Deceiving Network Reconnaissance Using SDN-Based Virtual Topologies

Stefan Achleitner, Thomas F. La Porta, *Fellow, IEEE*, Patrick McDaniel, *Fellow, IEEE*, Shridatt Sugrim, Srikanth V. Krishnamurthy, *Fellow, IEEE*, and Ritu Chadha, *Senior Member, IEEE*

***Abstract*—Advanced targeted cyber attacks often rely on reconnaissance missions to gather information about potential targets, their characteristics and location to identify vulnerabilities in a networked environment. Advanced network scanning techniques are often used for this purpose and are automatically executed by malware infected hosts. In this paper, we formally define network deception to defend reconnaissance and develop a reconnaissance deception system, which is based on software defined networking, to achieve deception by simulating virtual topologies. Our system thwarts network reconnaissance by delaying the scanning techniques of adversaries and invalidating their collected information, while limiting the performance impact on benign network traffic. By simulating the topological as well as physical characteristics of networks, we introduce a system which deceives malicious network discovery and reconnaissance techniques with virtual information, while limiting the information an attacker is able to harvest from the true underlying system. This approach shows a novel defense technique against adversarial reconnaissance missions which are required for targeted cyber attacks such as advanced persistent threats in highly connected environments. The defense steps of our system aim to invalidate an attackers information, delay the process of finding vulnerable hosts and identify the source of adversarial reconnaissance within a network.**

***Index Terms*—Software-defined networks, security services, security management.**

I. Introduction

THE STATIC nature of computer networks enables adversaries to perform network reconnaissance and identify vulnerabilities which can be exploited by advanced targeted cyber attacks. Network reconnaissance missions provide a tactical advantage for attackers on cyber infrastructure by identifying potential targets and their vulnerabilities, as discussed in [18], [26], [29], [34], [36], and [43]. In particular, insider adversaries are probing networked environments to identify hosts and open ports and map their topology to find known and zero-day vulnerabilities to perform further attack maneuvers.

Highly effective scanning strategies for network reconnaissance are known and have been analyzed in the context of computer worms such as in [20], [39], and [40], and are precursors to a high percentage of cyber attacks. In particular, Panjwani *et al.* [30] conclude that up to 70% of attacks are preceded by an adversarial scanning activity. Such reconnaissance techniques are highly effective by exploiting certain features in networks, such as the uneven distribution of hosts in the address space or the configuration and composition of network topologies, to increase their efficiency of identifying potential targets. Sophisticated targeted attacks, such as APT [1], [13], [36], depend on fingerprinting of organizational networks to identify hosts and vulnerabilities necessary for the development of a battle plan and the execution of further attack maneuvers.

In this paper, we aim to deceive such malicious reconnaissance and discovery techniques by showing a virtual topological view of a networked system which hides its true underlying network and its potential vulnerabilities that can be exploited by attackers. The simulation of a virtual network view invalidates the set of information an attacker collects about a networked system and achieves the goal of significantly delaying the rate of identifying vulnerable hosts. This procedure gains additional time that can be used to identify a malicious scanner and isolate it from the network.

Techniques such as social engineering, zero-day vulnerabilities, drive by downloads or manual infection [1], [12], [13], [36] of hosts inside organizational networks are serious security threats which can cause significant damage and are hard to detect. In our threat model we consider such adversaries that are present inside a network and have at least one host infected with some sort of malware. Our proposed defense system addresses adversarial reconnaissance and discovery activity, which presents the third stage of an advanced cyber attack as defined by Symantec [12]. To address threats caused by reconnaissance missions of insider attackers, we develop a formal deception approach which identifies certain features of a networked system that are modified by our Reconnaissance Deception System (RDS) to invalidate the set of information an attacker collects about the system. The challenge in the design of such a system is to guarantee the network functionality and minimize the performance impact for legitimate traffic, while maximizing the effectiveness of the defense strategy. A crucial

Manuscript received February 28, 2017; revised June 5, 2017; accepted June 29, 2017. Date of publication July 7, 2017; date of current version December 8, 2017. The effort described in this article was sponsored by the U.S. Army Research Laboratory Cyber Security Collaborative Research Alliance under Cooperative Agreement W911NF-13-2-0045. The associate editor coordinating the review of this paper and approving it for publication was F. De Turck. *(Corresponding author: Stefan Achleitner.)*

S. Achleitner, T. F. La Porta, and P. McDaniel are with Pennsylvania State University, University Park, PA 16802 USA (e-mail: stefan.achleitner@cse.psu.edu).

S. Sugrim and R. Chadha are with Vencore Labs, Basking Ridge, NJ 07920 USA.

S. V. Krishnamurthy is with the University of California at Riverside, Riverside, CA 92521 USA.

Digital Object Identifier 10.1109/TNSM.2017.2724239

part of our proposed defense solution is the composition of virtual network views, the placement of hosts and honeypots and the simulation of consistent network characteristics in virtual topologies to delay attackers from identifying real and vulnerable hosts in the underlying network.

In our system, a different virtual network view can be assigned to every host or to specific hosts. This makes our defense approach independent of the source of malicious network scans, which we assume is not known initially. The implementation of our system ensures a clear separation between virtual network views, which are only visible to the assigned hosts, and the remaining underlying network.

As our main contribution, we develop a formal deception approach and use this definition as the basis to design and implement a RDS (Reconnaissance Deception System). Further, we evaluate its goals of deceiving and detecting insider adversaries, while limiting the cost to achieve increased security. We publish an open-source proof of concept prototype of our system at [10]. In the evaluations we demonstrate that our system increases the duration required to identify vulnerable endpoints in a network up to a factor of 115 while limiting the performance impact on legitimate traffic. In more detail, the following is a summary of our contributions:

- ***Definition of a reconnaissance deception approach:*** We identify an information set collected by adversaries during reconnaissance missions. To minimize the usefulness of this information for attackers, we define a set of network features that have to be modified to deceive the reconnaissance goal of an adversary. We discuss different approaches and strategies for the generation of virtual network topologies in which vulnerable entities are strategically placed to minimize their detection likelihood by adversaries.
- ***Design and implementation of RDS:*** To realize and evaluate the defined deception goals, we implement a research prototype of RDS, based on SDN, to simulate complete virtual network topologies including its physical network characteristics. The combined functionality of our *deception server*, *SDN controller*, *honeypot server*, *delay handler* and *virtual topology generator* achieves thorough network deception, while maintaining the full network functionality for legitimate traffic.
- ***Evaluation of defense effectiveness and performance:*** By executing malicious scanning techniques on simulated virtual topologies in our test environment we show that our system is able to significantly delay the detection of vulnerable hosts up to a factor of 115. We also demonstrate that our solution keeps the performance overhead on legitimate traffic within defined limits.
- ***Identification of infected hosts in a network:*** Our SDN controller implementation dynamically analyzes SDN flow rule statistics and is able to identify scanning activity based on the distribution of network traffic on specific flow rules. We show in our experiments that our system is able to identify malicious scanners before an attacker is able to find vulnerable hosts in a network.

A preliminary version of our work on cyber deception appeared in [14]. In this paper we extend the system architecture of RDS to simulate the physical characteristics of a virtual network topology, which is crucial for the deception of advanced attackers.

II. Related Work

In recent publications [18], [25], [26], [41], the authors introduce systems that performs dynamic address space randomization based on Software Defined Networking (SDN) technologies to defend against adversarial scanners, such as worms. Although being an effective defense approach, these systems suffer from high overheads which we avoid in our solution and are not able to dynamically detect the source of scanning traffic that our solution achieves by simulating entire virtual topologies. In particular, the introduced system in [25] relies on DNS queries which have to be performed for every new connection to a legitimate host that is established within a new randomization interval (usually 1-5 seconds). In addition to sending a DNS request in the proposed system, the DNS reply message is intercepted by the SDN controller and rewritten to match the address randomization strategy. To evaluate their system performance, we implemented the proposed DNS protocol which requires 51.6ms on average to resolve a name which has to be done every 1-5 seconds if a new connection to a host is established. Our system in comparison only takes 16.7ms on average to resolve a name. In addition, our RDS has to perform name resolution only upon the deployment of a new virtual view which is done every few hours with the assignment of a new DHCP lease.

Sun and Sun [45] propose a defense system based on IP address randomization and the placement of decoys. They also introduce a network connection migration mechanism to seamlessly move existing connections between legitimate users when the IP addresses of servers change. In contrast to our system, which targets internal adversaries who are present in an enterprise network, [45] focuses on scanners such as ZMap [17] originating from the Internet.

Trassare *et al.* [37] propose techniques to deceive network reconnaissance focused on mapping a topology by using the standard *traceroute* function. The authors especially focus on the defense of critical routers and links in a network topology.

Chiang *et al.* [16] and Robertson *et al.* [33] discuss an approach to build a cyber deception system. An overview of the design of such a system is presented, but the authors do not provide a detailed formulation of network reconnaissance or evaluate specific attack strategies depending on reconnaissance missions.

In comparison to existing network deception systems, we demonstrate that RDS is able to identify the source of malicious scanning traffic by analyzing the flow statistics of SDN flow rules used to simulate virtual topologies. Current defense approaches do not consider such techniques to identify and isolate the source of adversarial scanning traffic. This gives our proposed system an advantage over current approaches to effectively defend and identify an adversary performing reconnaissance. A deception feature which is unique to RDS, and was not considered in previous publications, is the simulation of consistent physical network features as we introduce in this

work. Without simulating consistent physical characteristics of deception mechanisms advanced attackers are able to deduce that they are deceived and use such knowledge as a counter maneuver as we demonstrate in Section V-E.

Honeypots are an essential component in our RDS for the simulation of virtual network views. We use honeypots as traps and decoys in virtual networks to detect malicious scanning traffic and identify adversarial hosts. For the usage and configuration of honeypots we follow best practices as defined in well cited publications such as [32] and [44].

We consider a number of well cited publications [28], [39], [40] for the analysis and implementation of malicious network scanning strategies which have been initially observed in computer worms. To evaluate our RDS we implemented adversarial scanning strategies as discussed in these papers.

III. Problem Definition

In this section, we formally define insider reconnaissance and describe the assumed threat model.

A. Insider Reconnaissance

Adversarial reconnaissance is geared to gather information about potential targets in networked systems. Scanning strategies that perform active probing of addresses in a network to identify online hosts and collect information about them and their connectivities are typically used for this purpose. One can represent the information gleaned via reconnaissance as a set T, where:

$T = \{AV = \textit{Addresses of potentially vulnerable hosts}, NS = \textit{Network size}, ST = \textit{System topology}, PNC = \textit{Physical network characteristics}\}$.

Insider attackers (e.g., malware programs) typically scan a networked system at very low rates in order to stay undetected. Once an information set T is obtained, it can be further observed for exploitable targets, such as open ports at hosts or network services with known vulnerabilities. The attacker's goal is to increase the cardinality of T and execute parallel kill chains on the multiple targets identified, to achieve the highest rate of success.

To prevent such reconnaissance, one defense approach is to minimize the useful information an attacker can collect in T. Our RDS framework seeks to populate the set of T with *fake* information so that an attacker is not able to determine what information in T is virtual and what is real.

B. Threat Model

We consider insider adversaries who have placed themselves in the network (on one or more hosts), using techniques such as social engineering, exploiting zero-day vulnerabilities, via drive by downloads or by manual infection [1], [12], [13], [15], [36], [46]. Our defense approach is based on measuring the reconnaissance information a strong insider is able to gather in the information set T. Based on that, RDS aims to minimize the usefulness of T by transforming it into a different set T'. Planning of sophisticated targeted attacks (e.g., Advanced Persistent Threats or APTs) requires a high granularity of insider information T which our solution prevents with providing T' to an adversary.

We assume that the location of an attacker inside the network is unknown initially. With RDS, a different virtual network view can be assigned to every host in a network to make our defense approach independent of the source of malicious scanning traffic.

For securing the SDN controller from attacks, numerous solutions have recently been published (e.g., [27], [31], [35], and [46]), which can be deployed to protect the SDN control infrastructure from being compromised. For the purposes of our analysis, we do not consider attackers penetrating the SDN controller or outside scanners which are addressed by mechanisms such as Firewalls or Intrusion Detection Systems, and have inherently less information than insiders.

C. Reconnaissance Deception

Our key idea is to map a set of network features NF, to a different set NF'; this new set mis-informs the attacker and provides a set T' which is populated with false information. RDS achieves this by simulating a virtual network which is the only view exposed to an attacker performing reconnaissance. An attacker collects information to construct a set T as defined previously. With RDS, the adversary will populate T with information with regards to the simulated virtual network instead of the real underlying network. The composition of a virtual network is critical to ensure that the information collected by an attacker is useless for further attack planning. Let the set of network features that RDS wants to hide be:

$NF = \{TL = \textit{Topological location of hosts}, NH = \textit{Number of hosts}, AH = \textit{Addresses of hosts}, CH = \textit{Connectivity between hosts}, LB = \textit{Link bandwidth}, HD = \textit{Host delay}\}$.

These network features NF, are transformed into a new (virtual) set of features NF'. With NF', the attacker generates a new set T' that is quite different from T, i.e., T is now transformed into T': Stated formally,

$$\begin{aligned} NF' &= \{TL', NH', AH', CH', LB', HD'\} \rightarrow T' \\ &= \{AV', NS', ST', PNC'\} \end{aligned}$$

Towards achieving the above transformation, RDS performs a set of agile maneuvers. These maneuvers can be seen as a function, $T' = f(NF')$, which result in T'. Performing these transformations will significantly delay adversarial scanners, invalidate any collected information by the adversary and allows the quick identification of infected hosts. These maneuvers are:

- ***Dynamic address translation*** *($AV' = f(AH')$):* Our system performs on-the-fly packet header rewriting to hide the real host addresses and make the overall address space of a network appear larger. The translation of the real underlying network's address space into a significantly larger virtual address space increases the search space for adversarial scanners. Since the addresses of potentially vulnerable hosts are changing with every

assignment of a new virtual network view, as we discuss in more detail in Section IV-B, previously collected addresses of potential targets are invalidated.

- ***Route mutation*** *($ST' = f(TL', CH')$):* We introduce virtual routers with our deception system and simulate virtual paths spanning multiple hops from a source to a destination host. This maneuver enables RDS to alter the topology of different network views so that a scanner is not able to draw conclusions about the real network topology.
- ***Vulnerable host placement*** *($ST' = f(TL', CH')$):* Dynamic address translation in combination with route mutation allows us to simulate virtual topologies consisting of multiple subnets. By placing vulnerable hosts in virtual subnets according to different strategies we discuss in Section IV-B, RDS aims to increase the duration a malicious scanner takes to identify them. We define vulnerable hosts as real hosts of the underlying network which could potentially be corrupted by a cyber attack.
- ***Honeypot placement*** *($NS' = f(TL', AH', NH')$):* To enlarge virtual networks we place honeypots which act as decoys and are closely monitored to detect malicious activity. By placing honeypots, we increase the number of potential targets for an attacker. Hereby, dynamic address translation allows RDS to make a single honeypot server appear as a target at many addresses and therefore significantly increases the search space for malicious scanners. For the setup and generation of honeypots, we follow best practices as introduced in previous publications such as [44].
- ***Delay and bandwidth adjustment*** *($PNC' = f(LB', HD')$):* Attackers collecting data from a network can use statistical analysis tools or network tomography techniques [22], [23], [38] to determine certain topological characteristics or differentiate a real target from a honeypot. By using specific queuing policies and adjusting the delay to make observed network characteristics consistent, our RDS system aims to deceive adversarial probing techniques which analyze physical characteristics of a topology.
- ***Dynamic detection of malicious flows:*** By evaluating the statistics of every flow rule in SDN switches, our deception system is able to detect malicious flows which try to establish connections to honeypots or protected hosts. We demonstrate in Section V-F, that RDS is able to detect the location of adversarial scanners before they are able to identify any vulnerable hosts in a virtual network view.

To present a simple example of the defense approach we achieve with RDS, we show a virtual topology in Figure 1 (top), which is deployed to be seen from the perspective of node 3 and significantly differs from the real network (bottom). We refer to node 3 as the *view node*. The view node is the node in the network to which a specific virtual network view is given. The design of our system considers the assignment of different network views to all or specific hosts in a network, making this defense approach independent of the source of malicious scans, which we assume is not known initially.

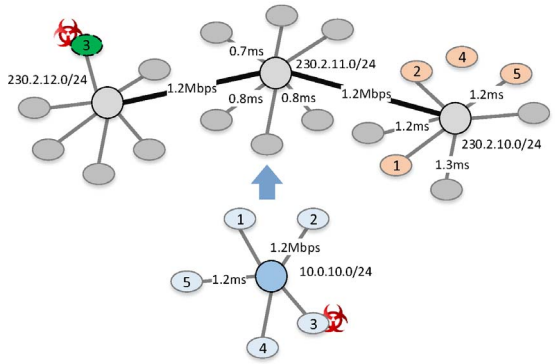

Fig. 1. An example of network deception via the projection of a virtual network which places critical resources in a way to deceive adversaries.

Hosts 1, 2 and 5 are seen as vulnerable resources that have to be protected. In case node 3 is performing an adversarial network scan the placement of the vulnerable resources, is critical. The location of vulnerable hosts, relative to the position of an attacker, impacts their detection time, since malicious scanning strategies often depend on locality as we discuss in Section V-C. Because host 4 in the virtual network topology is not supposed to be contacted by host 3, the link connecting it to the rest of the network is deactivated. The remaining nodes in the virtual topology in Figure 1 are honeypots and act as traps for potential adversaries.

Certain nodes in a network depend on the knowledge of the real underlying network topology. Examples are scheduling or load balancing algorithms that often choose geographically close nodes for load distribution, or applications that perform automatic discovery for legitimate purposes of resources and services in a network. A deception system, such as ours, would interfere with legitimate network discovery applications as listed. Therefore, nodes that require a real view of the network topology have to be identified by a network operator and should not have virtual network views assigned that would deceive information collected by legitimate network discovery. To ensure this, we emphasize a clear separation between virtual network views and the real underlying topology in the implementation and design of our system.

IV. System Design

In this section we introduce the implementation and design of our Reconnaissance Deception System in detail. The core parts of RDS consist of a sophisticated system of SDN flow rules generated by our SDN controller, which cooperates with a deception server to manipulate the network traffic in a way such that a network appears different than it actually is.

A. System Architecture Overview

Our system comprises five main components, a *SDN controller* responsible for dynamic generation and management of the flow rules to steer and control the network traffic, a *deception server* to manipulate the network traffic and simulate certain virtual network resources considering a specific user policy, a *honeypot server* to simulate virtual honeypot nodes,

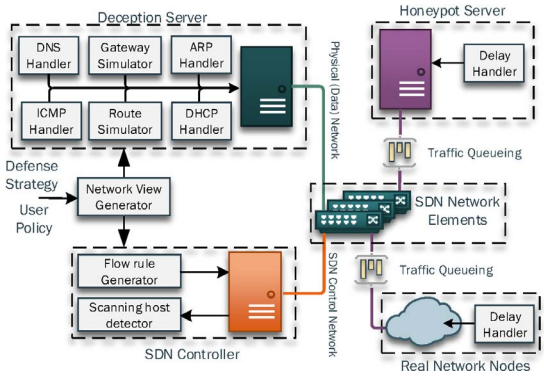

Fig. 2. Architecture of RDS.

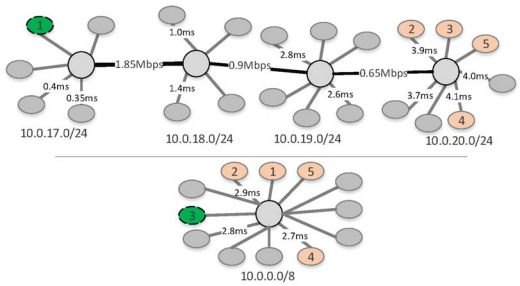

Fig. 3. Virtual Network view from the perspective of node 1 (top) and from the perspective of node 3 (bottom).

a *virtual network view generator* which provides a description of the virtual network components and their connectivity and *traffic queuing* to control link delay and bandwidth.

When a packet arrives at a SDN switch connected to our system the controller deploys a flow rule in accordance with the virtual network view description which either forwards the packet to the deception server, or sends the packet to its destination host after tags are added to the packet and addresses are translated. If a packet is sent to the deception server, a reply packet is crafted in accordance to the network view and sent back to the source, possible artificial delays are considered. If a packet is forwarded to a real end host or honeypot, artificial delay is added to the reply packet to guarantee consistency which is performed by implementing our proposed queuing disciplines at the end hosts. For the implementation of our deception server and SDN controller we follow best practices to meet security standards.

In our RDS the deception server is responsible to handle certain packets and generate reply messages according to the specified virtual network view. In a large network the deception server can turn into a bottleneck since requests of many end-hosts need to be handled. To maintain scalability of our system, the deception server can be replicated so that each instance of the server handles a specific subnet or a share of the IP address space of the underlying network topology. Based on a packet's source address, the deception server is able to differentiate between different deployed virtual network views and generate reply packets accordingly. Such a distribution can be managed by the SDN controller to forward packets to their dedicated deception server.

Our system is implemented in *Python*. We use the POX framework [8] to implement our SDN controller and the Scapy framework [11] to implement our deception server. We tested our implementation in Mininet [4] which is the current state-of-the-art SDN network emulator. Our SDN environment supports OpenFlow [6] version 1.3, which serves as the protocol for the communication between SDN controller and SDN switch. In Figure 2 we show an architectural overview of our RDS system.

B. Virtual Network View Generator

As we discuss in Section III-C, a virtual network view is a topology that is exposed to a client and is significantly different from the actual underlying network topology. A virtual network view is specified by a machine readable description of real hosts, honeypots and network paths between such endpoints as defined in the set of features *NF*. Generating a virtual network view depends on the number of simulated subnets S and the number of hosts (real hosts and honeypots) H per subnet. For the complexity of our generation algorithm we can define an upper bound of $O(S \cdot H)$. Since the number of honeypots per subnet is determined randomly between an upper and a lower bound and the number of real hosts in a virtual subnet is typically significantly smaller than the number of honeypots, H can be seen as a constant factor. Therefore we can assume a linearly increasing computation time of a virtual network view description with a growing number of subnets S. Measurements on a machine with an Intel i7 processor show a computation time for a virtual network view with 25 subnets of 200-250ms on average.

In the description file of a virtual network view it can also be specified if a real host of the underlying network is visible in the virtual network view or not. This feature makes our system also function as a distributed access control list, since different subsets of real hosts can be shown in a virtual view. Each virtual network view is associated with a DHCP lease that is offered to a host in order to connect to the network. The assignment of a DHCP lease triggers the generation and deployment of a virtual network view in our system. Here, the deployment time consists of the view computation time as discussed above, as well as the deployment time of SDN flow rules, which is done on demand in a re-active way and is in the order of milliseconds depending on the hardware switch. The update interval of a virtual network view is therefore dependent on the duration of the assigned DHCP lease, which typically ranges from a few hours to multiple days. The overall generation and deployment time, which can be assumed to be $\sim$1 second does therefore not present a significant overhead in our system since it only occurs with the assignment of a new DHCP lease.

As an example of virtual topologies, consider the network views shown in Figure 3. The top virtual network view shows a topology from the perspective of node 1. Besides honeypots (unnumbered nodes), the real nodes 2, 3, 4 and 5 appear to be in a different subnet which is three hops away. The real nodes can be considered as vulnerable and therefore have to be protected from malicious scans. The bottom part of Figure 3 shows the virtual network from the perspective of node 3.

Instead of simulating multiple virtual subnets, our RDS places all nodes, including vulnerable nodes and honeypots in one subnet with a big IP address space of 10.0.0.0/8. Both virtual topologies show valid defense strategies that can be simulated with RDS by generating a particular virtual view description.

The specific description of a virtual view is generated by our virtual network view generator. As input, the list of real hosts visible in the virtual view, the addresses for the honeypot and deception servers, the virtual IP address space, the number of subnets and the placement strategy for real hosts have to be provided. Our generator starts by assigning a random number of honeypots within an upper and a lower bound to each virtual subnet and places the real hosts in the virtual subnets according to the specified placement strategy. Placing a number of honeypots in each subnet of a virtual network view enables our system to detect the location of adversarial scanners by closely monitoring the traffic to honeypots which is done by the SDN controller. Therefore, honeypots in RDS primarily act as traps to detect unwanted traffic in the network. By choosing a strategy which randomly places honeypots in the address space of a virtual network, we aim to reveal scanning strategies typically used for network reconnaissance before attackers are able to detect vulnerabilities in our system as we evaluate in Section V-F. We plan to investigate additional honeypot placement strategies to further reduce the required detection time of the source of scanning traffic in our future work.

For the placement of vulnerable hosts we have to consider if the module to simulate the physical network characteristics, such as bandwidth and delay, is activated. If physical network characteristics are simulated, hosts have to be placed under consideration of their real hop distance and round trip time to the view node as we discuss in detail in Section IV-E. If our RDS is simulating virtual topologies without the physical characteristics, the placement strategies of real hosts include (i) random placement, (ii) placement with high address distance from the view node, and (iii) a placement to create a uniform distribution of nodes across the simulated IP address space.

The virtual IP addresses used in a view are assigned randomly within the provided address space to each real host and honeypot, considering their placement in the virtual topology. This procedure follows the deception principle introduced in Section III-C to transform the set of network features NF into NF' as summarized below:

- IPv4 address space of a virtual network $AH \rightarrow AH'$
- List of real hosts that are visible in a virtual network $CH \rightarrow CH'$
- Placement strategy of real hosts in a virtual network $AH \rightarrow AH'$, $NH \rightarrow NH'$, $TL \rightarrow TL'$
- Number of simulated subnets in a virtual network $AH \rightarrow AH'$, $NH \rightarrow NH'$, $TL \rightarrow TL'$
- Number of honeypots per subnet $NH \rightarrow NH'$, $TL \rightarrow TL'$
- Host delay and link bandwidth $HD \rightarrow HD'$, $LB \rightarrow LB'$

To simulate physical network characteristics in a virtual network view, our RDS uses queuing and introduces artificial delay as we discuss in detail in Section IV-E. Here, our system aims to limit the performance overhead added to legitimate traffic. Delay and bandwidth characteristics for connections to honeypots are correlated with the measured characteristics of real nodes. For an attacker collecting physical network characteristics, measurements from honeypots will appear indistinguishable from those observed from real hosts. This will prevent attackers from differentiating honeypots from real hosts based on their physical network characteristics. Figure 3 shows a simplified example of the adjusted delay and link bandwidths in virtual network views.

To summarize, a machine readable virtual topology description contains the following information:

- Virtual network view node specification
- Port and address information of the deception servers
- Real addresses of visible real hosts
- Real addresses of honeypot servers
- Deceptive IP addresses of real hosts and honeypots, as they appear in the virtual topology
- Virtual path information to real hosts and honeypots
- Configuration of traffic queuing and delay adjustment

The generation and deployment of a network view has a delay in the order of a second and can therefore be performed on request as an agile defense maneuver.

C. Software Defined Networking Controller

The main tasks of the SDN controller component is to dynamically generate flow rules which are pushed to an SDN switch to steer and control the network traffic. An additional task of the SDN controller is to analyze the flow statistics of the switch rules and identify malicious behavior of the network endpoints. To steer and control the network traffic, the SDN controller dynamically generates rules upon the arrival of a packet that does not match any current flow rule in the SDN switch. For reasons of system scalability we chose a re-active rule generation approach versus a pro-active approach which we will explain in more detail in Section IV-C1. Our SDN controller constructs the following flow rules based on the provided virtual network view description:

- ***Forward ARP requests to deception server:*** Handling ARP packets is a crucial part in our deception system. ARP requests are usually flooded into the network to discover hosts and match IP to MAC addresses. In our system the deception server handles all ARP requests and sends the appropriate response packets. This way we can ensure that hosts which are not supposed to be discovered stay hidden. We are also able to introduce honeypots into the system by sending appropriate ARP responses.
- ***Send packets with specific TTL to deception server:*** Our SDN controller generates rules to match packets with specific TTL (Time To Live) values. This is an important part for route mutation and the introduction of virtual routers. With this function our system is able to deceive network mapping functions such as *traceroute* and make paths appear different than they actually are.
- ***On the fly adjustment of TTL fields:*** An important part of deceptive route mutation is to adjust the TTL field of response packets appropriately. This can be done on the fly with specific SDN rules in the switch when packets are passing through.

- ***Forwarding of ICMP error packets to the deception server:*** ICMP error packets, such as *Destination Unreachable* contain a nested packet which reflect the original packet received by the sender. The information in a nested packet is not automatically adjusted when it passes through a SDN switch and would therefore leak information from the real network into the virtual network. Therefore ICMP error messages are forwarded to the deception server in our system and the information in nested packets is adjusted appropriately before it is delivered to its destination host.
- ***Routing of DHCP packets:*** As we discuss in Section IV-B, a virtual network view is associated with a DHCP lease. Therefore our deception server also serves as a DHCP server and assigns a lease associated with a virtual network view to a host that tries to connect to the network. Our SDN controller installs rules to match DHCP discover packets and forward them to the deception server. Additional rules ensure that response packets are correctly transmitted to the requesting host.
- ***Routing of DNS packets:*** To guarantee reachability of legitimate services in a network, DNS requests are handled by our deception server as we explain in Section IV-D. To route DNS packets, the appropriate flow rules between nodes and the deception server are established.
- ***Routing packets to and from honeypots:*** The use of honeypots enables our system to make a network appear significantly larger than it actually is. Honeypots are also used as decoys for adversaries. With the use of dynamic header rewriting (explained later), we are able to make one honeypot appear as many different network endpoints to a scanner. The flows from and to honeypots are monitored and flow statistics are analyzed by the SDN controller for the identification of malicious hosts.
- ***Dynamic address translation:*** To hide the real addresses in a virtual network view, our system rewrites packet headers on-the-fly according to the specification of a virtual network view. Hereby we differentiate between the *real IP address*, which is seen in the real underlying network and the *deception IP address* which is only seen by a host that has a specific virtual network view assigned.
- ***Packet tagging and queueing:*** To adjust delay and bandwidth for the simulation of network characteristics, the flow rules to control traffic deployed by our SDN controller forward packets through specified queues on a switch to regulate bandwidth and add a tag to each packet. We configure the queues for bandwidth regulations according to a fair queuing policy to limit the negative impact of bandwidth regulation on benign traffic. The addition of artificial delay is done in a decentralized way at the end host by sending reply packets through a pre-deployed queuing discipline at the end host. With this architectural design we aim to prevent the formation of a system bottleneck. We explain this in detail in Section IV-E.

By using SDN flow rules to steer and control the network traffic, RDS also acts as a distributed access control list. If a view node tries to send packets to a host that is set as being invisible in the virtual network view, the packet is silently dropped and not forwarded. Besides constructing SDN flow rules based on the virtual network view description, our SDN controller also analyzes flow statistics to detect malicious activity. Upon the identification of a host with scanning activity, based on its transmitting traffic pattern, the SDN controller notifies an administrator and removes the appropriate flow rules from the switch to isolate a malicious host. We further discuss this in Section V-F.

1) On the Benefits of Using SDN: Software Defined Networking provides a unique platform for efficient and centralized network management. The framework provided by SDN to define traffic rules on a packet flow level, allows network operators to dynamically deploy security policies with a higher granularity compared to common network stacks. This ability, to programmatically control network traffic in an agile way, enables the implementation of novel, policy driven, defense approaches as we present in this paper.

To maintain a scalable number of flow rules on a SDN switch, which can impact performance [21], [42], a best practice for the deployment of flow rules is a re-active way which we implement in our system. Here, an initial packet triggers the generation of a flow rule in the controller which is dynamically deployed on the data plane and automatically expires after an idle timeout (30 seconds in our implementation). In Section VI we evaluate the scalability of flow rules to implement our network deception system.

D. Deception Server

To process the traffic forwarded through the flow rules to our deception server, we implement different handlers which receive packets from nodes connected to the network and craft responses according to the view specification. It has six main components that are essential for deceiving malicious scanners as introduced as follows:

- ***DHCP Handler:*** The DHCP handler component acts similar to a DHCP server and is responsible for assigning DHCP leases to nodes which want to connect to the network. Every virtual network view is associated with a DHCP lease that is assigned for a specific duration to a node connecting to the network. If a device sends a request for a DHCP lease to our deception server, the deception server triggers the creation of a virtual network view and assigns a DHCP lease.
- ***ARP Handler:*** All transmitted ARP requests are forwarded by appropriate flow rules to our deception server. Based on the specifications in a virtual view file, our deception server crafts an ARP response packet and sends it to the requesting node. If the requesting node is not allowed to connect to the address requested in an ARP packet, the deception server will not send a response. In case the view specification places the requested node outside of the requester's subnet, an ARP packet with the

address of the according virtual gateway/router is sent in response.

- ***ICMP Handler:*** ICMP error messages are forwarded by specific flow rules to our deception server. Packets such as a *Destination Unreachable* messages often contain nested packets with the original information received by the transmitting host. The information in nested packets is not automatically updated in SDN switches, therefore we are forwarding such packets to our deception server where the information in nested packets is adjusted according to the specified virtual network view before the packet is delivered to its destination.
- ***DNS Handler:*** To guarantee the reachability of legitimate services, our deception server also handles DNS requests, and creates appropriate responses. Hereby, the DNS entries are specific to network views assigned to nodes and are removed with the expiration of virtual network views. In RDS, the overhead caused by updating DNS entries is acceptable since it only has to be done with the assignment of a new network view, which duration is usually between a few hours to multiple days.
- ***Gateway Simulator:*** To appear realistic, some endpoints are simulated by our deception server which sends appropriate response packets if a probing packet is received. Certain components of a virtual network view do not have an actual endpoint. Such endpoints are for example virtual routers or gateways that connect virtual subnetworks.
- ***Route Simulator:*** The route simulator is responsible for the deception of network mapping functions such as *traceroute*. If a malicious scanner is sending probing packets to a specific node with TTL values lower than the number of hops specified in the virtual network view, our deception server answers on behalf of a virtual gateway/router that is on the path between the scanning source and destination.

E. Traffic Queuing and Delay Handling

Besides using scanning methods to map a network topology as we discuss in Section V-C, advanced attackers can also use techniques, such as analyzing the statistics of round trip times or the measured bandwidth on links to find inconsistencies between the physical network characteristics and the observed topology. This would enable attackers to differentiate real from virtual nodes. To thwart such attack strategies, we introduce methods to guarantee consistency of collected measurements. By using queuing and introducing artificial delay to certain packets, we change the link bandwidth $LB \rightarrow LB'$ and the host delays $HD \rightarrow HD'$ so that the physical characteristics an attacker is able to observe from a virtual network view are transformed into characteristics that are controlled by our system $PNC \rightarrow PNC'$. The introduction of *noise* into measurements will significantly limit the accuracy of attack methods analyzing collected measurements as stated in [38].

We calibrate the listed physical features of a network for honeypots and real hosts with the generation of a virtual network view. In the following, we discuss our approach for configuring delay handling and traffic queuing, so that honeypots and real hosts are indistinguishable for adversaries.

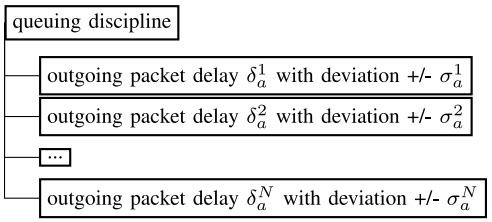

Fig. 4. Queuing disciplines to delay outgoing packets.

Adjusting the characteristics, bandwidth and delay, measured from a view node to real hosts has to be done with caution since we want to limit the overhead we artificially introduce to benign traffic in the network.

Bandwidth and delay measurements to a benign server under realistic traffic conditions will naturally show some variance depending on the current network status. To guarantee consistent characteristics, our systems initially collects measurement data from real servers and uses those as the basis to generate characteristics for virtual network views.

1) Delay Consistency: To control the delay in virtual network views, RDS deploys a system of traffic queuing disciplines at each destination host in a decentralized way, which contain multiple traffic control classes to adjust the delay of reply packets from a destination host depending on its location in a virtual topology. The virtual delay δ_v observed to an endpoint can be seen as the sum of existing delay δ_e plus the introduced artificial delay δ_a as shown in Equation 1.

$$\delta_v = \delta_e + \delta_a \tag{1}$$

Based on the hop distance in a virtual topology from a view node to a real server, we calculate the artificial delay δ_a that has to be added to the existing delay in a virtual network view. To add the delay δ_a to a host which is placed in a virtual network view, we calculate the artificial delay δ_a^{hd} for hop distance *hd* and delay outgoing packets at the specific host. To delay packets from a node we use the traffic control functionality on Linux hosts. Since hosts can be placed in different virtual network views concurrently, we install a system of *qdiscs (queuing disciplines)* [3], where each queue adds a specific artificial delay δ_a^{hd} to outgoing packets as shown in Figure 4.

The queuing disciplines as shown in Figure 4 are calibrated on system startup and deployed on each honeypot and real host, which represent potential attack targets, in a network. To determine which packets to assign to which queuing discipline, the SDN controller in our system deploys flow rules which tag packets to a destination host according to their virtual network view. Based on the VLAN tag, the system of queuing disciplines at a destination host can identify the packet and delay its outgoing transmission by δ_a^{hd}. The tag on reply packets is removed at the switch, as we show in Figure 5.

To guarantee consistency of hosts within subnets we need to ensure that the combined delay δ_v^{hd} of δ_e and δ_a^{hd} in a subnet *s* with a hop distance of *hd* to the view node is consistent for all honeypots and real hosts in *s*.

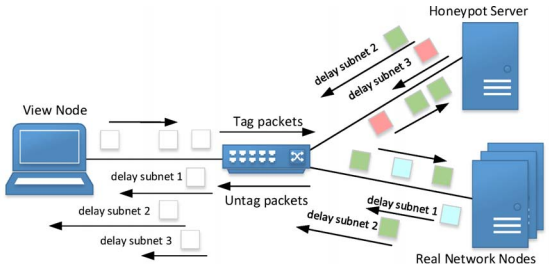

Fig. 5. Tagging packets to add artificial delay.

Algorithm 1 DetermineDelayValues()

1: $delay_{RH}$ = {}, $delay_{HP}$ = {}
2: $deviation_{RH}$ = {}, $deviation_{HP}$ = {}
3: *hopdistance* = {}, *hopdelay* = {}, *hopdev* = {}
4: $delay_{artificial}$ = {}, $dev_{artificial}$ = {}
5: **for all** *node* in network **do**
6: d_s[] = collect list of delay samples to *node*
7: calculate average delay $\overline{d_s}$ from d_s[]
8: calculate standard deviation σ from d_s[]
9: **if** *node* is real host **then**
10: $delay_{RH}[node] = \overline{d_s}$
11: $deviatios_{RH}[node] = \sigma$
12: **end if**
13: **if** *node* is honeypot **then**
14: $delay_{HP}[node] = \overline{d_s}$
15: $deviatios_{HP}[node] = \sigma$
16: **end if**
17: *hopdistance*[*node*] = hopcount to *node*
18: **end for**
19: calculate average delay per hop $\overline{hop_{delay}}$ based on $delay_{RH}$
20: calculate average deviation $\overline{\sigma}$ based on $delay_{RH}$
21: **for all** hop distance *hd* in maximum virtual network diameter **do**
22: **if** *hopdelay*[*hd*] and *hopdev*[*hd*] can be interpolated **then**
23: *hopdelay*[*hd*] = interpolate(*hd*,*hopdistance*,$delay_{RH}$)
24: *hopdev*[*hd*] = interpolate(*hd*,*hopdistance*,$deviatios_{RH}$)
25: **else**
26: $hopdelay[hd] = hd \cdot \overline{hop_{delay}}$
27: $hopdev[hd] = hd \cdot \overline{\sigma}$
28: **end if**
29: **end for**
30: **for all** *node* in network **do**
31: **for all** hop distance *hd* in maximum virtual network diameter **do**
32: δ_v^{hd} = *hopdelay*[*hd*]
33: σ_v^{hd} = *hopdev*[*hd*]
34: **if** *node* is real host **then**
35: $\delta_a^{hd} = \delta_v^{hd}$ - $delay_{RH}[node]$
36: $\sigma_a^{hd} = \sigma_v^{hd}$ - $deviatios_{RH}[node]$
37: **end if**
38: **if** *node* is honeypot **then**
39: $\delta_a^{hd} = \delta_v^{hd}$ - $delay_{HP}[node]$
40: $\sigma_a^{hd} = \sigma_v^{hd}$ - $deviatios_{HP}[node]$
41: **end if**
42: $delay_{artificial}[node][hd] = \delta_a^{hd}$
43: $dev_{artificial}[node][hd] = \sigma_a^{hd}$
44: **end for**
45: **end for**
46: **return** [$delay_{artificial}$,$dev_{artificial}$]

The calibration of consistency features for honeypots is performed based on the features observed from real nodes. In the case of honeypots we have to ensure that the physical characteristics of honeypots in a subnet *s* correlate to the characteristics of real hosts in the same subnet *s*, so that an adversary is not able to differentiate these two types of endpoints in a virtual network view.

To configure the system of queuing disciplines for each host as shown in Figure 4, we have to determine the base delay δ_a^{hd}, as well as the bounds $+/-\sigma_a$ to randomly select the delay time from a uniform distribution for an outgoing packet for each queue. We determine these factors in Algorithm 1.

TABLE I
MEASURED AND CALCULATED DELAY VALUES (IN [MS])

Type	Hop distance	σ_e	δ_e	-	-	-
Measured	6	0.09	25.23	-	-	-
Measured	1	0.07	1.46	-	-	-
Measured	5	0.12	22.07	-	-	-
Measured	4	0.16	16.43	-	-	-
Measured	1	0.09	1.80	-	-	-
Measured	10	0.25	76.25	-	-	-

Calculated values for a host with $\sigma_e = 0.07$ and $\delta_e = 1.46$						
Type	**Hop distance**	σ_a	δ_a	δ_v	Chi^2 **Result**	$> \alpha$
Calculated	1	0.01	0.17	**1.63**	1.000	✓
Calculated	2	0.02	5.10	**6.56**	1.000	✓
Calculated	3	0.04	10.03	**11.49**	1.000	✓
Calculated	4	0.04	14.95	**16.41**	0.999	✓
Calculated	5	0.04	20.59	**22.05**	0.999	✓
Calculated	6	0.02	23.77	**25.23**	0.999	✓
Calculated	7	0.04	36.52	**37.98**	0.999	✓
Calculated	8	0.06	49.29	**50.75**	0.999	✓
Calculated	9	0.07	62.06	**63.52**	0.999	✓
Calculated	10	0.10	74.83	**76.29**	0.999	✓

We calibrate the introduced queuing disciplines based on collected measurement data from existing hosts in the network. Algorithm 1 starts by sampling delay data from real hosts and honeypots in the network and calculate the average delay $\overline{d_s}$ and standard deviation σ from a node (lines 5-18). Based on the collected data we calculate the average delay per hop distance $\overline{hop_{delay}}$ and the average standard deviation per hop distance $\overline{\sigma}$ (lines 19-20).

If enough hosts can be sampled for delay, δ_v values for certain hop distances can be interpolated (lines 23-24). If interpolation is not possible, for example if a delay value for a specific hop distance is outside the collected data, we multiply the hop distance *hd* with the average delay and standard deviation (lines 26-27). This results in an array of delay values, $hopdelay[\delta_v^1, \delta_v^2, \delta_v^3, \ldots, \delta_v^N]$, where N is the maximum hop distance in a virtual network view. A similar array is generated for the standard deviation per hop.

In the remaining part of Algorithm 1 (lines 30-45) we use the determined data to calculate the artificial delay δ_a^{hd} and standard deviation σ_a^{hd} for each real node in the network and for 1...*N* hop distances *hd* to configure the introduced queuing disciplines.

Since artificial delay can only be added to packets, but not subtracted, the existing delay δ_e to a node has to be considered for the placement in a virtual network view. The returned list of delay values in Algorithm 1 indicates the minimum hop distance a host has to be placed in a virtual network view from the view node to guarantee consistent delay measurements. For example, if the list $[-\delta_a^1, -\delta_a^2, -\delta_a^3, \delta_a^4, \delta_a^5]$ is returned, the corresponding node has to be placed at least 4 hops away from the view node in a virtual topology.

To evaluate the correctness of Algorithm 1 we compare the distribution of generated artificial delays with the distribution of measured delays with a Chi^2 test. At the top part of Table I we show measured delay data in our network from different hosts. The bottom part shows the calculated δ_a^{hd} and σ_a^{hd} for a host with $\delta_e^1 = 1.46$ and $\sigma_e^1 = 0.07$. The bottom part of Table I also shows the resulting δ_v values and the Chi^2 test results which show that the artificially generated host delays are statistically indistinguishable from the measured delay distributions. If the calculated Chi^2 test result is greater than α

(we use a 95% significance level $\alpha = 0.05$) the compared distributions are statistically indistinguishable, which is the case in all our evaluated samples.

2) Bandwidth Consistency: Since real hosts and honeypots can be used in multiple different virtual network topologies, we use queuing in our system to guarantee a minimum bandwidth for each view node and make the observed link bandwidths to network endpoints consistent. To implement QoS in our system, we use the OpenFlow functionality to forward packets to a *queue id* instead of forwarding them directly to a switch port. On a switch, we have to configure a list of queues to control the bandwidth to virtual network endpoints. The actual queues have to be configured on the switch which can require vendor specific commands. For the development of RDS we used OpenVSwitch [7], which can also be executed on bare-metal hardware switches. Here a QoS policy has to be created to implement queues which aim to stay between an upper and a lower bound of a specified bandwidth.

To configure queuing for controlling the rate to nodes, we have to specify the overall rate of the QoS policy QoS_{rate}, the guaranteed rate on each queue $Queue_{min}$ and the maximum rate on each queue $Queue_{max}$. To configure the QoS policy, it has to hold that the overall QoS rate is less than the sum of all guaranteed rates: $(QoS_{rate} \leq \sum_i^N Queue^i_{min})$.

To provide a fair share of bandwidth to each user, we consider the available link rate LR to a host in our network. To determine the minimum guaranteed rate to a node $Queue^i_{min}$, we divide the overall rate LR by the number users N of this link, so that $Queue^i_{min} = LR \ / \ N$. We can set the maximum available rate for a user to $Queue_{max} = LR$.

With this configuration the rate a user can observe to different network endpoints depend on the current traffic load and network status. This models the bandwidth distribution between hosts how it would usually be done in a network with a fair queuing scheduling policy.

F. System Prototype

We release a proof of concept prototype implementation of RDS at [10] as open-source. The prototype can be tested in standard SDN emulators, such as Mininet, and on hardware platforms that support the required OpenFlow functions. We would like to point out that the implementation of our system we release is a research prototype that gives a proof of concept of the introduced deception techniques and was used in the experiments we discuss in the evaluation section. The released software is not at a status that is ready for the market or can directly be deployed in a production network.

V. Evaluation

In the following we present experimental results about the performance of our proposed defense system.

A. Test Environment

To evaluate our system in a real world setting and measure its performance, we emulate a SDN based enterprise network as shown in Figure 6. Our test network consists of multiple connected subnetworks, where the nodes in each subnetwork are connected with an instance of OpenVSwitch [7] which is controlled by a SDN controller. To emulate the functionality of our test-network we use Mininet [4]. All hosts in our test-network are running Ubuntu Linux 14.04. The topology of the test-network, we use for our experiments, is a medium-size enterprise network with four subnetworks that contain different servers providing services to the clients on the network. The servers host services such as Web-servers, printer server, database server and shared network directories which are constantly used by the clients.

Multiple client endpoints in our test-network have virtual network views simulated (we visualize two in Figure 6), and therefore see a different network than the real underlying network and have access to a subset of the endpoints and services provided in the real network.

In Figure 6 we also show how our system can be deployed in a distributed manner. Deployment of distributed SDN controllers is an ongoing research topic, for the purpose of this work to evaluate the introduced defense techniques, we implemented the required functionalities in our SDN controller to forward network traffic appropriately between different connected subnetworks. Two of the SDN controllers that control the subnetworks where nodes have virtual network views simulated, are the RDS controllers. The remaining SDN controllers steer the network traffic in two subnets where currently no nodes have virtual network views simulated. These controllers can be off-the-shelf such as layer 2 learning switches implemented in platforms like POX, OpenDayLight or Floodlight. Figure 6 also shows the deployment of our deception servers; each deception server handles the simulation of the virtual network views in a subnetwork. We want to demonstrate with this setup that our system can be deployed in a distributed manner and is therefore able to scale to larger networked systems.

B. Invalidation of Attacker Information

Many targeted cyber attacks, such as Advanced Persistent Threats (APT) [1], [13], [15], [36], [46], depend on network reconnaissance and discovery missions where the internal topology of a network is mapped. The purpose of an attacker with such missions is to gather the set of information T (as discussed in Section III-A) for further attack planning.

As discussed in Section IV, RDS generates a new virtual network view with every assignment of a DHCP lease to a host in a network. The duration of a DHCP lease can be adjusted by network administrators and can be in the order of a few hours to multiple days. RDS is able to assign a different virtual network view to every host in a network. Using RDS, the features of the real network NF are transformed into the feature set of a virtual network view NF'; this will invalidate the information collected by an attacker, by transforming T into T'. This is periodically achieved with every assignment of a new DHCP lease that is correlated to a different virtual network view. In a newly assigned virtual network view the topology, network size and address space, honeypot and host placement has changed after the assignment of a new DHCP lease and the connecting host sees a new network topology

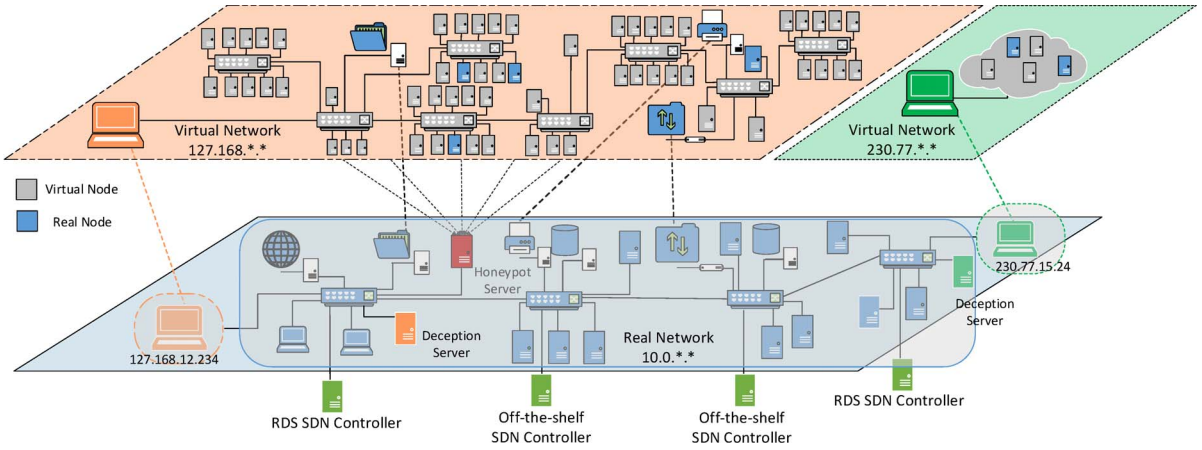

Fig. 6. Emulated enterprise test network.

that is significantly different than the previous one. Targeted cyber attacks, such as APT, which depend on collecting information over long periods of time about the composition of a network are not able to gather consistent information about the system infrastructure necessary to plan further attack steps.

To show that our system achieves deception and makes an attacker believe a virtual network topology is real, we evaluated our deception system with NMAP [5] and multiple adversarial scanning strategies we discuss in the next section. We performed hundreds of NMAP scans in virtual network topologies simulated by our deception system. All matched the exact specification of the virtual network view, thus achieving complete deception and minimizing the amount of useful information an attacker gathers in the set T'. We also evaluated virtual network views generated by our system with the recently proposed open source tool *Degreaser* [19]. *Degreaser* is designed to detect network deception techniques. We tested multiple virtual network views with different strategies and configurations, *Degreaser* did not label any hosts or honeypots in these views as being a decoy. This shows that our system is able to make virtual network views appear realistic and believable. In addition to invalidating the collected information of adversaries, we will show in the following that our system significantly delays the detection rate of vulnerable hosts by attackers and show how the gained time is used to identify the source of malicious reconnaissance traffic.

C. Defending Malicious Network Scanning

To evaluate the effectiveness of RDS against network scans we implemented a number of common network scanning techniques which are discussed in the literature and are known to be used in malware as discussed in [18], [25], [26], [39], and [40]. To implement these scanning strategies we used the python library *libnmap* [9], which provides an API to NMAP [5], as well as the python framework Scapy [11].

As discussed in [40], an adversarial scanner selects a *scanning space (Ω)* which denotes the IP address space that is considered for selecting addresses to probe. In an enterprise network, the considered scanning space is usually selected based on the IP address prefix of the network an adversary aims to probe. Also the *address distance (λ)*, which specifies the numerical difference between the IP addresses of a scanner and its scanned target, has an impact on the performance of a scanning technique. The following introduced scanning techniques actively probe a network for its features NF to retrieve an information set T which can be used for further attack planning.

Uniform scanning probes random hosts within the scanning space Ω. Each probe transmitted by a scanner, has an equal probability to detect a potentially vulnerable host in the network. The detection time of vulnerable hosts in this scanning strategy depends on the size of the overall scanning space Ω. IP addresses to probe are chosen randomly, therefore every transmitted probe has an equal chance of $\frac{n}{\Omega}$ to hit a vulnerable host h if a network contains n vulnerable hosts.

Local-preference scanning, as discussed in [25] and [40], is a biased scanning technique where certain regions of a network are chosen based on information that can be retrieved from the local host. In current state-of-the-art networks, hosts are not uniformly distributed within the address space. An adversarial scanner can increase the speed to detect vulnerable hosts when it scans the IP space where hosts are more densely distributed as explained in [40]. Local preference scanning takes advantage of this and scans IP addresses that are closer to its own local address and therefore have a smaller address distance $\lambda(h)$, with higher probability.

Preference sequential scanning probes the IP address space sequentially, i.e., in an additive way. In *preference sequential scanning* we assume that a scanner is using local preference and selects a start IP address with a small address distance $\lambda(h)$ to its host IP address.

Non-preference sequential scanning is similar to *preference sequential scanning*, but selects its starting IP address in a random manner within the scanning space Ω. Sequential scanning without local preference can show a better performance than *preference sequential scanning*, since the selected start IP address can be closer to addresses of vulnerable hosts than the local address of the scanning host.

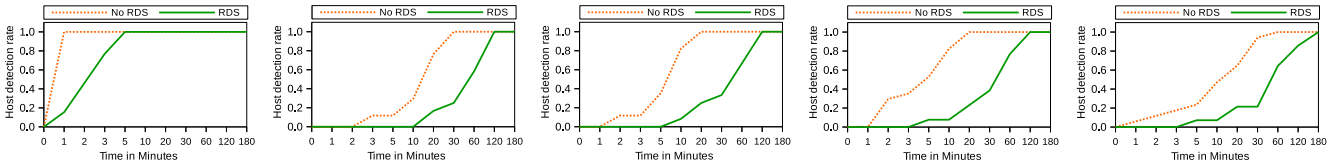

Fig. 7. Average vulnerable host infection rate over time for the scanning strategies *Preference Parallel*, *Local Preference*, *Preference Sequential*, *Non-Preference Sequential*, *Uniform* with and without our deception system. Vulnerable hosts are distributed evenly over the address space within 12 virtual subnets.

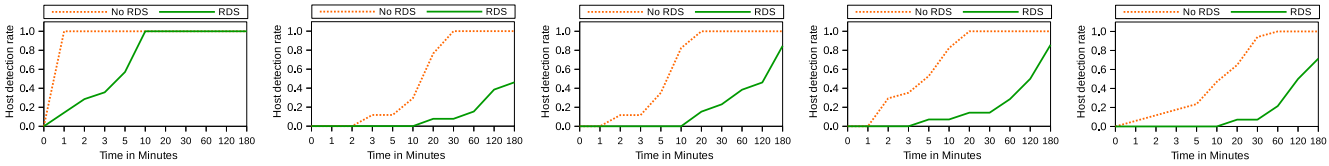

Fig. 8. Average vulnerable host infection rate over time for the scanning strategies *Preference Parallel*, *Local Preference*, *Preference Sequential*, *Non-Preference Sequential*, *Uniform* with and without our deception system. Vulnerable hosts are distributed evenly over the address space within 25 virtual subnets.

Preference parallel is a technique which is using parallelism that can significantly increase the performance of a scanning method, but has the drawback of causing a large amount of network traffic which makes it easy to detect. This technique can also be seen as a simulation of a type of *cooperative scanning* or *divide-and-conquer scanning* where multiple hosts cooperate with each other to find vulnerabilities. With this strategy multiple probing messages are sent out in parallel using local preference. We use 12 parallel probing messages in our experiments.

D. Delaying Adversarial Scanners

To show the effectiveness of our deception system in delaying the identification of vulnerable (real) hosts by adversaries, we executed the introduced malicious scanning techniques in the real underlying network without our system in place and compare it to the performance of the scanning techniques when virtual networks are simulated for the nodes in the underlaying network. In our evaluation we consider every real host that is visible in a virtual network view as being a potential target that can be corrupted by a cyber attack and is therefore vulnerable.

A simplified visualization of the test-network for this experiment is shown in Figure 6. We measured the detection ratio of vulnerable (real) hosts in relation to scanning time over multiple iterations and show the averaged results. For each scanning technique, the scanning performance is shown with *(RDS)* and without *(No RDS)* our system deployed.

In Figure 7 and 8 we present experimental results when the introduced adversarial scanning techniques are applied in simulated virtual networks. In the shown charts, we present the detection ratio of vulnerable (real) hosts which are placed in a virtual network on the Y axis in relation to the scanning time shown on the X axis.

The results presented in Figure 7 are measured when virtual networks with 12 subnets and up to 25 hosts per subnetwork are simulated and vulnerable (real) hosts are distributed evenly over the virtual network. RDS delays a malicious scanner in our experimental scenario by ∼40 minutes on average. This delays attackers scanning for the network for vulnerable hosts by a factor of 10 on average, which is sufficient to identify a malicious scanner and isolate it from the network as we will show in Section V-F.

In Figure 8 we present the performance of the introduced adversarial network scanning techniques when virtual networks with 25 subnets and up to 45 hosts per subnetwork are simulated and vulnerable (real) hosts are distributed evenly over the virtual network. Due to the simulation of virtual networks that are significantly larger, our system is able to delay an adversarial scanner by a factor of 22 on average resulting in an additional ∼100 minutes on average until an attacker is able to identify a vulnerable host.

Delaying reconnaissance missions with our deception system depends on the size of the simulated virtual network and the placement of vulnerable hosts which is achieved by transforming the feature set NF of the real network into the feature set NF' of a virtual network as discussed in Section III-C. In further experiments we were able to delay adversaries seeking to identify vulnerable hosts of up to a factor of 115. This can be achieved by simulating large enough virtual networks and using a strategy where vulnerable hosts are placed with high address distance from the scanning source. This shows that for the defense against adversarial network reconnaissance and discovery, the principle: *The bigger the haystack, the longer the search* is simple but very effective.

E. Defending Analysis of Network Characteristics

As we discuss in Section IV-E, attackers can use techniques, such as statistical analysis, to determine certain features of a network and differentiate real hosts from honeypots. To analyze the techniques introduced in Section IV-E, we collect response time statistics from hosts and apply a number of statistical tests to detect if we are able to differentiate vulnerable hosts from honeypots. In the performed experiment, the real hosts, which we consider as vulnerable, are located

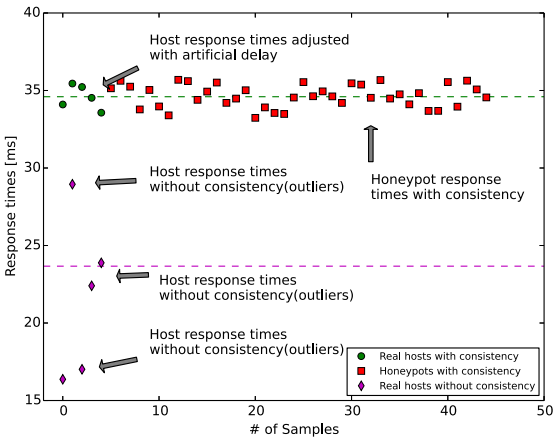

Fig. 9. Comparison of host response times with and without consistency.

TABLE II
HOST DETECTION BASED ON RESPONSE TIME

Test	# Hosts	Detected real hosts	False Positives	Consistency
Generalized ESD Test	50	5 (out of 5)	1	deactivated
Z-score Test	50	5 (out of 5)	0	deactivated
Modified Z-Score Test	50	5 (out of 5)	0	deactivated
Generalized ESD Test	50	0 (out of 5)	0	activated
Z-score Test	50	1 (out of 5)	7	activated
Modified Z-Score Test	50	0 (out of 5)	0	activated

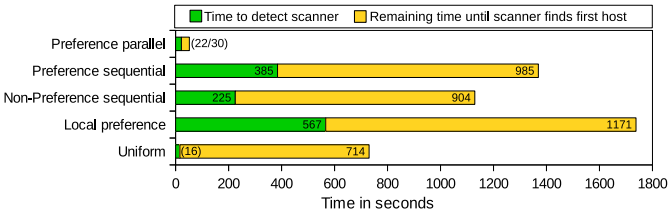

Fig. 10. Average time to detect a malicious scanning source and remaining time until the first vulnerable host is identified by a scanner in a virtual network with 12 subnets and 25 hosts per subnet.

closer to the view node and therefore have a faster response time compared to our honeypot server.

In Figure 9 we show a comparison of measured host response times. The red squares show the response times of honeypots when the consistency techniques introduced in Section IV-E are applied, while the purple diamonds show real hosts without the consistency techniques applied. In this case the real hosts are outliers and can be identified by applying methods to detect statistical outliers.

The green circles show the adjusted host response times with our consistency techniques in place, which can no longer be differentiated from the response times observed from honeypots. Depending on the network status and traffic load, the existing delay of hosts may vary. To address such cases a higher granularity of queuing disciplines can be defined so that the SDN controller is able to dynamically change the tagging of packets and send them to a different queuing category at the destination to adjust the artificial delay δ_a.

In Table II we show a comparison of the statistical test results to distinguish real hosts from honeypots based on the observed response times. As defined by the *National Institute of Standards and Technology (NIST)* [2] we use three different tests for the detection of statistical outliers to differentiate real hosts from honeypots. In the shown experiment we simulate virtual subnetworks with 50 hosts, consisting of 45 honeypots and 5 real hosts. We compare the test results when our consistency module is deactivated and activated. If the response times are not adapted to be consistent, the applied tests are able to distinguish the real hosts from the honeypots within a virtual subnetwork. When our consistency modules are activated the response times of honeypots correlate to the response times of real hosts which makes them statistically indistinguishable. For the results shown in Table II we configured the honeypot response times by using Algorithm 1.

F. Identification of Malicious Nodes

In a software defined network, the controller is able to request flow rule traffic statistics from a switch. In our RDS, the SDN controller monitors the flow rules between honeypots and real hosts. We assume that in general benign network nodes are not sending probing messages to random addresses or establish connections to honeypots. In case a node starts to send packets to honeypots a malicious activity can be assumed and the transmitting node can be observed more closely. In our RDS, the SDN controller periodically requests flow statistics from the SDN switches to check how many packets where transmitted to honeypots. If our controller notices traffic from a node to honeypots, we flag the transmitting node as a potential scanner and isolate it from the network. Using flow statistics has already been proven as being an efficient technique for real time detection of anomalies as discussed in [24].

In Figure 10 we show the average identification times of a malicious scanning host in a our test environment by our SDN controller, and the remaining time until a scanning node will detect the first vulnerable (real) host in a virtual network. The virtual topologies used to evaluate these results have similar characteristics as used for the results presented in Figure 7.

As shown, by analyzing the SDN flow rule statistics our system is able to identify a malicious scanning source before it detects any vulnerable hosts. Upon the detection of a scanning node, our system is able to isolate a potentially malicious host by updating the appropriate flow rules from the SDN switch and notify an administrator. Updating flow rules in SDN switches can be done in a fraction of a second as discussed in [42]. As we present in Figure 10, in the test environment RDS identifies scanning activity at least 30 seconds before any vulnerable hosts will be detected, this gives our system enough time to isolate a potentially malicious host from the rest of the network.

VI. SYSTEM PERFORMANCE

The increased security we seek to achieve with the simulation of virtual networks, has associated costs. In the case of RDS, the cost to defend malicious reconnaissance missions can be quantified in terms of the latency overhead added to legitimate traffic and the number of required SDN flow rules for the simulation of virtual networks.

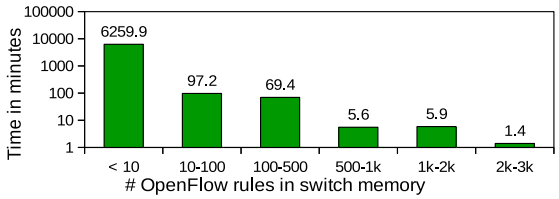

Fig. 11. Average number of OpenFlow rules in a SDN switch per minutes of system operation time over 100 hours.

To guarantee consistency in the measured physical characteristics in a virtual network view, we introduce artificial delay and control the link bandwidth as discussed in Section IV-E. The latency overhead on benign users if our consistency module is activated is defined by the artificial delay δ_a and standard deviation σ_a added to the existing delay δ_e of a network endpoint. The total delay, $\delta_v = \delta_e + \delta_a$, also depends on the placement of a node in a virtual topology. As explained in Section IV-E1, the artificial delay δ_a increases with the virtual hop distance of a host in a virtual topology to the view node. Therefore the upper bound of delay overhead introduced by our system can be defined as $(\delta_a^{hd} + \sigma_a^{hd})$, where *hd* is the hop distance to the view node. The overhead in terms of bandwidth capacity is bounded by the guaranteed rate $Queue^{i}_{min}$ we assign to a traffic queue for a host i as we discuss in Section IV-E2.

To evaluate the overhead in terms of the number of SDN rules generated by our system, we measured the current number of rules in a switch's memory per minute while our system was deployed in a network. In Figure 11 we show data collected over more than 100 hours of operation of our system, while various scanning activity was performed by multiple hosts that had virtual network views with 12 subnets and 25 honeypots per subnet simulated. As shown, the vast majority of the time, less than ten OpenFlow rules where installed on a switch and overall more than 500 rules where installed for less than 15 minutes during an operation time of more than 100 hours. On average we measured 5.4 rules in a switch's memory during more than 100 hours of operation time of our system. As we discuss in Section IV-C1, a number of SDN rules of this magnitude can be handled by modern SDN switches and should not cause any noticeable performance impact.

VII. Conclusion

In this paper we define a deception approach as a defense strategy against network reconnaissance and introduce the design and implementation of RDS. We introduce a generator to create virtual network views, based on our defined deception approach, to camouflage critical resources and place vulnerable endpoints at locations in a virtual network topology to significantly increase their detection time by an adversary. By analyzing the traffic flows, our system is able to identify the source of adversarial reconnaissance traffic and isolate a malicious host from the network. The evaluation we present in this work shows that our system is able to delay malicious network scans up to a factor of 115, while controlling the performance impact on legitimate traffic in the network.

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 the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation hereon.

References

[1] *Advanced Persistent Threats—Attack and Defense*. Accessed on Feb. 20, 2017. [Online]. Available: http://bit.ly/2jrUIyI
[2] *Detection of Outliers*. Accessed on Jan. 27, 2017. [Online]. Available: http://bit.ly/1mneyrN/
[3] *Linux Traffic Control*. Accessed on Jan. 15, 2017. [Online]. Available: http://man7.org/linux/man-pages/man8/tc-netem.8.html
[4] *Mininet—Realistic Virtual SDN Network Emulator*. Accessed on Jul. 17, 2016. [Online]. Available: http://mininet.org/
[5] *Nmap Network Scanner*. Accessed on Jun. 25, 2016. [Online]. Available: https://nmap.org/
[6] *OpenFlow*. Accessed on Jun. 12, 2016. [Online]. Available: http://bit.ly/1J7hfED
[7] *Openvswitch*. Accessed on Jun. 12, 2016. [Online]. Available: http://openvswitch.org/
[8] *Pox—Python SDN Controller*. Accessed on Mar. 21, 2016. [Online]. Available: http://www.noxrepo.org/pox/about-pox/
[9] *Python API for nMap*. Accessed on Mar. 21, 2016. [Online]. Available: https://libnmap.readthedocs.org/
[10] *Reconnaissance Deception System Prototype Implementation*. Accessed on Mar. 2, 2017. [Online]. Available: https://github.com/deceptionsystem/master
[11] *Scapy*. Accessed on Mar. 21, 2016. [Online]. Available: http://www.secdev.org/projects/scapy/
[12] *Stages of a Cyber Attack*. Accessed on Mar. 30, 2016. [Online]. Available: http://symc.ly/1PHHl3n
[13] (2011). *Symantec Advanced Persistent Threats*. [Online]. Available: http://symc.ly/2gjylGk
[14] S. Achleitner *et al.*, "Cyber deception: Virtual networks to defend insider reconnaissance," in *Proc. Int. Workshop Manag. Insider Security Threats*, Vienna, Austria, 2016, pp. 57–68.
[15] E. Baize, "Developing secure products in the age of advanced persistent threats," *IEEE Security Privacy*, vol. 10, no. 3, pp. 88–92, May/Jun. 2012.
[16] C.-Y. J. Chiang *et al.*, "ACyDS: An adaptive cyber deception system," in *Proc. MILCOM*, Baltimore, MD, USA, 2016, pp. 800–805.
[17] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast Internet-wide scanning and its security applications," in *Proc. Usenix Security*, Washington, DC, USA, 2013, pp. 605–620.
[18] E. Al-Shaer, Q. Duan, and J. H. Jafarian, "Random host mutation for moving target defense," in *Proc. Security Privacy Commun. Netw.*, Padua, Italy, 2013, pp. 310–327.
[19] L. Alt, R. Beverly, and A. Dainotti, "Uncovering network tarpits with degreaser," in *Proc. 30th Annu. Comput. Security Appl. Conf.*, New Orleans, LA, USA, 2014, pp. 156–165.
[20] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis, "Defending against hitlist worms using network address space randomization," *Comput. Netw.*, vol. 51, no. 12, pp. 3471–3490, 2007.
[21] Y. Chiba, Y. Shinohara, and H. Shimonishi, "Source flow: Handling millions of flows on flow-based nodes," in *Proc. ACM SIGCOMM*, 2010, pp. 465–466.
[22] M. Coates *et al.*, "Maximum likelihood network topology identification from edge-based unicast measurements," in *Proc. ACM SIGMETRICS Perform. Eval. Rev.*, Marina Del Rey, CA, USA, 2002, pp. 11–20.
[23] B. Eriksson, G. Dasarathy, P. Barford, and R. Nowak, "Efficient network tomography for Internet topology discovery," *IEEE/ACM Trans. Netw.*, vol. 20, no. 3, pp. 931–943, Jun. 2012.
[24] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Comput. Netw.*, vol. 62, pp. 122–136, Apr. 2014.
[25] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Adversary-aware IP address randomization for proactive agility against sophisticated attackers," in *Proc. INFOCOM*, Hong Kong, 2015, pp. 738–746.

[26] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "OpenFlow random host mutation: Transparent moving target defense using software defined networking," in *Proc. HotSDN*, Helsinki, Finland, 2012, pp. 127–132.
[27] D. Kreutz, F. M. V. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, Hong Kong, 2013, pp. 55–60.
[28] Y. Li, Z. Chen, and C. Chen, "Understanding divide-conquer-scanning worms," in *Proc. Perform. IPCCC*, Austin, TX, USA, 2008, pp. 51–58.
[29] S. McClure, J. Scambray, and G. Kurtz, *Hacking Exposed: Network Security Secrets and Solutions*. Emeryville, CA, USA: McGraw-Hill, 2009.
[30] S. Panjwani, S. Tan, K. M. Jarrin, and M. Cukier, "An experimental evaluation to determine if port scans are precursors to an attack," in *Proc. DSN*, Yokohama, Japan, 2005, pp. 602–611.
[31] P. Porras *et al.*, "A security enforcement kernel for OpenFlow networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, Helsinki, Finland, 2012, pp. 121–126.
[32] N. Provos, "A virtual honeypot framework," in *Proc. USENIX Security Symp.*, San Diego, CA, USA, 2004, pp. 1–14.
[33] S. Robertson *et al.*, "CINDAM: Customized information networks for deception and attack mitigation," in *Proc. SASO Workshop*, Cambridge, MA, USA, 2015, pp. 114–119.
[34] S. A. Shaikh, H. Chivers, P. Nobles, J. A. Clark, and H. Chen, "Network reconnaissance," *Netw. Security*, vol. 11, pp. 12–16, Nov. 2008.
[35] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: Scalable and vigilant switch flow management in software-defined networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, Berlin, Germany, 2013, pp. 413–424.
[36] A. K. Sood and R. J. Enbody, "Targeted cyberattacks: A superset of advanced persistent threats," *IEEE Security Privacy*, vol. 11, no. 1, pp. 54–61, Jan./Feb. 2013.
[37] S. T. Trassare, R. Beverly, and D. Alderson, "A technique for network topology deception," in *Proc. MILCOM*, San Diego, CA, USA, 2013, pp. 1795–1800.
[38] Y. Tsang, M. Yildiz, P. Barford, and R. Nowak, "Network radar: Tomography from round trip time measurements," in *Proc. 4th ACM SIGCOMM Conf. Internet Meas.*, Taormina, Italy, 2004, pp. 175–180.
[39] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, "A taxonomy of computer worms," in *Proc. ACM Workshop Rapid Malcode*, 2003, pp. 11–18.
[40] C. C. Zou, D. Towsley, and W. Gong, "On the performance of Internet worm scanning strategies," *Perform. Eval.*, vol. 63, no. 7, pp. 700–723, 2006.
[41] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "An effective address mutation approach for disrupting reconnaissance attacks," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 12, pp. 2562–2577, Dec. 2015.
[42] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about SDN flow tables," in *Proc. Int. Conf. Passive Active Netw. Meas.*, New York, NY, USA, 2015, pp. 347–359.
[43] E. L. Malcot, "MitiBox: Camouflage and deception for network scan mitigation," in *Proc. HotSec*, 2009, p. 4.
[44] N. Provos, "Honeyd—A virtual honeypot daemon," in *Proc. 10th DFN-CERT Workshop*, vol. 2. Hamburg, Germany, 2003, p. 4.
[45] J. Sun and K. Sun, "DESIR: Decoy-enhanced seamless IP randomization," in *Proc. IEEE INFOCOM*, San Francisco, CA, USA, 2016, pp. 1–9.
[46] A. Vance, "Flow based analysis of advanced persistent threats detecting targeted attacks in cloud computing," in *Proc. Problems Infocommun. Sci. Technol.*, 2014, pp. 173–176.

Stefan Achleitner received the B.S. and M.S. degrees in computer science from the Vienna University of Technology, Austria. He is currently pursuing the Ph.D. degree in computer science and engineering with Pennsylvania State University, University Park, PA, USA. As a member of the Institute for Networking and Security Research, his research focuses on security of network virtualization. Additional research interests include virtual machines, Internet of Things, and machine learning.

Thomas F. La Porta (F'02) is the Director of the School of Electrical Engineering and Computer Science at Penn State University. He is an Evan Pugh Professor and the William E. Leonhard Chair Professor in the Computer Science and Engineering Department. He received his B.S.E.E. and M.S.E.E. degrees from The Cooper Union, New York, NY, and his Ph.D. degree in Electrical Engineering from Columbia University, New York, NY. He joined Penn State in 2002. He was the founding Director of the Institute of Networking and Security Research at Penn State. Prior to joining Penn State, Dr. La Porta was with Bell Laboratories for 17 years. He is an IEEE Fellow and Bell Labs Fellow. He also won two Thomas Alva Edison Patent Awards. Dr. La Porta was the founding Editor-in-Chief of the IEEE TRANSACTIONS ON MOBILE COMPUTING. He served as Editor-in-Chief of *IEEE Personal Communications Magazine*. He was the Director of Magazines for the IEEE Communications Society and was on its Board of Governors for three years. He has published numerous papers and holds 39 patents.

Patrick McDaniel (F'14) is a Distinguished Professor in the School of Electrical Engineering and Computer Science and Director of the Institute for Networking and Security Research at the Pennsylvania State University. Professor McDaniel is a Fellow of the IEEE and ACM and serves as the program manager and lead scientist for the Army Research Laboratory's Cyber-Security Collaborative Research Alliance. Patrick's research centrally focuses on a wide range of topics in security and technical public policy. Prior to joining Penn State in 2004, he was a senior research staff member at AT&T Labs-Research.

Shridatt Sugrim is a Research Scientist in Applied Research at Applied Communication Sciences (ACS), where he does research and development. He supports the research efforts of the ARL Cyber Security Applied Research & Experimentation Partner (AREP) program and the DARPA Wireless Network Defense program at ACS. His areas of interests are software defined networks (SDN), model validation for machine learning, and cyber security. He is currently a PhD candidate at Rutgers University, NJ.

Srikanth V. Krishnamurthy (F'12) received the Ph.D. degree in electrical and computer engineering from the University of California at San Diego, La Jolla, CA, USA, in 1997. From 1998 to 2000, he was a Research Staff Scientist with the Information Sciences Laboratory, HRL Laboratories, LLC, Malibu, CA, USA. He is currently a Professor of Computer Science with the University of California at Riverside, CA, USA. His research interests are primarily in wireless networks and security.

Ritu Chadha (SM'07) is a Senior Research Director in Applied Research with Applied Communication Sciences (ACS), where she manages the Data Analytics Research Department. She is currently the Principal Investigator for the ongoing ARL Cyber Security Applied Research and Experimentation Partner Program and the DARPA Wireless Network Defense Program with ACS. Her areas of interest lie at the intersection of cyber security, data analytics, and wireless networking. She authored a book on policy-driven mobile ad hoc networks in 2007. She is a Telcordia Fellow.