Cyber Deception: Virtual Networks to Defend Insider Reconnaissance

Stefan Achleitner
stefan.achleitner@cse.psu.edu

Thomas La Porta
tlp@cse.psu.edu

Patrick McDaniel
mcdaniel@cse.psu.edu

Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802

Shridatt Sugrim
ssugrim@appcomsci.com
Applied Communication Sciences
Basking Ridge, NJ 07920

Srikanth V. Krishnamurthy
krish@cs.ucr.edu
Computer Science and Engineering
University of California, Riverside
Riverside, CA 92521

Ritu Chadha
rchadha@appcomsci.com
Applied Communication Sciences
Basking Ridge, NJ 07920

ABSTRACT

Advanced targeted cyber attacks often rely on reconnaissance missions to gather information about potential targets and their location in a networked environment to identify vulnerabilities which can be exploited for further attack maneuvers. 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 RDS (Reconnaissance Deception System), which is based on SDN (Software Defined Networking), to achieve deception by simulating virtual network topologies. Our system thwarts network reconnaissance by delaying the scanning techniques of adversaries and invalidating their collected information, while minimizing the performance impact on benign network traffic. We introduce approaches to defend malicious network discovery and reconnaissance in computer networks, which are required for targeted cyber attacks such as Advanced Persistent Threats (APT). We show, that our system is able to invalidate an attackers information, delay the process of finding vulnerable hosts and identify the source of adversarial reconnaissance within a network, while only causing a minuscule performance overhead of 0.2 milliseconds per packet flow on average.

1. 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 [16, 40, 32, 23, 27, 35]. 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 [37, 39, 18], and are precursors to a high percentage of cyber attacks. In particular, the authors of [28] 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 [35, 2, 1], 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 simulated 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 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 [2, 35, 1, 14] 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 [14].

MIST'16, October 28 2016, Vienna, Austria

DOI: http://dx.doi.org/10.1145/2995959.2995962

To address threats caused by reconnaissance missions of insider attackers, we define a formal deception approach which identifies certain features of a networked system that are modified by our 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 part of our proposed defense solution is the composition of virtual network views and the placement of hosts and honeypots in virtual topologies to delay attackers from identifying real hosts in a virtual topology.

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 define 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 minimizing the cost to achieve increased security. We publish an open-source proof of concept prototype of our system at [12]. 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 minimizing the performance impact on legitimate traffic (0.2 milliseconds per connection on average compared to standard SDN networks). 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 a reconnaissance deception system***
  To realize and evaluate the defined deception goals, we implement a research prototype of RDS, based on SDN, to simulate complete virtual network topologies. The combined functionality of our *deception server*, *SDN controller* and *virtual topology generator* achieves thorough network deception, while maintaining the full network functionality for legitimate traffic. As a key part of our system, we introduce a defense view generator to create virtual topologies and achieve an effective defense strategy against adversarial reconnaissance.
- ***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 has only a minuscule performance impact of 0.2 milliseconds per packet flow on average.
- ***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.

2. PROBLEM DEFINITION

In this section, we formally define what we mean by insider reconnaissance. We also describe the threat model that we apply in the remainder of the paper. As mentioned earlier, our goal is to deceive an adversary who seeks to perform insider reconnaissance.

2.1 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:
$T = \{AV =$ *Addresses of potentially vulnerable hosts,* $NS =$ *Network size,* $ST =$ *System topology*$\}$
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 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 cardinality of 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.

2.2 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 [2, 14, 35, 1, 15, 42]. 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. [34, 29, 24, 42, 38]), 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.

2.3 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:
$NF = \{TL = Topological\ location\ of\ hosts,\ NH = Number\ of\ hosts,\ AH = Addresses\ of\ hosts,\ CH = Connectivity\ between\ hosts\}$
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,
$NF' = \{TL', NH', AH', CH'\} \rightarrow T' = \{AV', NS', ST'\}$
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 3.2, 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 3.2, 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 [41].
- ***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 4.4, 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.

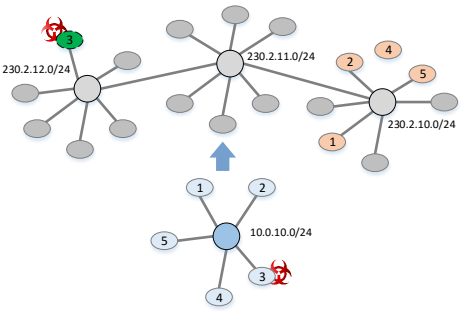

Figure 1: An example of network deception via the projection of a virtual network which places critical resources in a way to deceive adversaries.

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 virtual 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.

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 4.3. 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 as we will explain in detail in Section 3.

3. 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.

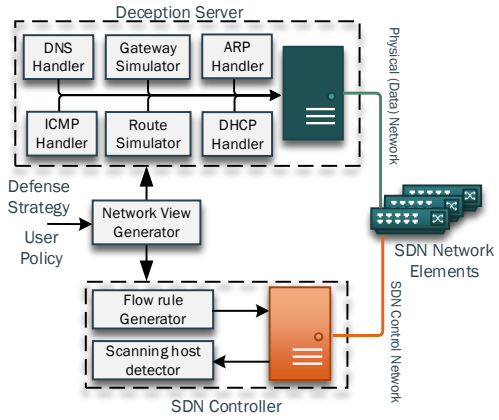

Figure 2: Architecture of RDS

3.1 System architecture overview

Our system comprises three 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, and a *virtual network view generator* which provides a machine readable description of the virtual network components and their connectivity.

In the design of our system we consider a strict separation of the physical (data plane) network and the SDN control network which is established between SDN components. This ensures that the SDN controller is protected from potential threats emerging from the data plane and adds an additional layer of security. Due to these design decisions we do not inject packets from the SDN controller into the data network. Generation and injection of packets is handled entirely by our deception server. Using a deception server, which is separated from the SDN controller, for handling the virtual traffic in our system is an important aspect in terms of scalability since the deception server can be replicated to distribute the load of network traffic.

Our system is implemented in *Python*. We use the POX framework [10] to implement our SDN controller and the Scapy framework [13] to implement our deception server. We tested our implementation in Mininet [5] which is the current state-of-the-art SDN network emulator. Our SDN environment supports OpenFlow [8] 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.

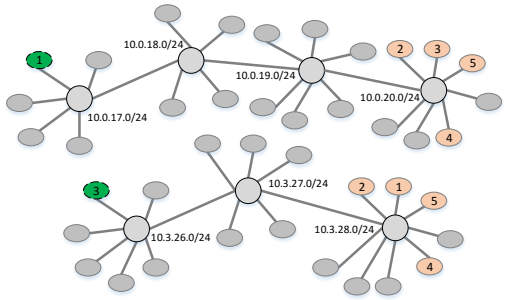

Figure 3: Virtual Network view from the perspective of node 1 (top) and from the perspective of node 3 (bottom).

3.2 Virtual network view generator

As we discuss in Section 2.3, 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. In the description file of a virtual network view it can also be specified if a real host in 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.

As a simple example of a virtual topology, 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. Nodes 2, 4 and 5, which appeared to be in the same subnet as 3 before, are in a different subnet 2 hops away from node 3 in this virtual network view. The underlying real network topology is unchanged and contains all nodes, 1-5, in one subnet.

An adversary probing the network from host 1 and from host 3, would discover the other vulnerable nodes with a significant delay since it first has to search through different subnets. The performance of legitimate network traffic with virtual network views is approximately identical to the underlying physical topology, as we will show in detail in Section 5. We do not simulate delay on virtual network paths since it would penalize legitimate traffic. We are aware that reconnaissance missions trying to map a network based on features such as link delay are theoretically possible. In such cases, our system is capable of simulating artificial delay by forwarding a packet that passes through an SDN switch to the controller and holding it for a specific duration. We will discuss this in more detail in Section 6.

To generate virtual network views, our system creates a machine readable description of a network topology. This description is generated internally and only accessible to the SDN controller and the deception server. For the generation of virtual network views, the features NF as defined in Sec-

tion 2.3 are considered. We derive the following list of input parameters and effects which are specified in a virtual network view to transform the set of features NF into NF':

- 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 $NH \rightarrow NH'$, $TL \rightarrow TL'$

Specific user policies and access rules as well as strategy specifications are considered for the defense against cyber attacks using reconnaissance as an attack planning step. The different strategies for the placement of vulnerable hosts include (i) random placement, (ii) placement in a subnet with high hop and address distance from the view node, and (iii) a placement to create a uniform distribution of nodes across the simulated subnets. The selection of deceptive IP addresses for nodes in a virtual network view can be done randomly within address bounds, or can be matched to specific defense requirements. For the generation of virtual network topologies we also consider critical locations of links and nodes, as discussed in [33, 19]. Our network view generator ensures that vulnerable nodes are not placed in critical locations of a network topology where they have a higher likelihood of being attacked. In particular, a machine readable topology description contains the following information:

- Virtual network view node specification
- Port and address information of the deception server
- Real IP and MAC addresses of visible real hosts
- Real IP and MAC addresses of honeypots
- Deceptive IP addresses of real hosts and honeypots, as they appear to the view node
- Virtual path information to real hosts and honeypots

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

3.3 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 3.3.1. 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 bleed 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 3.2, 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 3.4. 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. By performing dynamic address translation we ensure a clear separation of a virtual network, which is only visi-

ble to a specific host, and the remaining underlying real network.

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 4.4.

3.3.1 SDN rule scalability

Since a core part of our defense approach is a sophisticated system of flow rules, we analyze the scalability of the number of required SDN sules. Contrary information is found in the research community versus industry about the number of rules modern SDN switches can support. While sources such as [20, 7] claim that 64k-512k flow rules can be handled by a modern SDN switch, SDN hardware vendors such as Brocade state that most of their products only support a maximum number of 3k flow rules while their heavy duty products support 12k-65k flow rules [3, 4]. Further, it is unclear how a growing number of flow rules impacts the performance of a SDN switch; contradictary information is provided by different hardware vendors.

Considering the differences in rule scalability of different SDN network elements, we decided to implement the RDS SDN controller with a re-active approach so that flow rules are dynamically generated and automatically removed from the switch after an idle timeout threshold. Upon the arrival of a packet in a SDN switch, the packet is matched with the flow rules currently present in the switch memory. If no appropriate match is found, the packet is forwarded to the controller by default. If a packet arrives at our SDN controller, the packet meta-information is compared to the policy specified in the network view description and a flow rule is generated, pushed into the switch and the packet is appropriately forwarded. All following packets belonging to this flow will match the generated flow rule and can be handled directly on the switch.

Such a re-active approach is currently used by most off-the-shelf SDN controller implementations such as POX, OpenDayLight or Floodlight. In our implementation we set the default idle rule timeout to 30 seconds. This threshold can be adjusted appropriately by a network operator to ensure that flow rules are removed fast enough to release space in the switch memory after they are idle for a while. In Section 5 we evaluate that only a negligible delay overhead is introduced by re-active rule generation and show data of the number of flow rules stored in a SDN switch memory per unit of time during the operation of our deception system.

The benefit of a pro-active approach, where all flow rules are pre-calculated and directly pushed into a switch, would be a slightly decreased network latency during operations. We calculated that simulating a virtual network view with 10 subnets connected in a line with 50 nodes per subnet for one client-host in the network, would require 3762 flow rules. This shows that a pro-active approach is not sustainable in our system due to the high number of required SDN flow rules to cover all network functionality. This is the case for most SDN applications, therefore pro-active SDN approaches where all rules are pre-calculated and directly stored in the switch, are rarely seen in real world implementations.

3.4 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 specification of a virtual network view. The complexity level of the deception server is kept simple. 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. Our deception system has a pre-defined list of nodes and their privileges which are reflected in the description file of a virtual network view. If a device sends a request for a DHCP lease to our deception server, the deception server looks up the node privileges and notifies the view generator which 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 network 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 virtual network 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. With this functionality our deception system has the ability to simulate paths over multiple hops between two nodes in a virtual network view.

With the introduced functionality of our deception server in collaboration with the SDN controller, RDS is able to simulate complete virtual network topologies and deceive malicious reconnaissance and discovery.

3.5 System prototype

We release a proof of concept prototype implementation of RDS at [12] as open-source. The protoype 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. We are in contact with a company that is interested in the implementation of such a system as an actual product.

4. EVALUATION

In the following we present our test network where we evaluate a number of scenarios to show that RDS can be applied as a defense strategy and show experimental results about its performance.

4.1 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 4. Our test network consists of multiple connected subnetworks, where the nodes in each subnetwork are connected with an instance of OpenVSwitch [9] which is controlled by a SDN controller. To emulate the functionality of our test-network we use Mininet [5]. 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 4), 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. The simulation of virtual network views for hosts does not impact the functionality of services provided over the network besides a minor performance overhead as we further discuss in Section 5.

In Figure 4 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 4 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 (since our deception system has no influence on the underlying core network).

4.2 Invalidation of attacker information

Many targeted cyber attacks, such as Advanced Persistent Threats (APT) [1, 2, 15, 42], 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 Seciton 2.1) for further attack planning.

As discussed in Section 3, 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 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 [6] 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* [17]. *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.

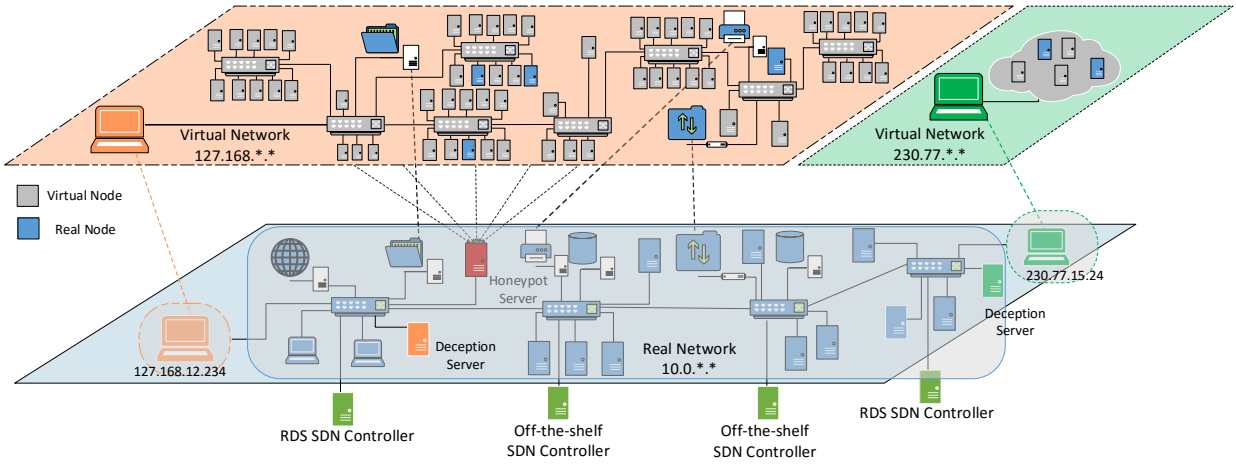

Figure 4: Emulated enterprise test network

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.

4.3 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 [39, 37, 16, 23, 22]. To implement these scanning strategies we used the python library *libnmap* [11], which provides an API to NMAP [6], as well as the python framework Scapy [13].

As discussed in [39], 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 (δ)*, 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 probe transmitted by a scanner 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 [22] and [39], 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 [39]. 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 $\delta(h)$, 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 $\delta(h)$ 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.

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.

4.3.1 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 4. 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 5 and 6 we present experimental results when the introduced adversarial scanning techniques are applied in simulated virtual networks. In the shown charts, we

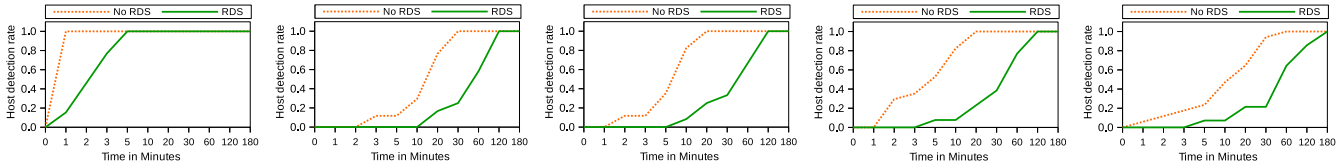

Figure 5: Average vulnerable host infection rate over time for the scanning strategies *Preference Parallel*, *Local Preference*, *Preference Sequential*, *Non-Preference Sequential*, *Uniform* with and without our deception system. Vulnerable hosts are distributed evenly over the address space in a virtual network view with 12 simulated virtual subnets.

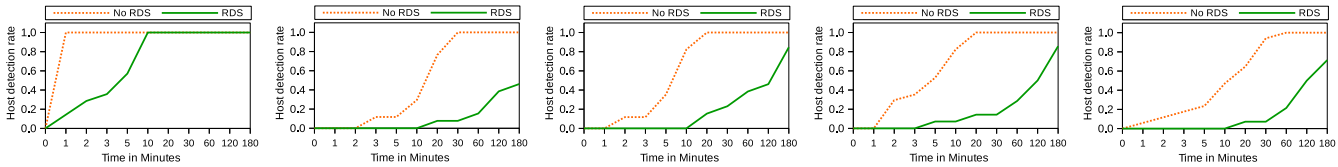

Figure 6: Average vulnerable host infection rate over time for the scanning strategies *Preference Parallel*, *Local Preference*, *Preference Sequential*, *Non-Preference Sequential*, *Uniform* with and without our deception system. Vulnerable hosts are distributed evenly over the address space in a virtual network view with 25 simulated virtual subnets.

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 5 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 shown later in Section 4.4.

In Figure 6 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 2.3. 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.

4.4 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 SDN flow statistics have already been proven as being an efficient technique for real time detection of anomalies as discussed in [21].

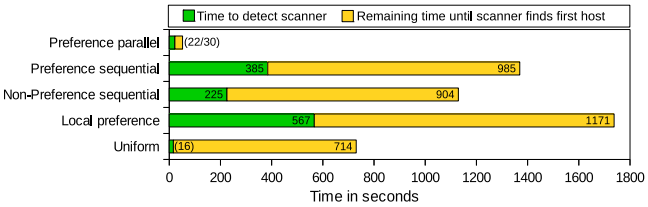

Figure 7: Average time to detect a malicious scanning source and remaining time until the first vulnerable host is identified by a scanner in a virtual network with 12 subnets and 25 hosts per subnet

In Figure 7 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 5.

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 [25]. As we present in Figure 7, 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.

5. EVALUATION OF RDS OVERHEAD

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.

To analyze the latency overhead reconnaissance deception introduces into a network, we measure the response times of hosts connected with a virtual network topology and compare it to response times of the same hosts, without the usage of RDS. We also measure the number of OpenFlow rules in the switch memory per unit of time when RDS is deployed.

In a SDN network that follows a re-active approach, the biggest cause of delay compared to legacy networks, is the dynamic construction of flow rules upon the arrival of a packet that has no rule match in the current switch memory. This delay is usually of the order of milliseconds and is experienced by the first few packets of a flow. To evaluate the overhead in terms of the delay of our system we record the response times of the first 100 packets of flows when our system is deployed and simulates different sizes of virtual networks, and compare it to the delay of the first 100 packets when an off-the-shelf SDN learning switch (we used the layer 2 learning switch of the POX framework) is used in the same networks. On average, we measured an increased response time of 0.2 milliseconds per packet flow when our deception system simulates a virtual network topology. This slightly increased initial delay of a flow can be explained by the lookup operation that has to be performed on the virtual network view description by our SDN controller for the construction of a new flow rule, rather than constructing flow rules only based on packet header information. We also measured the additional time overhead that is caused by the dynamic rewriting of packet headers with SDN flow rules. This delay overhead is only of the order of microseconds and is therefore negligible.

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 8 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 the 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 3.3.1, a number of SDN rules of this magnitude can be handled by modern SDN switches and should not cause any noticeable performance impact.

Besides that, we are not expecting to see a significant performance overhead since our deception system only manipulates the network traffic in order to deceive malicious scanners, but does not emulate the physical characteristics of a virtual network topology. This means that the paths to hosts which appear to be separated by multiple hops in the virtual topology, actually show the same network performance as in the real underlying topology. In our solution we do not introduce artificial delay on virtual paths, which would penalize legitimate traffic on the network. We discuss the pros and cons of this design decision in Section 6.1.

The overhead for assigning a new DHCP lease to a connected host, which involves the generation and deployment of a new virtual network view, can be done in the order of seconds.

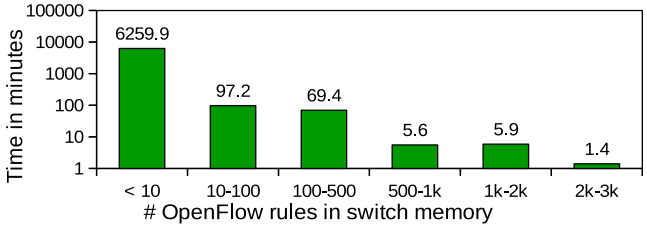

Figure 8: Average number of OpenFlow rules in a SDN switch per minutes of system operation time over 100 hours

5.1 Comparison to current defense approaches

Current approaches to defend against adversarial scanning 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. One of the most recent defense systems is discussed in [23] and its successor publication [22]. Here the authors follow a moving target defense approach and perform very fast randomization of the address space (every 1-5 seconds) to match the scanning rate of adversaries like computer worms. This approach is effective against known scanning strategies (besides uniform scanning approaches), but introduces significant performance overhead. In particular, the introduced system in [22] 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. The authors of [22] omit detailed results for brevity about the performance of their system. 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 with the deployment of a new virtual network view which is typically done every few hours with the assignment of a new DHCP lease. Considering this, the proposed system in [22] will have a significant performance impact, that we are able to avoid in our solution.

In [22] Jafarian et al. show the effectiveness of their defense system in the first 1000 seconds after an adversary starts the scanning process. Within this time frame our system shows equal or better performance, depending on the defense strategy used in virtual network views. In Section 4.4 we also 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 advan-

tage over current approaches by minimizing the performance overhead while effectively defending and identifying an adversary performing reconnaissance.

6. DISCUSSION

6.1 Design decisions

In this section we discuss the design decisions we made for the defense of malicious scans with virtual network views. One can argue that an adversary with insider information can adjust a scanning strategy with counter active measures against our defense approach. This may involve scanning the network with a start address that has a high address distance δ and probe the address space in reverse. If we place all vulnerable hosts with a high address distance from the view node, this would be effective for an attacker. In such a case, we are still able to use a defense strategy where vulnerable hosts are distributed evenly over multiple subnets. With this approach, scanning the address space in reverse would not help an attacker, and we would still be able to delay the detection time of vulnerable hosts as we show in the results in Figure 5 and 6.

There are certain legitimate tasks which rely on the knowledge of the real network topology for administrative purposes. For example certain schedulers select nodes with close proximity for load distribution. For hosts performing tasks that rely on the true view of a network, no virtual network view should be assigned to such a node as we discuss in Section 2.3. Network administrators, have to identify such hosts prior to deploying our system, since the assignment of a virtual network view would interfere with tasks relying on the knowledge of the true underlying network.

Another important design decision we made is not to simulate the physical characteristics of virtual network paths by introducing artificial delay. As we explain in Section 3.2, our system is capable of simulating delay on links but we choose not to, since it would penalize legitimate traffic. One can think of an advanced adversary who determines that deception is used in a network by comparing the observed topology to its physical characteristics. In such a case an adversary still has to scan the whole virtual network since the locations of vulnerable hosts are unknown, and experience the same host detection delays as an attacker who is not aware of the deception system. Security measures always come with some cost; the simulation of physical characteristics in virtual network views would result in an almost perfect deception of a network topology, but would also come with a higher price regarding to performance. In the design decisions we made in our system we try to find a good balance for the trade-off between security and performance.

6.2 Adaptive adversaries

Considering an advanced adaptive attacker who stays on a host and records a number of virtual network views, we ask the question if it would be possible to compare these views, estimate the used defense strategy and determine real and virtual components. In case of a coordinated adversary, who compromises multiple hosts, collects their views and compares them it is theoretically possible that such a colluding attacker can conclude that it has been deceived. In such a case, an attacker would still not be able to make conclusions about the underlying system, since it cannot be determined what information in the provided set T' is fake and what is real. Assuming such an attacker, the generation of virtual network views has to be done with caution, such as alternating different host placement strategies. We leave a detailed analysis of adaptive coordinated adversaries for future work, but argue that our system is able to defend against such an attacker, since we show that malicious scanning hosts can be detected and isolated before they find vulnerable hosts.

7. RELATED WORK

In recent publications [22, 23, 16], the authors introduce a system that performs dynamic address space randomization based on Software Defined Networking (SDN) technologies to defend against adversarial scanners, such as worms. The evaluations in these papers show that dynamic address space randomization is an effective defense technique against reconnaissance strategies such as local-preference or sequential scanning. The authors of [22, 23, 16] observe that their technique is ineffective against a uniform scanning strategy. Although they do not give details about their system overhead, we show in Section 5.1 that the performance overhead of their system is significant. Our RDS minimizes the overhead by using complete virtual network topologies as a defense strategy. We also show that our system is effective against all scanning techniques including uniform scanning.

In [36] the authors propose techniques to deceive network reconnaissance focused on mapping a network topology by using the standard *traceroute* function. The authors especially focus on the defense of critical routers and links in a network topology. They propose defense techniques following a random and an intelligent approach to deceive network scans. The authors give little details about the evaluated results of their system and do not show how this technique can be used to defend against an actual cyber attack depending on topology mapping.

The authors of [31] discuss a SDN based approach for cyber deception. In the paper, an overview of the design of a cyber deception system is presented, but no specific defense strategies on attacks depending on reconnaissance missions are evaluated.

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 [41] and [30].

We consider a number of well cited publications [37, 39, 26] 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 for network reconnaissance as discussed in these papers.

8. CONCLUSION

In this paper we define a deception approach as a defense strategy against network reconnaissance and introduce the design and implementation of a Reconnaissance Deception System (RDS). Our solution is able to manipulate network traffic to show a virtual topology to an internal adversary gathering information inside organizational networks, required for planning and executing of targeted cyber attacks such as Advanced Persistent Threats (APT). 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 experimental evaluation we present in this work shows that our system is able to delay malicious network scans up to a factor of 115, while minimizing the performance impact on legitimate traffic on the network to 0.2 milliseconds per packet flow.

Acknowledgment

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 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.

9. REFERENCES

[1] Advanced persistent threats - attack and defense (infosec institute. http://resources.infosecinstitute.com/advanced-persistent-threats-attack-and-defense/. Accessed: 2016-04-01.

[2] Advanced persistent threats (2011). http://www.symantec.com/content/en/us/enterprise/white_papers/b-advanced_persistent_threats_WP_21215957.en-us.pdf. Accessed: 2016-03-30.

[3] Brocade flow optimizer. http://www.brocade.com/content/brocade/en/backend-content/pdf-page.html?/content/dam/common/documents/content-types/user-guide/flowoptimizer-1.1.0-userguide.pdf. Accessed: 2016-03-27.

[4] Fastiron ethernet switch software defined networking (sdn). http://www.brocade.com/content/dam/common/documents/content-types/configuration-guide/fastiron-08030-sdnguide.pdf. Accessed: 2016-03-27.

[5] Mininet - realistic virtual sdn network emulator. http://mininet.org/.

[6] Nmap network scanner. https://nmap.org/.

[7] Openflow - can it scale? www.sdxcentral.com/articles/contributed/openflow-sdn/2013/06/.

[8] Openflow protocol. https://www.opennetworking.org/sdn-resources/openflow.

[9] Openvswitch. http://openvswitch.org/.

[10] Pox - python based sdn controller framework. http://www.noxrepo.org/pox/about-pox/.

[11] Python library for the nmap network scanner. https://libnmap.readthedocs.org/.

[12] Reconnaissance deception system prototype implementation. https://github.com/deceptionsystem/master.

[13] Scapy - python framework for packet crafting and manipulation. http://www.secdev.org/projects/scapy/.

[14] Stages of a cyber attack. http://www.symantec.com/content/en/us/enterprise/other_resources/b-preparing-for-a-cyber-attack-interactive-SYM285k_050913.pdf. Accessed: 2016-03-30.

[15] Eric Baize. Developing secure products in the age of advanced persistent threats. *IEEE S&P 2012*.

[16] Al-Shaer et al. Random host mutation for moving target defense. In *Security and Privacy in Communication Networks*. 2013.

[17] Alt et al. Uncovering network tarpits with degreaser. In *Proceedings of the 30th Annual Computer Security Applications Conference*, 2014.

[18] Antonatos et al. Defending against hitlist worms using network address space randomization. *Computer Networks*, 2007.

[19] Arulselvan et al. Detecting critical nodes in sparse graphs. *Computers & Operations Research*.

[20] Chiba et al. Source flow: handling millions of flows on flow-based nodes. *ACM SIGCOMM 2011*.

[21] Giotis et al. Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments. *Computer Networks*, 2014.

[22] Jafarian et al. Adversary-aware ip address randomization for proactive agility against sophisticated attackers. In *INFOCOM 2015*.

[23] Jafarian et al. Openflow random host mutation: transparent moving target defense using software defined networking. In *HotSDN 2012*.

[24] Kreutz et al. Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013.

[25] Kuzniar et al. What you need to know about sdn flow tables. In *Passive and Active Measurement*, 2015.

[26] Li et al. Understanding divide-conquer-scanning worms. In *Performance, IPCCC 2008*.

[27] McClure et al. Hacking exposed: network security secrets and solutions. 2009.

[28] Panjwani et al. An experimental evaluation to determine if port scans are precursors to an attack. In *Dependable Systems and Networks, DSN 2005*.

[29] Porras et al. A security enforcement kernel for openflow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012.

[30] Provos et al. A virtual honeypot framework. In *USENIX Security Symposium 2004*.

[31] Robertson et al. Cindam: Customized information networks for deception and attack mitigation. In *SASO Workshop, 2015*.

[32] Shaikh et al. Network reconnaissance. *Network Security*, 2008.

[33] Shen et al. On the discovery of critical links and nodes for assessing network vulnerability. *IEEE/ACM Transactions on Networking (TON)*, 2013.

[34] Shin et al. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*.

[35] Sood et al. Targeted cyberattacks: a superset of advanced persistent threats. *IEEE security & privacy*, 2013.

[36] Trassare et al. A technique for network topology deception. In *MILCOM 2013*.

[37] Weaver et al. A taxonomy of computer worms. In *Proceedings of the 2003 ACM workshop on Rapid malcode*.

[38] Zaalouk et al. Orchsec: An orchestrator-based architecture for enhancing network-security using network monitoring and sdn control functions. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*.

[39] Zou et al. On the performance of internet worm scanning strategies. *Performance Evaluation*, 2006.

[40] Erwan Le Malcot. Mitibox: camouflage and deception for network scan mitigation. In *HotSec 2009*.

[41] Niels Provos. Honeyd-a virtual honeypot daemon. In *10th DFN-CERT Workshop, Hamburg, Germany*, 2003.

[42] Andrew Vance. Flow based analysis of advanced persistent threats detecting targeted attacks in cloud computing. In *Problems of Infocommunications Science and Technology, 2014 First International Scientific-Practical Conference*.