex})$ **then** ▷ $S(H_{ex})$ is a set of H_{ex}
9: **add** host h_j to host set $h(a_i, t)$
10: **end if**
11: $AP^t_{a_i} \leftarrow h(a_i, t)$ ▷ $AP^t_{a_i}$ is a_i's t-th attack path
12: **end for**
13: **end for**

BF search algorithm to the AG. In order to determine the optimal attack path in the AG, we employ the average attack trials (AAT) metric, which can be estimated by the inverse of the compromise probability of a host (i.e., $1/P_{UE}(h)$ for the intermediate host and $1/P_R(h)$ for the critical asset). By using the geometric distribution, we can obtain the required number of attack trials to get one attack success (e.g., a host is first compromised by the attacker). If the probability of success on each attack trial is p (i.e., compromise probability), then the probability that the k^{th} attack trial is the first success is given by: $Pr(x = k) = (1 - p)^{k-1}p$. For example, if the exploitability of a host is 0.2, it implies that the attacker has to make an average of five attack trials to compromise a host. In this work, we assumed that the sum of AATs represents the severity of an attack path (i.e., the higher the sum of AATs, the lower the vulnerability of an attack path). Moreover, we utilize a BF search (i.e., TS and DFS) to generate optimal solutions that can be compared against those given by our proposed attack path prediction algorithm. The performance comparison of the TS and our proposed algorithm is discussed in Section VII-D.

2) Backward Attack Path Prediction Algorithm: To mitigate high complexity in BF, we propose a greedy, heuristic prediction method, called BAP, which is a low-cost solution method that significantly reduces computational complexity. Given a network topology and path exploitabilities, we show how to predict attack paths that are more vulnerable than others and should be protected with a higher priority. To this end, we aim to find the most vulnerable path that hosts with high AC are on by backtracking from the end host (i.e., one of the most critical assets in the network), to one of the externally accessible hosts, H_{ex}'s. Algorithm 1 shows the procedure of the BAP prediction for each asset in a given network. For each asset, iterating from the most vulnerable attack path (i.e., $k = 1$) to the k-th vulnerable attack path, attack paths are computed (lines 2 to 10). Moreover, given the TAG model and the exploitability of each host, the BAP executes a series of steps to identify attack paths. In Fig. 3, suppose h_9 is first selected, then h_6 is chosen as a next node to search as it has the highest user privilege compromise probability with 0.39; thus, exposing the highest vulnerability among other neighboring hosts (i.e., nodes directly connected to h_9). The selected attack path for the target host h_9 is $h_2 \rightarrow h_4 \rightarrow h_6 \rightarrow h_9$ (which is shown with a green-colored curve in Fig. 3).

In this paper, the proposed BAP prediction algorithm identifies k attack paths for each critical asset, leading to $k \cdot n$ attack paths where n is the number of hosts with $RC_i > \rho$, and ρ is a threshold to determine highly critical asset hosts. The proposed BAP prediction algorithm does not need to enumerate all the attack paths in an AG. In Algorithm 1, for each host, we compute k attack paths based on the BAP. Instead of searching all attack paths, the BAP only searches $n \times k$ number of attack paths by using a greedy selection method to reduce computational complexity. For example, if the BAP searches $k = 1$ attack path, it will find the most vulnerable attack path exploitable by attackers; thus, k refers to the number of the most vulnerable attack paths to each target host. However, if k is too small, it is likely to mispredict the attacker's attack path. To investigate this scalability issue, we demonstrate the running time of the BAP under different k and BF in Section VII.

C. Asset Criticality-Aware Shuffling Probability

In Section V-A, we defined two types of asset criticality, including role-based criticality (RC) and influence-based criticality (IC). Accordingly, we define two shuffling methods for a host's network configurations as follows:

- *Shuffling with RC:* At every time interval T_s, each host h_i is selected to shuffle its network configuration based on the degree of RC as follows:

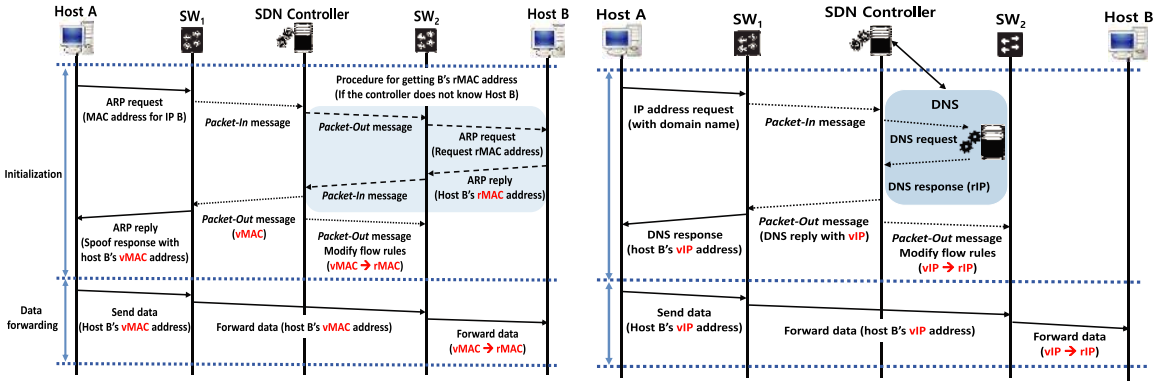
$$F_{RC}(h_i) = \frac{RC(h_i)}{\sum_{h_j \in AG(h_i)} RC(h_j)}, \tag{7}$$

  where $AG(h_i)$ is a set of hosts in an asset criticality-aware AG involving h_i, and $\sum_{h_j \in G} F_{RC}(h_j) = 1$ where G refers to a given network.
- *Shuffling with IC:* Per T_s, every host h_i is selected to shuffle its network configuration based on the degree of its IC as follows:

$$F_{IC}(h_i) = \frac{IC(h_i)}{\sum_{h_j \in AG(h_i)} IC(h_j)}, \tag{8}$$

  where $\sum_{h_j \in G} F_{IC}(h_j) = 1$ and G refers to a given network.

In both shuffling schemes mentioned above, where the sum of probability portions of all hosts is one, upon every T_s, one of the hosts is selected to shuffle its network configuration where each host has a chance to be selected with the corresponding shuffling probability. Note that 'shuffling with IC' considers a broader concept of criticality in terms of a host's importance in its role and impact in a network than 'shuffling with RC.'



(a) MAC address shuffling (b) IP address shuffling

Fig. 4. Address shuffling sequence to send data from host A to host B.

VI. SDN-Based Shuffling MTD

In this section, we discuss: (i) how to implement the proposed shuffling-based MTD with the asset criticality-awareness in Section V; and (ii) how the address shuffling mechanism can be implemented in an SDN.

A. SDN-Based MAC Shuffling

The MAC address is a primary address for communications in the data link layer network. The address resolution protocol (ARP) provides the address mapping between the data link layer (i.e., MAC) and the network layer (i.e., IP). Attackers can obtain various types of information on hosts in a target network by passively sniffing or actively broadcasting ARP packets. Notwithstanding, the static configurations of MAC addresses in wireless mobile networks are commonly exploited by attackers to track specific machines. Network discovery, the process of identifying active hosts in a network, is a crucial step in the network information gathering process. Since the primary step in all IPv4 network communication is broadcasting ARP packets to find the MAC address of the target host, an attacker can identify active hosts from the ARP response packets. By executing the address mapping process, the attacker can detect all active hosts in the target network, regardless of the presence of a firewall or an IDS.

We consider the following procedures when implementing MAC address shuffling as an obfuscation technique in an SDN. In a conventional Layer 2 switching network, a source host, h_s, retrieves the MAC address of a destination host, h_d, on its ARP cache table before sending a packet. However, if no MAC address is found, an ARP request packet will be broadcast to the network. When h_s receives h_d's MAC address, it uses the MAC address in each frame when sending a packet to h_d. In this section, we propose a MAC address obfuscation scheme that virtually and dynamically changes a host's MAC address. Here, we assume that the channel between a host and its directly connected switch is a secure point-to-point direct link, and we do not take into account scenarios in which an attacker compromises an SDN-enabled switch itself.

Fig. 4 (a) shows the obfuscation procedure of the real MAC address, *rMAC*, which virtually and dynamically changes a host's MAC address. It depicts the procedure of the proposed MAC address obfuscation scheme for a simple SDN topology consisting of two SDN switches and two hosts. The host *A* sends a packet to the host *B*, and their MAC addresses are obfuscated. At the start of a communication, the host *A* broadcasts an ARP request packet to obtain *B*'s MAC address before transmitting a data packet. The switch SW_1 receives the ARP request packet from *A* and sends a packet-in message to the SDN controller because it does not know how to handle the ARP packet. Instead of broadcasting the ARP packets to the entire network, the SDN controller responds to the ARP request packet by directly injecting the ARP reply packet through a packet-out message. However, if the SDN controller does not know *B*'s *rMAC*, it performs additional procedures to obtain *B*'s *rMAC*. Further, when the SDN controller sends the ARP reply packet to *A*, it selects a virtual MAC address, *vMAC*, for *B* and includes *B*'s *vMAC* in the ARP reply packet. In addition, the SDN controller updates the flow tables of SDN switches on the path from *A* to *B*. Here are the additional procedures of the MAC obfuscation:

- *ARP reply packet injection:* In an SDN, the SDN controller receives a 'packet-in message' from the SDN switches upon new packet arrivals. It decides where to forward a packet and then sends a 'packet-out message' to an SDN switch. If the SDN controller doesn't know h_d's location, it will forward the ARP packets to find h_d. Instead of being sent to the requesting h_s, after updating the *vMAC* on the SDN controller, the actual ARP reply packet is discarded. After this, the SDN controller blocks the ARP packet broadcast and injects an ARP reply packet containing h_d's *vMAC*. The manipulated ARP reply packet is directly retransmitted to h_s. Then, h_s attempts to communicate with h_d using *vMAC* in the ARP reply packet.
- *Spoof response for non-existing hosts:* The SDN controller responds not only to ARP requests for active hosts, but also to ARP requests for temporarily non-existing hosts, h_{fake}, in the network (i.e., some hosts' connectivity is up and down with a certain probability). If the SDN controller responds to ARP packets from h_{fake}, the attacker will spend more time in checking active hosts. Thus, attackers can easily detect any defensive deception (e.g., decoys) based on inconsistent patterns of spoof

responses. Therefore, it is vital to develop a probability model for different-sized networks and to respond to ARP requests based on it.

- *Untraceable virtual MAC address generation:* The SDN controller generates a set of *vMAC* using vendor organizational unique identifiers (OUI) information [4]. In the proposed method, the first 24 bits of the *vMAC* are determined for providing fake vendor information while the last 24 bits of *vMAC* are generated randomly. The generated *vMAC*s should not overlap *vMAC*s of other existing hosts or recently used and stored in the ARP caches of the devices.
- *Assignment of virtual MAC address:* When the SDN switches receive the ARP packet, the SDN controller looks up its ARP mapping table. If the IP address requested in the ARP packet is found in the table, the SDN controller returns the *vMAC* of the requested IP address. Otherwise, it selects one from the set of pre-generated *vMAC*s based on the hash value of the requested IP address.

We discuss the overhead of the MAC shuffling technique in a scenario where one host sends ICMP packets to another host in Section VII-G.

B. SDN-Based IP/Port Shuffling

Using the static, common addresses, such as well-known ports including SSH (20), HTTP (80), and domain name system (DNS) (53), or application-specific ports; expose security vulnerabilities because in such situations a host's running services can be easily identified by attackers capturing the packet header. To prevent the security vulnerabilities exposed by using the static network configuration, we propose a dynamic IP/port shuffling-based MTD. The proposed IP/port shuffling procedures are detailed as follows:

- *DNS server & SDN controller:* In this work, the assignment of *vIP*'s is handled by an SDN controller and a DNS server. A host's virtual IP address, *vIP*, is shuffled periodically by the SDN controller, whereas its unchanged *rIP* must always be known by the authorized DNS server. The mapping between domain name and *rIP* is handled by a DNS server, and the mapping between *rIP* and *vIP* is handled by an SDN controller. Once a host sends a DNS request packet to a DNS server, the SDN-enabled switch that is the neighbor of the sender host sends the packet-in message to the SDN controller. If the SDN controller does not have *rIP* information of a host in the network, it sends a DNS request packet to the DNS server. After getting the *rIP* information of a destination host through a DNS response packet, the SDN controller stores *rIP* information of a host. Note that the DNS response packet generated by the DNS server does not forward to the requested host. Furthermore, the SDN controller registers the mapping information of *rIP* and *vIP* of a host. Mapping of addresses changes over time upon shuffling; therefore, the SDN controller should keep track of the changed addresses using a mapping record. When the SDN-enabled switches receive a new DNS request packet, the SDN controller generates the DNS response packet, and the IP address of the requested host in the packet is changed to *vIP* based on the address mapping record. The time-to-live (TTL) value of the DNS response packet is set to a small value considering the shuffling probability of the host. If the host has a high probability of shuffling, the TTL value will be set to a relatively smaller value for considering the situations where the host's vIP address is shuffling.
- *Flow table update for address conversion:* The SDN controller updates the flow tables of switches based on the address mapping table, such that h_d's *vIP* for data packets is converted back to its *rIP* at the switch to which the h_d is attached. Moreover, *rIP* of the h_s is converted to the *vIP* so that the h_d does not know the sender's rIP address. In the proposed method, *vIP*s are used only for forwarding packets toward h_d at the core network switches. At the edge switches, *vIP* is replaced with h_d's *rIP* to ensure the delivery of a packet to the right destination.
- *Seamless conversion from the previous virtual address:* During the address shuffling time, the performance of message delivery may be degraded owing to abrupt address changes. For seamless service provisioning, after a *vIP* is shuffled, the packets that are being forwarded to the previous *vIP* should be reliably delivered to the destination in time. This can be achieved by retaining the flow table entries for the corresponding previous and current *vIP*s for a certain time interval at the SDN switches.
- *IP/Port shuffling:* To transform the end hosts into untraceable moving targets, the SDN controller periodically reassigns *vIP* to the end hosts. This shuffling of network-related addresses should be performed in an untraceable, time-variant manner. In order to maintain high variability, a new random *vIP* should be selected from the pool of unused set of *vIP*'s. Notably, it is also crucial to guarantee that the set of IP addresses that are successfully used should be exclusive in the unused set of *vIP*s. During the forwarding phase, for all flows, the source and destination ports are replaced with *vPorts*. A flow to the same port on the same h_d can have a different *vPort*.

Fig. 4 (b) shows the obfuscation procedure of IP address, host *A* attempts to send a packet to the host *B*, through the domain name. If no IP address is found in the DNS cache in *A*, a DNS request packet is sent to the DNS server. When *A* receives *B*'s *vIP* from the SDN controller, *A* uses the *vIP* in each frame it sends to *B*. The performance of the SDN-based IP/port shuffling is discussed in Section VII-G.

C. Defense Techniques Against SDN Targeting Attacks

Some attacks aim to disrupt the SDN operations by reconstructing flow rules or poisoning network topologies, which mislead an SDN controller to make poor decisions. To prevent these, we consider the following defense mechanisms:

- *SDN-based packet manipulation:* We propose an SDN-based packet manipulation technique to deal with network forensic attacks in [6]. These attacks exploit various control packets such as ARP and ICMP packets for SDN

TABLE II
KEY DESIGN PARAMETERS AND THEIR DEFAULT VALUES

Parameters	Default Value	Parameters	Default Value
N	60	ρ	5
T_r	10 (s)	T_s	3 (s)
$\mathcal{V}_h$	8	k	2

network probing and reconstruct the flow rules in the victim network. Since our SDN controller manages DNS packets and ARP packets without broadcasting, the SDN controller can prevent the network information leakage of an active host list and their MAC address caused by the packet broadcasting. In addition, it can also prevent the leakage of *rIP* in the nested ICMP packets [6] by modifying *rIP* information encapsulated in the payload of ICMP error reply packets.

- *Topology update verification [18]:* We can leverage a topology update verification mechanism in [18] to defend against network topology poisoning attacks that inform false topology information to the SDN controller. The SDN controller monitors the host-generated traffic such as ARP and DNS packets and can verify the network topology change including host discovery and migration using several predefined precondition and postcondition rules.

VII. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we discuss: (1) our experimental setup; (2) metrics; (3) MTD schemes considered for comparative performance analysis; (4) simulation results for the attack path prediction; and (5) simulation results of the proposed MTD.

A. Experimental Setup

1) Simulation Setup: We conducted simulations using BRITE [1] for generating a large-scale topology and implemented the proposed method using the Mininet SDN emulation environment [2] and an ONOS SDN controller [3]. The obtained experimental results show the average performance based on 100 simulation runs with different network topologies. Further, we investigate the effect of varying the following key design parameters on the MTD performance: (1) Each host is given a different number of vulnerabilities ($\mathcal{V}_h$) as an integer ranging in [5, 10] and each vulnerability has the CVSS score as an integer in [1, 10]; (2) The ratio of different asset criticality composition of a network where a host's AC varies from high to low critical asset hosts. ρ, a threshold to determine whether a given host is a critical asset or not is set to five; (3) the reconnaissance attack interval (T_r) to consider different levels of attack intensity; and (4) MTD shuffling interval (T_s) to consider different levels of defense strength. Table II summarizes all the key design parameters and their default values used in this work.

2) SDN-Based Testbed Setup: We implemented our proposed asset criticality-aware MTD on the ONOS SDN controller in Section VII-G. The considered network has a fat-tree topology with 10 Aruba 2920 switches (OpenFlow 1.3 support), an ONOS SDN controller (version 1.9.0), and 16 raspberry-pi end-hosts. We selected two end-hosts to measure delay based on five hops between two hosts. Note that every host is communicating with one or more hosts in the network in every second. We consider scenarios using two different MTD shuffling techniques: MAC and IP/port shuffling.

B. Metrics

1) Metrics for Attack Path Prediction: We measure the BAP prediction algorithm in terms of the following two metrics:

- *Attack path prediction accuracy* (P_{AP_k}) is measured by the number of correctly selected hosts divided by the total number of hosts for each attack path, given *k* number of attack paths identified for each critical asset. Since the BF can capture the ground truth attack paths, P_{AP_k} is given by:

$$P_{AP_k} = \frac{\left|AP_{BAP_k} \cap AP_{BF_k}\right|}{\left|AP_{BF_k}\right|}, \tag{9}$$

where AP_{BAP_k} denotes the set of hosts contained in *k* attack paths for each critical asset discovered by the BAP, and AP_{BF_k} is the set of hosts contained in *k* attack paths discovered by the BF search algorithm. The denominator is the number of hosts obtained from the BF, reflecting the ground truth attack paths identified. In this metric, higher values are more desirable.

- *Computational complexity* is measured by the running time of a given algorithm in finding the most *k* vulnerable attack paths. Lower values are more desirable in this metric.

2) Metrics for MTD Overhead: We measure the MTD overhead with the following two metrics:

- *Data plane overhead* is measured by the end-to-end delay of the two end-hosts in the SDN-based network under the address shuffling MTD being applied. Since the SDN controller rearranges the address of a host by updating the flow table in SDN switches, this can cause some end-to-end delays when forwarding data packets to their destination. In this metric, lower values are more desirable.
- *Control plane overhead* is measured by the number of control messages coming into the SDN controller through the control plane. As the SDN controller rearranges the network configuration of more hosts, the number of network control messages, such as "FLOW_MOD" packets in the OpenFlow protocol, also increases in the control plane. Lower is better in this metric.

3) Metrics for MTD Effectiveness: We measure the MTD effectiveness in terms of the following two metrics:

- *Obfuscation ratio* ($\phi(IP_t, IP_0)$) estimates obfuscated information under reconnaissance attacks, and it is defined as the number of new addresses devided by the total number of initial addresses used; thus, the obfuscation ratio, $\phi(IP_t, IP_0)$, is given by:

$$\phi(IP_t, IP_0) = \frac{\|IP_t - IP_0\|_0}{N}, \tag{10}$$

where IP_t refers to the vector of IP addresses used at the time *t*, IP_0 is the vector of IP addresses given at the time

$t = 0$, and N is the number of entire IP addresses used in the network which is the same as the total number of hosts. $\|\cdot\|_0$ is the ℓ_0 norm, which returns the number of nonzero elements in the vector. Then, the ℓ_0 norm of the vector $(IP_t - IP_0)$ corresponds to the number of IP addresses that have changed at the time t. Since the proposed asset criticality-aware MTD shuffles a host's IP and MAC addresses simultaneously with a single shuffling interval T_s, the obfuscation ratio of IP and MAC addresses is the same. Higher obfuscation ratio reflects the degree of uncertainty introduced by the MTD for attackers.

- *Mean attack success probability* ($\bar{P}_{AS}$) measures the average attack success probabilities of the entire critical assets in the network, and it is obtained by:

$$\bar{P}_{AS} = \frac{\sum_{i=1}^{n} P_R(h_i)}{n}, \tag{11}$$

where n is the number of critical assets and $P_R(h_i)$ refers to the probability of the host h_i's root privilege being successfully compromised. Lower value is more desirable.

4) Metrics for MTD Efficiency: We measure the MTD efficiency in terms of the following two metrics:

- *MAC shuffling delay:* This metric measures the additional overhead of the proposed MAC shuffling method in a scenario where one host sends a packet to the destination host, h_d. If no MAC address is found in the ARP cache table in the host, an ARP request packet is broadcast to the network. The ARP request packet is forwarded to an SDN controller, then the SDN controller generates the ARP response packet with h_d's *vMAC*, and directly sends it back to the source host, h_s. When h_s receives h_d's *vMAC*, it uses the MAC address in each frame it sends to h_d. The MAC shuffling delay includes ARP packet forwarding time, ARP packet flooding time, and flow table update time.
- *IP/Port shuffling delay:* This metric measures the additional overhead of the proposed IP/port shuffling method in a scenario where one host sends a DNS request packet to the DNS server to obtain h_d's IP address from the domain name in a given network. h_s attempts to obtain the IP address from its DNS cache table. If there is no matching entry in the DNS table, a DNS request packet will be forwarded to the SDN controller; and then the SDN controller sends the DNS request packet to the DNS server, which replies h_d's *rIP*. The SDN controller rewrites the reply packet with h_d's *vIP*. The IP shuffling delay includes DNS request time, DNS caching/response time, and flow table update time.

C. MTD Comparing Schemes

We compare the following MTD shuffling strategies in terms of the mean attack success probability metric:

- *Role criticality-based shuffling* (RC-based shuffling): This scheme uses a host's RC to decide the frequency of shuffling its network configuration. Hence, the probability of shuffling the host's network configuration can be computed by $F_{RC}(h_i)$ given by (7).
- *Influence criticality-based shuffling* (IC-based shuffling): This scheme uses a host's IC to shuffle its network configuration where its shuffling decision is made based on $F_s(h_i)$ given by (8).
- *Betweenness centrality-based shuffling*: This scheme uses a host's betweenness centrality value as a metric to determine which host to shuffle. The BC of a host h_i is given by the expression:

$$BC(h_i) = \sum_{s!=v!=d} \frac{\sigma_{sd}(h_i)}{\sigma_{sd}}, \tag{12}$$

where σ_{sd} is the total number of shortest paths from the source host h_s to the destination host h_d and $\sigma_{sd}(h_i)$ is the number of those paths that pass through the host h_i. The probability of shuffling a host's network configuration is estimated as:

$$F_{BC}(h_i) = \frac{BC(h_i)}{\sum_{h_j \in AG(h_i)} BC(h_j)}, \tag{13}$$

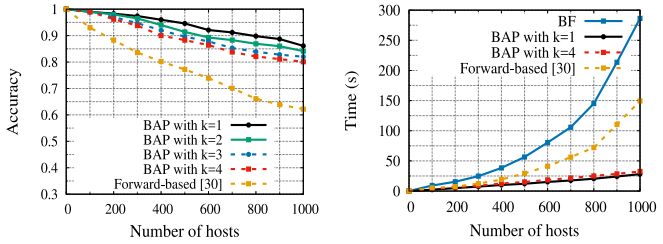
where $BC(h_i)$ denotes the betweenness centrality value of the host h_i, and $AG(h_i)$ is a set of hosts in an asset criticality-aware AG h_i involved.

- *OpenFlow Random Host Mutation (OF-RHM) [20]:* OF-RHM allows every host to have its optimal mutation rate based on the total size of allocated *vIP* address ranges to its subnet. If the hosts are in the same subnet, IP addresses of those hosts are shuffled based on a same interval, T_s. OF-RHM does not consider criticality of hosts in a network.

D. Results and Analysis for the BAP

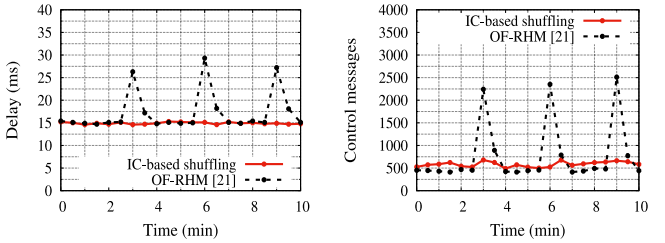
In this section, we compare the performance of the BAP with that of the BF (i.e., exhausted search) and forward-based attack path (FAP) prediction method [27] in terms of their prediction accuracy and computational complexity. In [27], the optimal attack path is discovered by the FindPath algorithm based on stack and graph coloring methods, the algorithm searches the most vulnerable attack path from the single source host to the final goal node in the access graph.

Fig. 5 (a) shows the effect of N (the total number of hosts) on the attack path prediction (P_{AP_k}) with respect to different numbers of attack paths (k) used. As in Fig. 5 (a), a reasonably high accuracy (i.e., over 90%) is observed when N is up to 400 hosts for all ks tested in this study. However, with more hosts, P_{AP_k} decreases because it naturally leads to higher chances to inaccurately select hosts for paths with more hosts in the network. Since the BF can capture an optimal attack path, we show the accuracy of the BAP with respect to the ground truth accuracy generated by the BF, as in (9). However, FAP shows the lower performance than our BAP because the BAP allows attackers to access different entry points (i.e., remotely accessible hosts) to penetrate into the system. Moreover, our proposed method finds the k most vulnerable attack paths rather than just one most vulnerable attack path. Fig. 5 (b) demonstrates



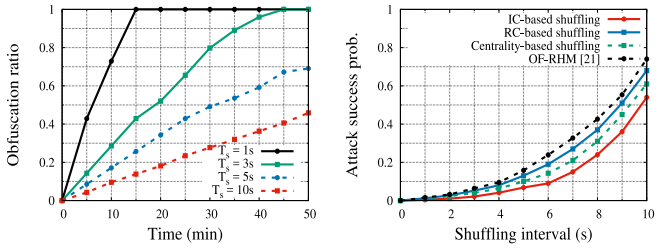
(a) Prediction accuracy of the BAP. (b) Complexity comparison of the BAP.

Fig. 5. Performance analysis of the BAP prediction.



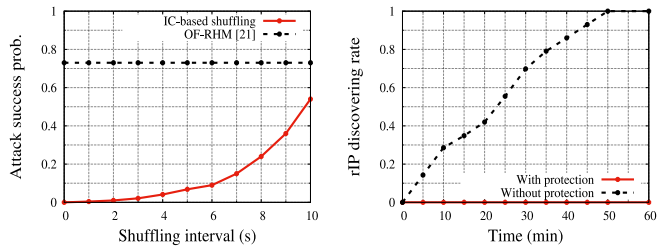
(a) End-to-end delay in data plane. (b) Control messages in control plane.

Fig. 6. Overhead analysis of the asset criticality-aware MTD.



(a) Effect of varying T_s. (b) Performance comparison in ASP.

Fig. 7. Performance analysis of asset criticality-aware MTD.



(a) ASP under forensic attacks. (b) Exposure rate of rIPs.

Fig. 8. Analysis of defending against forensic attacks.

comparative complexity analysis of the BAP, FAP, and BF in terms of the running time for attack path prediction. The BAP takes a backward-based greedy non-enumerative method. The clear out performance of the BAP over BF and FAP in the running time is observed as the network size increases while the BF and FAP show exponential growth in its running time.

E. Results and Analysis for the Asset Criticality-Aware MTD

In this section, we compare the proposed asset criticality-aware MTD and the existing counterparts in terms of the overhead (i.e., end-to-end delay and a number of control messages) and security improvement (i.e., obfuscation ratio and mean attack success probability).

Fig. 6 (a) shows the overhead in the data plane of an SDN-based network when the proposed shuffle-based MTD is applied. The OF-RHM method [20] shuffles the IP addresses of all hosts at once, so there is a lot of overhead upon every shuffling, resulting in delay in packet forwarding. However, since our proposed MTD changes network configurations by selecting one host every MTD interval, the proposed IC-based shuffling does not introduce high adverse effect on packet forwarding in the data plane. Note that the shuffling interval of OF-RHM is 3 min. while the number of shuffled hosts is the same as the proposed IC-based shuffling. Fig. 6 (b) shows the overhead in the control plane of SDN-based network when the proposed shuffle-based MTD is applied. As with the data plane overhead, OF-RHM rearranges the address of all hosts at once, which introduces a lot of control messages to the SDN controller. However, our proposed MTD constantly changes the host's address one by one, avoiding a large number of control messages being sent to the controller at the same time.

Fig. 7 (a) shows the effect of a different shuffling interval, T_s, on the obfuscation ratio over time under the proposed MTD when N is 300. For example, $T_s = 1$ means shuffling a selected host's MAC/IP address every second. It is observed that a shorter T_s leads to a higher obfuscation ratio with a sharper increase over time, and vice-versa. However, after about half of the hosts have been shuffled (recall a single host is selected to be shuffled upon each interval), the probability of previously shuffled hosts being shuffled increases, therefore, the overall obfuscation ratio does not increase linearly.

Fig. 7 (b) shows the performance of the compared MTD schemes considered in this work in terms of the attack success probability, $\hat{P}_{RC}(h_i)$. We compared the performance of the proposed IC-based shuffling, RC-based shuffling, with two other comparable schemes (centrality-based shuffling and OF-RHM scheme) as discussed in Section VII-C. Here, we assumed that attackers collect reconnaissance information in a certain period of time. During a reconnaissance attack interval, T_r, if the addresses of the attacker's neighbor host are shuffled, we assumed that the probability with vulnerability $v \in \mathcal{V}_r(h)$ where $h \in \mathcal{N}^{out}(h_i)$ is attacked by the attacker h_i decreases; because the attacker's information collected during the reconnaissance is invalidated. In other words, as the shuffling interval becomes shorter, the average attack success probability drops due to the confusion introduced by the MTD shuffling MAC/IP/port addresses. The result shows that the proposed IC-based shuffling shows the best performance to defend critical assets against the considered attacks while centrality-based shuffling shows better performance than both RC-based and OF-RHM shuffling schemes. The RC-based shuffling considering only the role of the host has less security effectiveness than the centrality-based shuffling scheme considering the configuration and connectivity information of the network. This implies that a node's network influence is critical because it may introduce severe damage to a network due to potential cascading failure leveraging its influence.

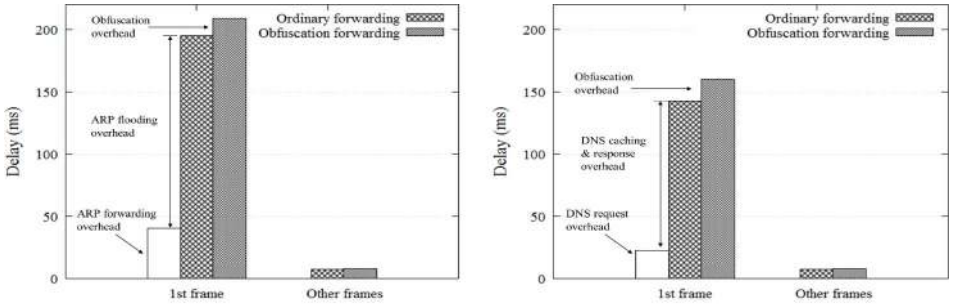


Fig. 9. Address shuffling delay measurements.

F. Results and Analysis for Defending Forensic Attacks

We also compared the performance of the proposed shuffle-based MTD and OF-RHM in terms of the attack success probability and the discovered *rIP* ratio in order to investigate the effect of the flow rule reconstruction attacks (i.e., forensic attacks). In Fig. 8 (a), when an attacker scans its neighbor hosts using nested ICMP packets, OF-RHM reveals the *rIP* information of neighboring hosts to the attacker. As a result, it fails to reduce the average attack success probability. On the other hand, the proposed method does not expose *rIP* information even if the attacker uses the various forensic methods, showing the decreasing attack success probability with shorter shuffling interval. Fig. 8 (b) shows the ratio of *rIP*'s that an attacker has successfully identified. If the SDN controller does not have any protection mechanism for nested ICMP probing packets, the attacker can find out *rIP*'s of all neighboring hosts in a few minutes. When the attacker sends a probing packet to one IP address every 15 seconds to a connected subnet, our SDN controller successfully removes *rIP* in the nested packet generated by the attacker's probing packets, hindering the attacker's reconnaissance activity.

G. Testbed Experiment-Based Results and Analysis

In this section, we will discuss our SDN-based experimental setup and the performance analysis in terms of the additional delay due to the IP and MAC address shuffling.

Fig. 9 (a) shows the MAC shuffling delay when the proposed asset criticality-aware MTD is deployed. For the first frame belonging to a new flow, if the SDN controller knows the path to its destination host, h_d, it forwards the ARP packets directly to the h_d rather than flooding them. In this case, the total ARP forwarding delay was approximately 41 ms. On the contrary, if the SDN controller does not know the path to h_d, it floods ARP packets to find h_d, taking approximately 195 ms, which includes ARP request/reply delay, flow table update (packet-in and packet-out messages) delay on each SDN switch, and data-forwarding delay. As shown in Fig. 9 (a), the proposed scheme incurred an additional 12 ms delay for the first frame due to *vMAC* assignment and the ARP injection process. For the next frames, the delay of the conventional forwarding includes only the data-forwarding delay. The proposed scheme requires MAC address modification of the switch to which h_d is connected; however, forwarding delays were nearly the same. This result indicates that the proposed asset criticality-aware MTD for MAC address shuffling can obfuscate the end-host information from attackers at an acceptable overhead.

Fig. 9 (b) shows IP shuffling delay upon the deployment of the asset criticality-aware MTD. In this scheme, the SDN controller receives a DNS request packet and forwards it to the DNS server, taking approximately 22 ms. The DNS server receiving the DNS request packet responds with a host's IP address with the corresponding domain name while the SDN controller changes the host's *rIP* to *vIP* in the DNS reply packet according to the sequence number of the packet. The proposed scheme incurred an additional 173 ms delay for the first frame due to *vIP* assignment and conversion. For the next frame, the delay of the proposed method includes only the data-forwarding and *vIP* modification delay.

The MAC/IP shuffling delays observed from the above shows an acceptable level of overhead as a reasonable tradeoff for obtaining enhanced security in increased obfuscation ratio and lowered attack success probability.

VIII. Conclusion

In this paper, we have proposed an asset criticality-aware MTD technique to effectively and efficiently obfuscate network information from potential attackers by leveraging the advanced SDN technology. In order to develop the asset criticality-aware MTD, we have also devised the backward attack path prediction as a lightweight solution to identify the most vulnerable attack paths that can be exploited by attackers. We have validated the out performance of the proposed asset criticality-aware MTD over its existing counterparts in terms of both its effectiveness and efficiency via extensive experiments in both simulation and real SDN testbeds. Our key findings prove that the proposed asset criticality-aware MTD provides scalable, adaptive security with an acceptable level of computation cost and communication delay.

Acknowledgment

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of CCDC ITC-PAC, CCDC-ARL, or the U.S. Government.

REFERENCES

[1] *Boston University Representative Internet Topology Generator (BRITE)*. Accessed: Apr. 21, 2020. [Online]. Available: https://www.cs.bu.edu/brite

[2] *Minenet*. Accessed: Apr. 21, 2020. [Online]. Available: http://mininet.org/

[3] *Open Source Network Operating (ONOS)*. Accessed: Apr. 21, 2020. [Online]. Available: https://onosproject.org/

[4] *Standard OUI List*. Accessed: Apr. 21, 2020. [Online]. Available: http://standards-oui.ieee.org/oui.txt

[5] S. Achleitner, T. F. La Porta, P. McDanial, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, "Deceiving network reconnaissance using SDN-based virtual topologies," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, no. 4, pp. 1098–1112, Dec. 2017.

[6] S. Achleitner, T. La Porta, T. Jaeger, and P. McDaniel, "Adversarial network forensics in software defined networking," in *Proc. ACM Symp. SDN Res. (SOSR)*, 2017, pp. 8–20.

[7] G. S. Bopche and B. M. Mehtre, "Graph similarity metrics for assessing temporal changes in attack surface of dynamic networks," *Comput. Security*, vol. 64, pp. 16–43, Jan. 2017.

[8] E. Bou-Harb, M. Debbabi, and C. Assi, "Cyber scanning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1496–1519, 3rd Quart., 2014.

[9] T. E. Carroll, M. Crouse, E. W. Fulp, and K. S. Berenhaut, "Analysis of network address shuffling as a moving target defense," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Sydney, NSW, Australia, 2014, pp. 701–706.

[10] V. Casola, A. De Benedictis, and M. Albanese, "A moving target defense approach for protecting resource-constrained distributed devices," in *Proc. IEEE Int. Conf. Inf. Reuse Integr. (IRI)*, San Francisco, CA, USA, 2013, pp. 22–29.

[11] J.-H. Cho *et al.*, "Toward proactive, adaptive defense: A survey on moving target defense," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 709–745, 4th Quart., 2020.

[12] A. Chowdhary, S. Pisharody, and D. Huang, "SDN based scalable MTD solution in cloud network," in *Proc. ACM Workshop Moving Target Defense (MTD)*, 2016, pp. 27–36.

[13] H. Cui, G. O. Karame, F. Klaedtke, and R. Bifulco, "On the fingerprinting of software-defined networks," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 10, pp. 2160–2173, Oct. 2016.

[14] X. Feng, Z. Zheng, D. Cansever, A. Swami, and P. Mohapatra, "A signaling game model for moving target defense," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Atlanta, GA, USA, 2017, pp. 1–9.

[15] J. Homer *et al.*, "Aggregating vulnerability metrics in enterprise networks using attack graphs," *J. Comput. Security*, vol. 21, no. 4, pp. 561–597, 2013.

[16] J. B. Hong and D. S. Kim, "Towards scalable security analysis using multi-layered security models," *J. Netw. Comput. Appl.*, vol. 75, pp. 156–168, Nov. 2016.

[17] J. B. Hong, S. Yoon, H. Lim, and D. S. Kim, "Optimal network reconfiguration for software defined networks using shuffle-based online MTD," in *Proc. IEEE 36th Symp. Rel. Distrib. Syst. (SRDS)*, Hong Kong, China, 2017, pp. 234–243.

[18] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *Proc. 22nd Annu. Netw. Distrib. Syst. Security Symp. (NDSS)*, San Diego, CA, USA, 2015, pp. 8–11.

[19] Y. Huang and A. K. Ghosh, "Introducing diversity and uncertainty to create moving attack surfaces for Web services," in *Proc. ACM Workshop Moving Target Defense (MTD)*, 2011, pp. 131–151.

[20] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: Transparent moving target defense using software defined networking," in *Proc. ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2012, pp. 127–132.

[21] P. Johnson, R. Lagerström, M. Ekstedt, and U. Franke, "Can the common vulnerability scoring system be trusted? a Bayesian analysis," *IEEE Trans. Depend. Secure Comput.*, vol. 15, no. 6, pp. 1002–1015, Nov./Dec. 2018.

[22] D. J. Leversage and E. J. Byres, "Estimating a system's mean time-to-compromise," *IEEE Security Privacy*, vol. 6, no. 1, pp. 52–60, Jan./Feb. 2008.

[23] Y. Li, R. Dai, and J. Zhang, "Morphing communications of cyber-physical systems towards moving-target defense," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Sydney, NSW, Australia, 2014, pp. 592–598.

[24] W. Lingyu, T. Islam, T. Long, A. Singhal, and S. Jajdia, "An attack graph-based probabilistic security metric," in *Proc. Annu. Working Conf. Data Appl. Security (DBSec)*, 2008, pp. 283–296.

[25] Y. Liu and H. Man, "Network vulnerability assessment using Bayesian networks," in *Proc. Data Mining Intrusion Detect. Inf. Assurance Data Netw. Security*, 2005, pp. 61–72.

[26] D. C. MacFarland and C. A. Shue, "The SDN shuffle: Creating a moving-target defense using host-based software-defined networking," in *Proc. 2nd ACM Workshop Moving Target Defense*, 2015, pp. 37–41.

[27] S. Malhotra, S. Bhattacharya, and S. K. Ghosh, "A vulnerability and exploit independent approach for attack path prediction," in *Proc. 8th IEEE Int. Conf. Comput. Inf. Technol. Workshops*, Sydney, QLD, Australia, 2008, pp. 282–287.

[28] L. Muñoz-González, D. Sgandurra, A. Paudice, and E. C. Lupu, "Efficient attack graph analysis through approximate inference," *ACM Trans. Privacy Security*, vol. 20, no. 3, p. 10, 2017.

[29] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs, "Efficient minimum-cost network hardening via exploit dependency graphs," in *Proc. Annu. Comput. Security Appl. Conf. (ACSAC)*, Las Vegas, NV, USA, USA, 2003, pp. 86–95.

[30] W. Peng, F. Li, C. Huang, and X. Zou, "A moving-target defense strategy for cloud-based services with heterogeneous and dynamic attack surfaces," in *Proc. IEEE Int. Conf. Commun.*, Sydney, NSW, Australia, 2014, pp. 804–809.

[31] C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis," in *Proc. ACM New Security Paradigms Workshop (NSPW)*, 1998, pp. 71–79.

[32] R. Sawilla and D. Skillicom, "Partial cuts in attack graphs for cost effective network defence," in *Proc. IEEE Conf. Technol. Homeland Security (HST)*, Waltham, MA, USA, 2012, pp. 291–297.

[33] D. P. Sharma, D. S. Kim, S. Yoon, H. Lim, J. Cho, and T. J. Moore, "FRVM: Flexible random virtual IP multiplexing in software-defined networks," in *Proc. 17th IEEE Int. Conf. Trust Security Privacy Comput. Commun. (TrustCom)*, New York, NY, USA, 2018, pp. 579–587.

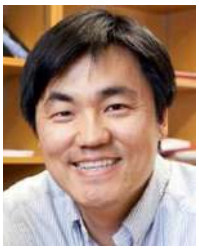
[34] M. Taguinod, A. Doupé, Z. Zhao, and G. Ahn, "Toward a moving target defense for Web applications," in *Proc. Int. Conf. Inf. Reuse Integr. (IRI)*, San Francisco, CA, USA, 2015, pp. 510–517.

[35] S. Vikram, C. Yang, and G. Gu, "NOMAD: Towards non-intrusive moving-target defense against Web bots," in *Proc. IEEE Conf. Commun. Netw. Security (CNS)*, National Harbor, MD, USA, 2013, pp. 55–63.

[36] Y. Wang, Q. Chen, J. Yi, and J. Guo, "U-TRI: Unlinkability through random identifier for SDN network," in *Proc. ACM Workshop Moving Target Defense (MTD)*, 2017, pp. 3–15.

Seunghyun Yoon (Student Member, IEEE) received the B.S. degree from the School of Computer Science and Electrical Engineering, Handong Global University, Pohang, South Korea, in 2016. He is currently pursuing the Ph.D. degree with the School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology, Gwangju, South Korea. From August 2019 to February 2020, He was a Visiting Scholar with the Department of Computer Science, Virginia Tech. His research interests include software defined networking, resource allocation, deep reinforcement learning, cybersecurity for various network domains, and moving target defense.

Jin-Hee Cho (Senior Member, IEEE) received the M.S. and Ph.D. degrees in computer science from the Virginia Tech in 2004 and 2008, respectively, where she is currently an Associate Professor with the Department of Computer Science since August 2018, and the Director of the Trustworthy Cyberspace Laboratory. She worked as a Computer Scientist with the U.S. Army Research Laboratory, Adelphi, MD, USA, since 2009. She has published over 120 peer-reviewed technical papers in leading journals and conferences in the areas of trust management, cybersecurity, metrics and measurements, network performance analysis, resource allocation, agent-based modeling, uncertainty reasoning and analysis, information fusion/credibility, and social network analysis. She received the Best Paper Awards in IEEE TrustCom'2009, BRIMS'2013, IEEE GLOBECOM'2017, 2017 ARL's Publication Award, and IEEE CogSima 2018. She is a winner of the 2015 IEEE Communications Society William R. Bennett Prize in the field of communications networking. In 2016, she was selected for the 2013 Presidential Early Career Award for Scientists and Engineers, which is the highest honor bestowed by the U.S. Government on outstanding scientists and engineers in the early stages of their independent research careers. She is a Senior Member of ACM.



Dong Seong Kim (Senior Member, IEEE) received the Ph.D. degree from Korea Aerospace University in February 2008. He is currently an Associate Professor of cybersecurity with the University of Queensland, Australia, since January 2019. He was a Senior Lecturer/Lecturer of cybersecurity with the University of Canterbury from August 2011 to December 2018. He was a Visiting Scholar with the University of Maryland, College Park, in 2007. From June 2008 to July 2011, he was a Postdoctoral Researcher with Duke University. His research interests are in automated cybersecurity modeling and analysis for the Internet of Things, cloud computing, and moving target defense. He was the General Co-Chair of ACISP2019 and the General Chair of IEEE PRDC 2017. He served as a Program Co-Chair of IEEE TrustCom2019, IEEE ICIOT2019, ATIS2017, GraMsec2015, and IEEE DASC2015, and a Program Committee Member of international conferences, including IFIP/IEEE DSN, ISSRE, SRDS, and ICC CISS.

Terrence J. Moore (Member, IEEE) received the B.S. and M.A. degrees in mathematics from American University in 1998 and 2000, respectively, and the Ph.D. degree in mathematics from the University of Maryland, College Park, in 2010. He is currently a Researcher with the Network Science Division, U.S. Army Research Laboratory. His research interests include sampling theory, constrained statistical inference, stochastic optimization, network security, geometric and topological applications in networks, and network science.

Frederica Free-Nelson is a Researcher and a Program Lead with U.S. Army Research Laboratory (ARL), Adelphi, MD, USA, where she leads research on machine learning and intrusion detection methods and techniques to promote cyber resilience and foster research on autonomous active cyber defense. She manages and negotiates the Research and Project Agreements for ARL between the network security branch and Academia or International Organizations. She is the lead for the Robust Low-Level Cyber-Attack Resilience for Military Defense Program working in collaboration with Army Tank Automotive Research, Development and Engineering Center, Office of Naval Research, and Air force Research Laboratory to build a cohesive in-vehicular resilient system for defense against sophisticated enemy malware that strives to blend in with normal system activities. She has over 20 years' combined experience in cybersecurity research, software engineering, and program management within the DoD and other federal services to include the Federal Bureau of Investigation and the Department of Justice. She has expertise in leading projects to success from conception to execution and delivery/transfer. She currently serves as the Chairperson for the International Science Technology (IST-163) Panel–NATO Science and Technology Organization on the topic of deep machine learning for military cyber defense. She is a participant in the Army Education Outreach Program as an Ambassador and a Virtual Judge for the eCybermission Program.

Hyuk Lim (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from the School of Electrical Engineering and Computer Science, Seoul National University, Seoul, South Korea, in 1996, 1998, and 2003, respectively. From 2003 to 2006, he was a Postdoctoral Research Associate with the Department of Computer Science, University of Illinois at Urbana–Champaign, Champaign, IL, USA. He is currently a Full Professor with the AI Graduate School and the School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology, Gwangju, South Korea. His research interests include network system design, optimization, and performance evaluation.