Hindawi
Security and Communication Networks
Volume 2017, Article ID 3659167, 11 pages
https://doi.org/10.1155/2017/3659167

WILEY | Hindawi

*Research Article*

# CHAOS: An SDN-Based Moving Target Defense System

**Yuan Shi,[1] Huanguo Zhang,[1] Juan Wang,[1] Feng Xiao,[1] Jianwei Huang,[1] Daochen Zha,[1] Hongxin Hu,[2] Fei Yan,[1] and Bo Zhao[1]**

[1]*Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, Computer School of Wuhan University, Wuhan, China*
[2]*Division of Computer Science, School of Computing, Clemson University, Clemson, SC 29634, USA*

Correspondence should be addressed to Juan Wang; jwang@whu.edu.cn

Received 2 August 2017; Accepted 11 September 2017; Published 16 October 2017

Academic Editor: Zhiping Cai



Moving target defense (MTD) has provided a dynamic and proactive network defense to reduce or move the attack surface that is available for exploitation. However, traditional network is difficult to realize dynamic and active security defense effectively and comprehensively. Software-defined networking (SDN) points out a brand-new path for building dynamic and proactive defense system. In this paper, we propose CHAOS, an SDN-based MTD system. Utilizing the programmability and flexibility of SDN, CHAOS obfuscates the attack surface including host mutation obfuscation, ports obfuscation, and obfuscation based on decoy servers, thereby enhancing the unpredictability of the networking environment. We propose the Chaos Tower Obfuscation (CTO) method, which uses the Chaos Tower Structure (CTS) to depict the hierarchy of all the hosts in an intranet and define expected connection and unexpected connection. Moreover, we develop fast CTO algorithms to achieve a different degree of obfuscation for the hosts in each layer. We design and implement CHAOS as an application of SDN controller. Our approach makes it very easy to realize moving target defense in networks. Our experimental results show that a network protected by CHAOS is capable of decreasing the percentage of information disclosure effectively to guarantee the normal flow of traffic.

## 1. Introduction

Nowadays, the network security issues become increasingly prominent as all kinds of network security events emerge one after another. However, the traditional network security tools cannot effectively defend increasingly complex and intelligent penetration of network intrusion and unknown vulnerability attacks. As usually, adversaries can break through or bypass firewalls and intrusion detection systems (IDS) so that an intranet can be easily compromised. As one of revolutionary technologies, Moving Target Defense (MTD) changes game rules, providing a dynamic and proactive network defense [1–3].

MTD aims at building a dynamically and continually shifting and changing system to increase complexity and cost for attackers, limit the exposure of vulnerabilities and opportunities for attackers, and increase system resiliency [4]. The idea of MTD has been applied to network security, for example, DYNAT [5] and DESIR [6].

The difference between MTD and traditional network tools, such as firewall and IDS, is that the latter will suspend suspicious actions once they break security rules. That makes it easy for adversaries to figure out the deployed network defense mechanism so that they will try to bypass them. However, MTD sends illegible fake information to potential threats to make them spend more time and cost so that they will leave more footprints, making them easier to be exposed.

However, due to its closed and static characteristics, traditional network is difficult to realize dynamic and proactive security defense effectively and comprehensively. As a new type of network security architecture, software-defined networking (SDN) points a brand-new path for building dynamic and proactive defense system [7, 8]. SDN has a couple of benefits. It decouples network control and data planes, enabling network control to become directly programmable [9]. It enables network managers to configure, manage, secure, and optimize network resources very quickly via dynamic and automated SDN programs [10]. Meanwhile,

SDN lets the underlying infrastructure be abstracted from applications and network services [11]. In addition, SDN controllers can provide a global view of the network. The central management of SDN makes networks more intelligent.

Therefore, our goal is to build an SDN-based dynamic network defense system. In order to realize the SDN-based MTD, it has some key challenges to be resolved. Firstly, we should leverage SDN to obfuscate network fingerprinting. Secondly, the moving target defense may make some networks services unavailable, such as database server. The IP address and port number of these services have to be opened to the outside and remain real. If MTD obfuscates these services fully, it will return users with fake IPs and ports, making these services unable to be used. Thirdly, obfuscating network parameters indiscriminately will severely reduce the performance of networks undoubtedly.

Motivated by the aforementioned goals and challenges, we propose CHAOS, a SDN-based MTD system. Utilizing the programmability and flexibility of SDN, CHAOS obfuscates the attack surface including host mutation obfuscation, ports obfuscation, and obfuscation based on decoy servers thereby enhancing the unpredictability of the networking environment. Furthermore, it discriminately obfuscates hosts with different security levels in networks. In CHAOS, we propose the Chaos Tower Obfuscation (CTO) method, which uses the Chaos Tower Structure (CTS) to depict the hierarchy of all the hosts in an intranet and define expected connection and unexpected connection. Moreover, we develop fast CTO algorithms to achieve a different degree of obfuscation for the hosts in each layer. We design and implement CHAOS as an application of SDN controller. Our approach makes it very easy to realize moving target defense in networks.

Furthermore, we evaluate our system and the results show that CHAOS can effectively hide real information of the target hosts from attackers and produce fake responses, which can disrupt an adversary's ability to sniff network traffic effectively. In addition, our tests show that the system has lower cost when compared with a fully obfuscated system, which strengthens its applicability in real networks.

Our contributions can be summarized as follows:

(i) We propose a new SDN-based MTD approach, CHAOS, where a Chaos Tower Structure (CTS) is constructed to represent a hierarchy of all the hosts in the network. Using the CTS, we can determine if a network connection is needed to be obfuscated.

(ii) We present a more unpredictable and flexible obfuscation method named Chaos Tower Obfuscation (CTO) in CHAOS, where the level of obfuscation is decided reasonably. Furthermore, through using host mutation obfuscation, ports obfuscation, and obfuscation based on decoy servers, CHAOS can flexibly forward and modify the packets in a network to obfuscate the attack surface.

(iii) We design and implement CHAOS as an SDN application and evaluate its performance. The results demonstrate that a network protected by CHAOS can decrease the percentage of information disclosure effectively and has a lower cost.

(iv) CHAOS is designed and implemented as an application of SDN controller and works with IDS that lets it very easy to realize moving target defense in networks, so CHAOS not only solves the key issues of building a practical SDN-based MTD system, but also can be used in the real-world systems instead of a theory model.

The remainder of this paper is organized as follows. Section 2 provides some background information relating to our system. Section 3 describes how we design our system. Section 4 shows the details of CHAOS obfuscation methods. Section 5 presents the implementation and evaluation of our system. Section 6 shows some related work. Section 7 concludes this paper.

## 2. Background and Threat Model

In this section, we first provide an introduction to SDN and its mechanism of asynchronous messaging. Then we introduce a threat model about our system.

*2.1. SDN and Its Asynchronous Messaging Mechanism.* SDN has emerged as a programmable and centrally controlling architecture providing an agile platform for vendors as well as enterprise users to control and define network.

The SDN controller plays the role of an operating system (OS) for networks [11]. All communications between network applications and network devices have to go through the controller. OpenFlow protocol as the first SDN standards defined the communication protocol between the SDN controller and the forwarding plane of network devices such as switches and routers. The controller uses the OpenFlow protocol to control network devices and choose the best path for application traffic. Because the network control plane can be programmed, contrary to the firmware of hardware devices, network traffic can be managed more dynamically and at a much more granular level.

Centralized control allows the SDN core controller to define the data flows [1]. Each flow through the network must first get permission from the controller, which verifies that the communication is permissible by the network policy [12].

*Flow Table.* The OpenFlow switch (OF switch) contains the flow tables, which are used to perform packet lookups and forwarding [12]. Using OpenFlow protocol, the controller can add, update, and delete flow entries in the flow table, both reactively (in response to packets) and proactively [12]. Each flow table in the switch contains a set of flow entries. Each flow entry consists of matching fields, counters, and a set of instructions to apply to matching packets [1]. If a packet matches the fields defined in the flow table, the instructions (i.e., "actions") are executed. If no match is found, a packet may be forwarded to the controller or continue to the next flow table.

*Packet-In Message.* For all packets that do not have a matching flow entry, a packet-in event may be sent to the controller. There are mainly two situations that produce these messages: a mismatch in the tables of the switch or a time to live

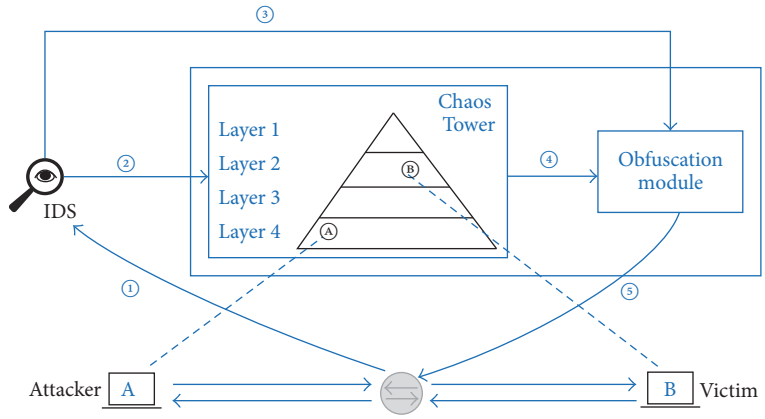


Figure 1: Overall system of CHAOS. ① shows that IDS is monitoring the flows of OF switch; ② and ④ shows that whether normal connections detected by IDS will be obfuscated or not is determined by Chaos Tower; ③ means the abnormal connections detected by IDS will be obfuscated directly; ⑤ means that the obfuscation is executed by OF switches.

(TTL) error [13]. Packet-in messages contain a variety of information about the flow.

After receiving the packet-in message, the core controller decides how to process irregular flows by dispatching a packet-out message.

*Packet-Out Message.* Packet-out messages are sent from the controller to a switch when the controller wishes to instruct the switch to send packets via a specified port of the switch or to instruct the switch how to forward packets received via packet-in messages.

In CHOAS, we use SDN features and its asynchronous messaging mechanism to implement our dynamic and proactive defense system.

*2.2. Threat Model.* In most cases, adversaries start an attack on an intranet by collecting as much information about the network as they can. Then they connect to those vulnerable hosts and send attack payloads. Our system, CHAOS, aims to build a dynamic and variable network, so as to defeat reconnaissance attacks on an intranet. Thus, we assume an adversary can scan a network and monitor the network traffic. Moreover, the adversary can eavesdrop network packets. We also assume the protected networks are able to support OpenFlow-based SDN switches and controllers.

## 3. CHAOS Design

In this section, we provide an overview of CHAOS and then highlight the design of Chaos Tower Structure (CTS).

*3.1. CHAOS System Overview.* The overall system is illustrated in Figure 1. We design two main modules: Chaos Tower Structure (CTS) and Chaos Tower Obfuscation (CTO) module. CTS defines the communication rules of hosts in a network. The communications that break the CTS rules will be obfuscated using CTO that implements obfuscation mechanisms. We do not obfuscate all network traffic because it will dramatically degrade network performance. In CHOAS, the network traffic will be first sent to IDS, such as Bro. If IDS judges that the traffic is suspicious, CTO module will obfuscate them through installing new flows into OpenFlow switches or modifying flows. Otherwise, if the traffic is judged normal, it will be redirected to our Chaos Tower Structure module. The reasons for doing this are that adversaries often can bypass IDS through some unknown vulnerability attacks. CTS judges the risk of flows and divide them into expected connections and unexpected connections, detailed in Section 3.2.1. Expected connections will be allowed. The unexpected connections will be obfuscated by obfuscation module according to different obfuscation levels.

*Chaos Tower Structure (CTS).* It is the module we design in the system to determine the communication rules. CTS builds a host hierarchy according to security level of information assets. The tower consists of several layers. Generally, important workgroups are placed in higher layers, whereas unimportant workgroups are placed in lower layers. The importance of every single node which can correspond to a host as well as the host cluster is determined based on the importance degree of services and the vulnerability assessment score in the node. Then we build our model to control network traffic by defining which pairs of hosts can communicate in our topology. Further, according to the tower, the system divides connections into two types: expected and unexpected connections.

*Chaos Tower Obfuscation (CTO).* It works on the basis of the CTS. It will obfuscate the suspicious connections detected by IDS and unexpected connections detected by CTS. Those connections will be divided into corresponding obfuscation levels. Then CTO obfuscates the connections according to the level.

We next elaborate the major processes of the whole system as shown in Figure 1. If an attacker tries to launch a request from a workgroup in relatively lower layers to a workgroup in higher layers, as indicated by A and B in Figure 1, the system examines the corresponding connection. Firstly, the IDS detects the request and then determines

whether it is a normal connection (Line ①). If it is suspicious, the connection will be directly obfuscated directly (Lines ③ and ⑤). Otherwise, CTS starts to work (Line ②). As shown before, CTS will judge the connection according to its rules. Once the connection is judged to be unexpected by CTS, it will be obfuscated by CTO (Lines ④ and ⑤). In Figure 1, the request is unexpected; as a result, the connection will be obfuscated and B is protected from being scanned or attacked.

*3.2. Chaos Tower Structure and Its Workflow.* The CTS is a combination of a tree structure and an oriented graph structure. We use a multibranch tree in which to store the workgroup (a host is assigned to a specific workgroup according to its function or importance degree) and the tree defines the privilege of every workgroup. This ensures that most of the layer-jumping behavior is obfuscated. Nonetheless, some layer-jumping behavior is necessary (e.g., the two-way communication between a web server and a database server is necessary, although they are in distinct workgroups). We can define or modify the information conveniently by editing the "Chaos Tower configure file" in the controller to add the special rules. The tower structure with its strict hierarchy enables a more secure and more reliable network.

*3.2.1. Tower Construction.* In CHAOS, every host or subnet group will be examined and thus a corresponding risk level will be calculated. Risk levels are based on the underlying security metrics. In our system, we use the base score of Common Vulnerability Scoring System (CVSS) [14] to determine the intrinsic qualities of vulnerability. CVSS base score includes two factors, *exploitability of vulnerability* and *impact of vulnerability*. CVSS classifies all the vulnerabilities depending on their features and effects and thus concludes several different kinds of vulnerabilities, such as SQL injection and buffer overflow. For all these kinds of vulnerabilities, CVSS assigns different score to signal the importance of the vulnerability. And in addition to CVSS score, another critical factor is service importance value (SIV). Normally, some hosts are more valuable than others. Thus, we adopt service importance value to represent service's inherent value. It is worth mentioning that, in different networks, the same service may be valued different. That is the reason why we set the SIV table as a part of configuration that administrators should define before the system works. In our system, we introduce the following generic equation to incorporate the CVSS base score and service importance value:

$$\mathrm{RL}(h) = \sum_{v \in V(h)} (\alpha \times \mathrm{SIV}(s) + (1 - \alpha) \times \mathrm{CVSS}(v)), \quad (1)$$

where RL($h$) is the risk level of node $h$; $V(h)$ is a function to return all vulnerability contained in the host $h$; SIV($s$) is a function to return the service importance value of the service $s$; and CVSS($v$) is a function to return the CVSS base score of the vulnerability $v$. We also introduce the weight coefficient $\alpha$ ($0 \leq \alpha \leq 1$) that allows an administrator to determine how important the service is. The value $\alpha$ can be increased, in which case the service is more important. Otherwise, the administrator can decrease the value of $\alpha$ to weaken the influence of the service but emphasize the influence of the possibility that the hosts would be attacked. According to this given information, we can continue building the original tower, which contains several layers. Each of these layers contains several workgroups, each of which includes several hosts that provide similar functions. CTS also can use some weights such as time, to further define access rules. For example, some access requests can be only allowed in some periods.

After the risk level of each hosts or groups is calculated, we put them into different layers of Chaos Tower. Hosts in the same layers should have the same risk level. Layers with higher risk level will have higher position (e.g., database servers). To deal with the situation that many new devices might well be added to specific subnets, we further divide hosts in the same layer into several groups. Each group contains at least one host. The group division is dependent on the hosts distribution in physical networks. Hence, when there are new devices added to the tower, CHAOS first exams whether they can belong to one existing group or not, if not, its risk level will be calculated and thus it will be mapped onto a new group in the corresponding layers.

In CHAOS, we deem that the more important and risky the host is, the higher the layer it is assigned to. These groups share some common traits; for example, they may be used to store some important network resources. In our system, the administrators can define those important hosts and specify their order of privilege by the risk level of group.

*Expected Connections.* Expected connections include normal connections and special connections:

(i) Normal connections: they represent the connections from higher layers towards lower layers. In CTS, the communication from higher layers to low layers should be allowed because the hosts of high layer are of high risk level. They often provide important service. So these connections correspond to the allowed communications in an intranet. For those which belong to higher layers only because of their high CVSS score, they can be hardly accessed, which indirectly protect them from being attacked. It is worth mentioning that if the connection from A toward B belongs to normal connections, it does not mean that the connection from B toward A belongs to normal connections.

(ii) Special connections: in order to deal with some special communication request, we define the special connections even though the connections where a host belonging to lower layer accesses a host belonging to higher layers are not judged as normal connections. CTS will judge the special connections as expected connections. We can release special connections temporarily and record them in system log so that administrator can carry out the analysis.

*Unexpected Connections.* We define unexpected connections as those connections that are not included in the list of expected connections. Generally, these connections are not

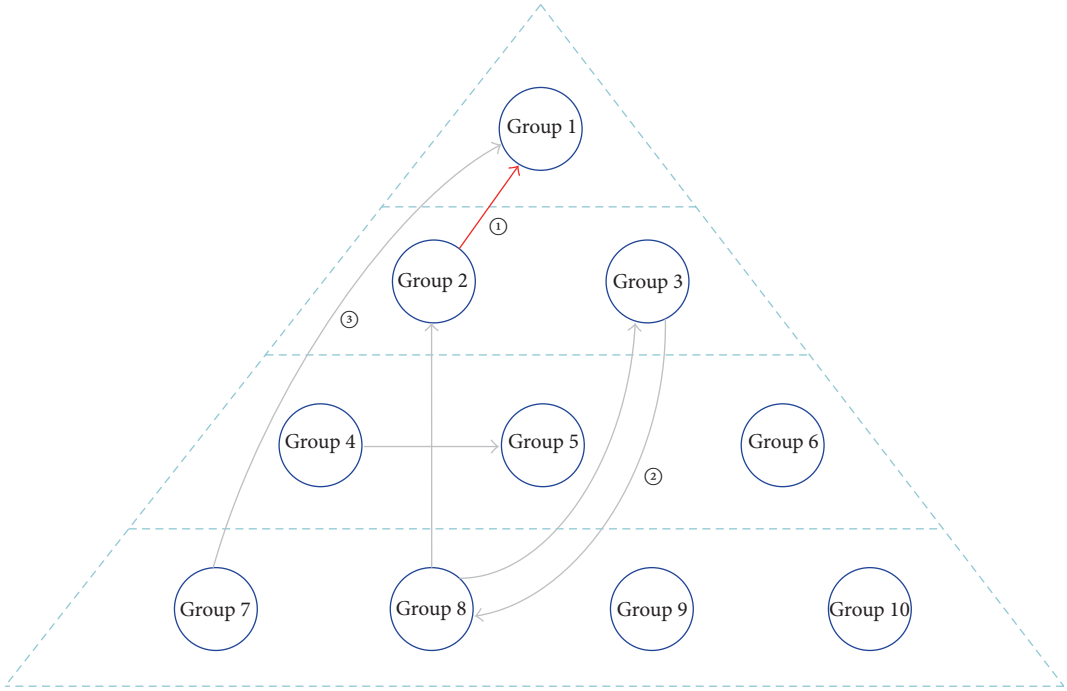


Figure 2: Logical structure of CTS. Red lines like ① represent the unexpected connections; gray lines from upper layers towards lower layers like ② represent the normal connections; gray lines from lower layers towards upper layers like ③ represent special connections.

defined as being allowed and will be detected by our CHAOS system. For example, the connection from a host in employee group toward a host in database group will be judged as unexpected connections.

Here we consider an example to illustrate our proposed CHAOS system in more detail. In Figure 2, Group 1 is placed to the top of tower due to its highest risk level. For line ②, it is a connection from a higher layer to a lower layer, which belongs to normal connections. For line ③, it is a connection from a lower layer to a higher layer but still allowed by CTS, which belongs to special connections. And for line ①, it is an unexpected connection even though it just transgresses only one layer.

*3.2.2. Exploiting the Tower.* The system reacts differently for expected and unexpected connections.

*Expected Connections.* We consider expected connections to be legal; thus, the system does not interfere with these connections.

*Unexpected Connections.* Attention should be paid to these connections. If confronted with an unexpected connection, the controller will send a request to obfuscation module to obfuscate it. Generally, if the connection is established by layer-jumping or occurs within the same layer, it is considered abnormal and will be obfuscated. However, some special connections can be defined by system administrator; these connections cannot be judged as abnormal communication and not be obfuscated.

## 4. Obfuscation

In our system, we implement three kinds of obfuscations, which are host mutation obfuscation, port obfuscation, and obfuscation based on decoy servers. For unexpected connections judged by CTS and the abnormal connections judged by IDS, our system will grade them and apply corresponding obfuscations according to their degree of abnormality.

*Host Mutation Obfuscation.* This technique is aimed to defend MITM (Man in the Middle) attack and third-party traffic monitoring by replacing source IP address and destination IP address of the packet to virtual IP addresses when transferring it between switches [15]. The mechanism is shown in the right-hand side of Figure 3. The OpenFlow controller frequently assigns a random virtual IP (vIP) to each real IP (vIP). When *Host1* initiates the connection to *Host2* and sends an initial packet using real source IP (*r1*) and real destination IP (*r2*), the first OF switch that captures the initial packet *(OF switch 1)* encapsulates and sends the packet to SDN controller, where a rIP-vIP mapping table is stored, and maps *r1* and *r2* to corresponding virtual IPs (*v1* and *v2*). When the initial packet reaches the OF switch that is nearest to *Host2 (OF switch n)*, a similar reverse mapping is executed, changing vIPs back to rIPs, namely, *v1* to *r1* and *v2* to *r2*. In this sense, packets in the middle (between *OF Switch 1* and *OF Switch n*) only contain virtual IPs so that real host IPs are concealed.

*Port Obfuscation.* This technique is aimed to defend port-scanning-based attack. In this case we inject some entirely fake information into responses as well as hiding some real

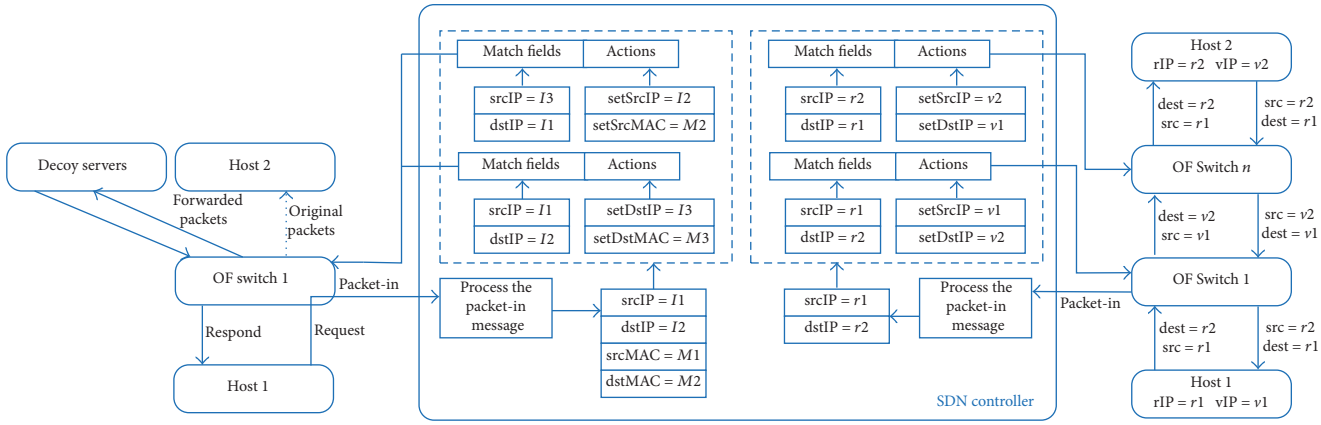


Figure 3: Mechanism of host mutation and decoy-servers-based obfuscation.

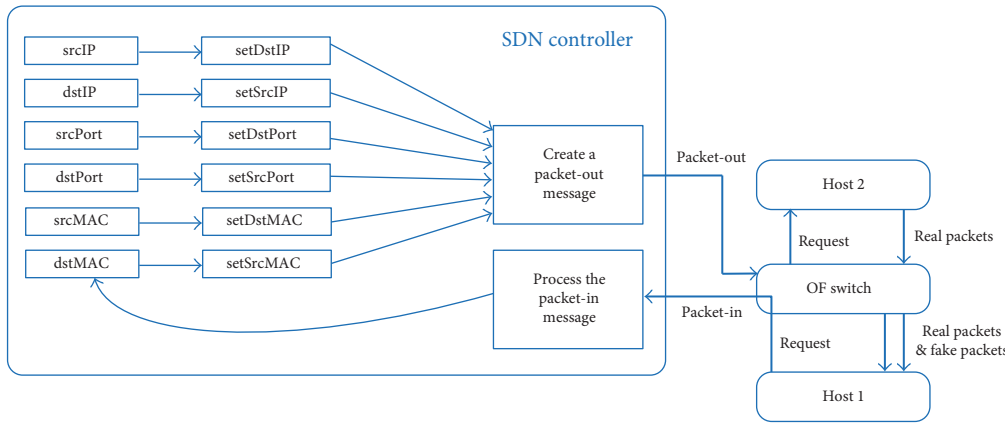


Figure 4: Mechanism of port obfuscation.

information. As is shown in Figure 4, when IDS detects a port scanning, CHAOS system will inject fake packets into the real packets by generating corresponding acknowledgment to obfuscate the result of the port scanning. For instance, when a TCP scan is detected and port obfuscation is applied, the TCP packets will be fetched by switch and sent to the controller through packet-in. Then the controller will analyze the packet, generate a corresponding packet-out, and send it to the switch. The acknowledgments of some injected packets are 0, while some are 1. Whether to inject or modify the packets is generally on a random basis. Therefore, the results of port scanning will show a certain degree of randomness and fuzziness.

*Obfuscation Based on Decoy Servers.* In CHAOS system, we deploy a number of decoy servers as an attack trap. In most cases, decoy servers can even delay the attack. When applying this strategy, our system will forward the unexpected connections to the decoy servers. As is shown in Figure 3, when a host launches a request, our system can analyze the packets and install flows into the switch, which will forward the unexpected connections to our decoy servers. In this way, suspicious users can only access various decoy servers. The services we deployed in the decoy servers can further help us discover the real attackers.

These three obfuscation strategies are applied under different circumstances. In the tower, we use the threshold factor to determine which strategy is applied. It is determined by calculating the ratio of leapfrog access number to the total number of the layers, named altitude. If the altitude of the connection is smaller than threshold, the connection will be obfuscated later. If not, the connection will be forwarded to decoy servers. In short, the threshold factor divides unexpected connections into two parts by altitude. Connections which belong to the first part will be obfuscated, while connections which belong to the other part will be forwarded to decoy servers. Administrators can change the threshold factor depending on the security level and structure of the network. The threshold factor assures that attacks will be obfuscated in theory.

In addition, we introduce a parameter named RandomIndex ($0 \leq \text{RandomIndex} \leq 1$) to define the possibility of CHAOS performing obfuscation; that is, the closer the RandomIndex to 0, the higher the likelihood of CHAOS injecting fake information into the network. We define srcLayer as the layer in which the host launches the request and dstLayer as

```
Require: packetInp, Inf, Sup, RandomIndex; {HEIGHT is the height of the tower}
  if isFromSrcSwitch (p) orisFromDstSwitch (p) then installHostMutationFlows (p);
  end if
  srcLayer ← getSrcLayer (p);
  dstLayer ← getDstLayer (p);
  Δ Altitude ← srcLayer – dstLayer;
  Possibility ← random [0, 1];
  if Δ Altitude ≥ 0 then
    Forward (p);
  else
    Δ Altitude ← −Δ Altitude;
    if Δ Altitude/HEIGHT ≤ threshold then
      if isRequestPacket (p) andPossibility ≥ RandomIndex
      then
        PacketOut (p);
      else
        ForwardToDecoyServer (p);
      end if
    else
    InstallForwardingFlows (p);
    end if
  end if
```

Algorithm 1: CHAOS.

the layer in which the host responds. Then we define altitude as the difference in height between these two respective layers (i.e., the height of srcLayer minus the height of dstLayer). RandomIndex assures that obfuscation is random so that attackers will not notice our system immediately.

Our design of obfuscation contains two aspects. First, as most network mapping tools perform their operations by using ICMP packets and TCP or UDP scans, ICMP messages are typically used to verify connectivity or reachability of potential targets. TCP and UDP port scans are used to identify running services of a target. Replies (TCP RST, silent drop, or ICMP unreachable) to scans can also reveal what services are allowed or filtered through transit devices. Additionally, the TTL field of IP packets is used to identify the hop distance between the target and the destination. SDN-enabled devices can be used to confuse the reconnaissance. For example, traffic to a destination that can be blocked according to a filtering policy can be silently dropped and SDN utilities can generate varying responses that will confuse the attacker. In the case of traffic that is permitted by the filtering policy (that is, it is legitimate), the SDN policy does not interfere. The action for each packet is kept in a buffer to ensure consistent behavior. As a result of this algorithm, random ports will appear to the scanner as being open. Digging deeper in order to identify services running on these fake open ports would require more resources from the attacker [16]. Secondly, the controller determines the type of connection (i.e., via srcIP or dstIP) and installs necessary flows in all OF switches in the path. These flows will change the srcIP and dstIP of each packet (assuming srcIP changed to be vsrcIP and dstIP changed to be vdstIP) so that the packet will be different from what they actually are. But meanwhile, these flows will also make sure that the packet can be sent to the destination host by changing the vsrcIP and vdstIP to srcIP and dstIP in the end. Each connection must be associated with a unique flow, because the rIP-vIP translation changes for each connection. This property guarantees the end-to-end reachability of hosts, because the rIP-vIP translation for a specific connection remains unchanged regardless of subsequent mutations [15].

The process is presented as Algorithm 1. Here we use a pseudo-code to clarify the process. Firstly, if we find that the packet-in message comes from the source switch or destination switch of the packet, we will install flow tables of host mutation. Then, the connection will be judged to be obfuscated or not. For expected connections, the packet will be forwarded directly. But for unexpected connections, the packet will be obfuscate or forwarded to a decoy server if the altitude is bigger than the threshold configured by administrator.

## 5. Implementation and Evaluation

*5.1. System Implementation.* The structure of our system is shown in Figure 5. The routing was managed entirely by the Floodlight controller and monitored by Bro. We implemented three modules. The first one we implemented is the Chaos Tower module, the purpose of which is to build the Chaos Tower and get unexpected flows. Then, we implemented the obfuscation module in Floodlight, which obfuscates the unexpected flows and abnormal traffic judged by IDS. Finally, we implemented the CHAOS management module which allows administrators to further configure their networks.

We provide an implementation of obfuscation with Bro's warning message. In the beginning, we push flow tables into switches so that all flows are allowed. Then, we use Bro to monitor the network. When suspicious flows are detected,

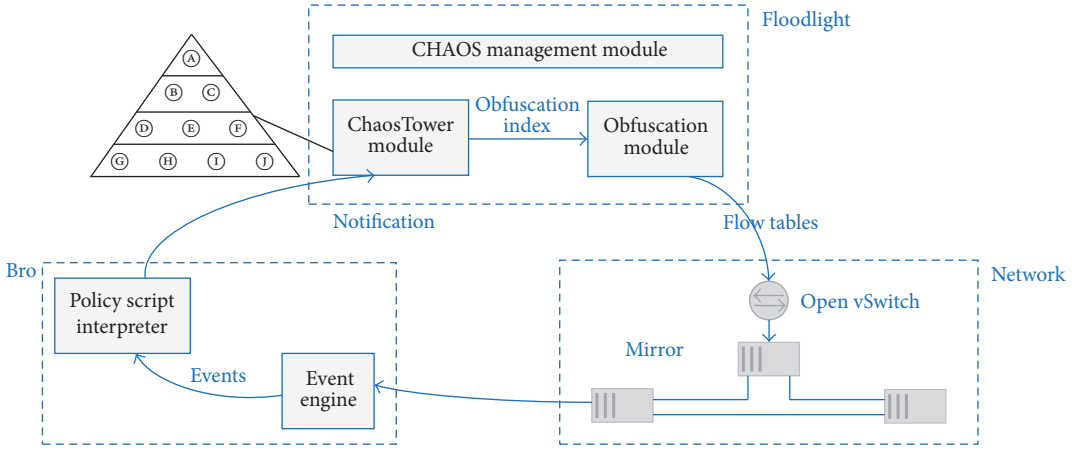


Figure 5: System implementation.

the tower will determine the corresponding obfuscation index and transfer it to obfuscation module. After that, corresponding flow tables will be updated to make sure that the obfuscation works in the network.

*5.2. Scanning and Foot-Printing Test.* Foot-printing and scanning are techniques for gathering information about computer systems in networks. These techniques are implemented by various security auditing tools as the first step when launching an attack. Nmap [17] and the scanner modules in Metasploit [18] contain many payloads to gather sensitive information from target machines, whereas Nessus [19] and WVS (Web Vulnerability Scanner) focus on vulnerability detection and exploitation.

In our test, we used Nmap to evaluate the information obfuscation ability of CHAOS. Nmap uses raw IP packets in novel ways to determine which hosts are available on the network, which services (application name and version) those hosts are offering and which operating systems (and OS versions) they are running, which type of packet filters/firewalls are in use, and many other characteristics [17]. Our test involved configuring some vulnerable hosts in the network, after which we used Nessus to detect vulnerabilities to test whether CHAOS would be able to confuse and deceit Nessus.

We tested the performance of our system by launching a series of attacks under different circumstances. We consider three situations against Nmap. In the first, the network was unprotected; in the second, we implement a fully obfuscated system [16]; and in the third, our CHAOS system was implemented. When simulating the attack, we used Nmap to scan the entire network several times. Based on its response and the reality of its given circumstances, we concluded the result (Figures 6 and 7). Besides this, we used a ping command to test the effect of our system on normal traffic (Figure 8).

*5.3. Results.* We carried out our experiments in CloudLab [20] and deployed the network shown in Figure 2.

First, we used Nmap to determine whether our CHAOS system was able to deceit the security tool. There are two situations involved in this experiment. We selected the hosts of Group 4 and Group 3 in Figure 2; thus, the obfuscation index is 0.5, so obfuscation based on decoy servers will work then.

We define information disclosure percentage (IDP) as our index and calculate it by the following formulas. ID is the amount of information that the adversary fetches from the victim. NONE represents the unprotected network. FON represents the fully obfuscated network. CHAOS represents the network protected by CHAOS.

$$\begin{aligned} \mathrm{IDP}_{\mathrm{CHAOS}} &= \frac{\mathrm{ID}_{\mathrm{CHAOS}}}{\mathrm{ID}_{\mathrm{NONE}}}, \\ \mathrm{IDP}_{\mathrm{FON}} &= \frac{\mathrm{ID}_{\mathrm{FON}}}{\mathrm{ID}_{\mathrm{NONE}}}. \end{aligned} \tag{2}$$

Figure 6 shows the percentage of information disclosure of an unprotected network and a network (Level 2) protected by CHAOS as a function of the number of times the network was scanned by Nmap. The figure shows that, for the network protected by CHAOS, the percentage of information disclosure is decreased effectively.

Secondly, we studied the correlation between the degree of threat of the adversary and the information disclosure he would experience. For comparison, we implemented another MTD system, fully obfuscated network, which obfuscates all the packets in the network. Figure 7 shows the information disclosure in an unprotected network, a network protected by CHAOS, and fully obfuscated network [15], all of which face different degrees of threats. The fully obfuscated network obfuscates all the packets by some static policies. Thus, it is able to decrease information disclosure when the threat reaches a certain degree, but does not decrease information disclosure further when the degree of threat is elevated beyond that certain degree, because of its static solution. However, the network protected by CHAOS decreases information disclosure when the degree of threat is elevated. Only

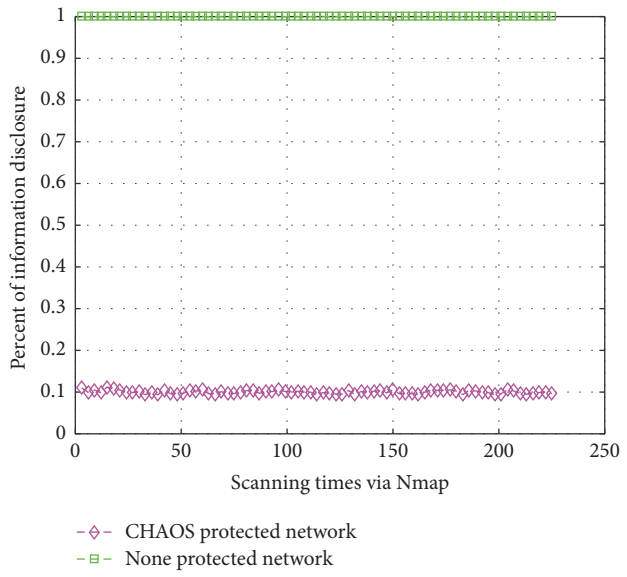


Figure 6: Information disclosure with scanning time.

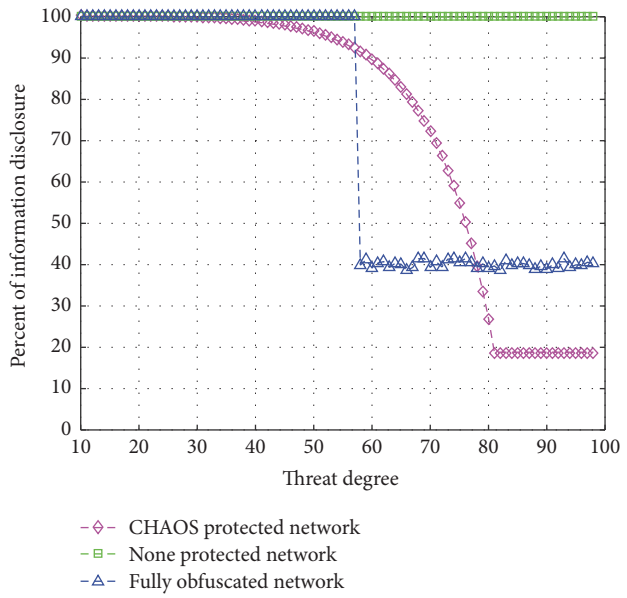


Figure 7: Information disclosure with respect to threat degree.

a few information disclosures exist when the threat reached a very high degree.

After that, we compared the performance cost of the three networks. As above, we compare the network protected by CHAOS with the unprotected and fully obfuscated network. We use the example shown above to test the performance of these systems and to measure the average delay time of the connections under each system. Figure 8 shows the delay time of the unprotected network, the network protected by CHAOS, and fully obfuscated network with changing package counts. We conclude that both the networks protected by CHAOS and fully obfuscated network increase the delay time to some extent, although the network protected

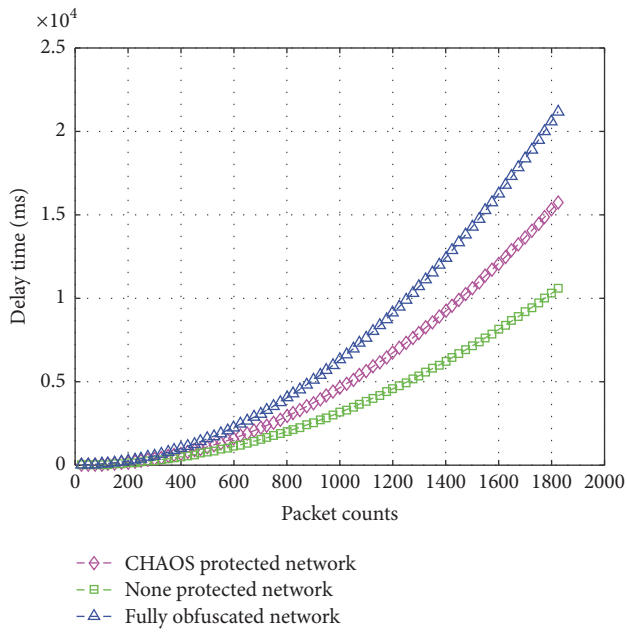


Figure 8: Delay time with respect to packet count.

by CHAOS has a reduced delay compared to that fully obfuscated network. Thus, our system enables the network to perform faster. We discovered that the transforming speed of our system is faster than that of random obfuscation system especially when the network is crowded.

The result above can be understood in terms of the following factors.

First, we use Bro to monitor the network and transfer those suspicious flows. The important point is that Bro runs stand-alone so it makes quite few effects to the speed of the network.

Then, the Chaos Tower is also a factor that reduces the delay time. We assume that the Chaos Tower is to be built as a binary tree in the network and the number of layers is $L$; hence,

$$N = 2^L - 1. \tag{3}$$

We consider a situation in which each workgroup sends a request to the remaining groups, which means that the sum of the connections the unprotected situation and the MTD solution would have to process would be

$$\begin{aligned} C_{\text{NONE}} &= 0, \\ C_{\text{MTD}} &= N * (N - 1). \end{aligned} \tag{4}$$

However, we only need to obfuscate the connections from the lower layers toward the higher layers in our CHAOS system, the number of which is

$$C_{\text{CHAOS}} = \sum_{i=1}^{L-1} \left(2^i * \left(2^i - 1\right)\right). \tag{5}$$

In the end, we launched several real attacks to testify robustness of our system. We employ some vulnerable hosts in the network. In the experiment, MS 08-067

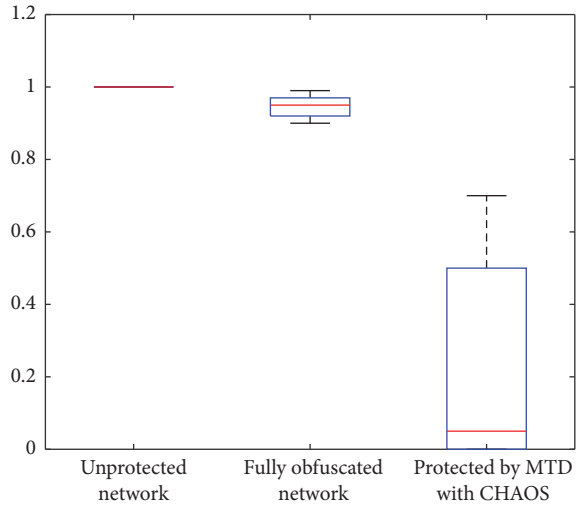


Figure 9: Attack testing.

is the vulnerability that we test. The hosts can be easily attacked by any pen-testing tools which contain payload of MS 08-067. Actually, in Chaos Tower, we employ a vulnerable host in each layer. Then we use one of them to play the role of attacker in turn. Figure 9 shows the results of the unprotected network, the network protected by CHAOS, and the fully obfuscated network. We conclude that, in the network protected by CHAOS, only a few attacks directed to hosts belonging to adjacent layers succeeded. However, in the fully obfuscated network, most attacks succeeded in the end. The worst is that almost all attacks succeeded in the unprotected network. Thus, our system can decrease the success rate of such kind of attacks significantly.

## 6. Related Work

Several researchers have reported work on MTD. Kewley et al. [21] performed the initial research in the area of dynamic network defense and proved that dynamic network reconfiguration, such as randomly changing the IP address and port numbers, would effectively inhibit an adversary's ability to gather intelligence and thus degrade the ability to successfully launch an attack. Al-Shaer proposed MUTE, a moving target defense architecture [5], which implements the moving target through random address hopping and random finger printing. Furthermore, they presented BDD, a model for creating a valid mutation of network configuration. Zhuang et al. [4] investigated the application of moving target defenses to network security and presented a high-level architecture of the MTD system. Their simulation results show the potential for MTD to be effective in preventing attacks against computer networks. Furthermore, they proposed a formal theory to describe the MTD system and its basic properties and formalized the MTD entropy hypothesis, which states that the greater the entropy of the system configuration, the more effective the MTD system [22, 23]. Stallings proposed the use of SDN in the implementation of MTD mitigations. Al-Shaer et al. [15] proposed OpenFlow Random Host Mutation (OF-RHM), which uses OpenFlow to develop an MTD architecture that transparently mutates host IP addresses with high unpredictability, while maintaining configuration integrity and minimizing operational overhead.

However, current network-based MTD obfuscates networks indiscriminately that makes some networks services unavailable, for example, some key services like web and DNS, because some information of these services has to be opened to the outside and remain real. If MTD obfuscates these services fully, it will return users with virtual IPs and ports, making these services unable to use. Moreover, obfuscation will affect the performance of networks. To obfuscate hosts indiscriminately will severely reduce the performance of networks undoubtedly. In contrast to the above work, CHAOS discriminately obfuscates hosts with different security levels in networks.

Zhang et al. [24] proposed to construct an incentive compatible moving target defense by periodically migrating virtual machines (VMs), thereby making it much harder for adversaries to locate the target VMs. Gillani et al. [25] proposed to defend against DDoS attacks by migrating virtual networks (VNs) to dynamically reallocate network resources. Different from their work, CHOAS leverages SDN features to obfuscate network information instead of migrating target objects.

Previous research involving memory address space randomization [26–28], instruction set randomization [29], and software diversification [30, 31] also used the idea of a moving target to increase the attack difficulty and cost by enlarging the exploration surface or moving the attack surface. The objective of our work is to enhance network security; hence, the aspects mentioned here are not discussed in detail.

## 7. Conclusion

MTD is able to create a type of changing network so as to increase the difficulty and cost for an adversary aiming to launch a network attack. In this paper, we proposes an SDN-based MTD system named CHAOS which discriminately obfuscates hosts with different security levels in networks so as to keep some key services available and low performance cost. CHAOS incorporates the Chaos Tower Structure to represent a hierarchy of all the hosts on the network and leverages SDN features to obfuscate the attack surface to enhance the unpredictability of the networking environment. CHAOS offers rapid obfuscation of unexpected network traffic but does not interfere with normal traffic. The evaluation shows that a network protected by CHAOS can effectively lower the percentage of information that is disclosed.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

This work was supported by the National Natural Science Foundation of China (nos. 61272452, 61332019, and 61402342), the National High-Tech Research and Development Program

of China ("863" Program) (no. 2015AA016002), and the National Basic Research Program of China ("973" Program) (no. 2014CB340601).

## References


[1] Open Networking Foundation, "OpenFlow1.1.0 specification," 2011, http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf.

[2] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Adversary-aware IP address randomization for proactive agility against sophisticated attackers," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '15)*, pp. 738–746, IEEE, April 2015.

[3] Y. Wang, J. Bi, and K. Zhang, "A tool for tracing network data plane via SDN/OpenFlow," *Science China Information Sciences*, vol. 60, no. 2, Article ID 022304, 2017.

[4] R. Zhuang, S. Zhang, A. Bardas, S. A. DeLoach, X. Ou, and A. Singhal, "Investigating the application of moving target defenses to network security," in *Proceedings of the 2013 6th International Symposium on Resilient Control Systems, ISRCS 2013*, pp. 162–169, San Francisco, Calif, USA, August 2013.

[5] E. Al-Shaer, "Toward network configuration randomization for moving target defense," in *Moving Target Defense*, vol. 54 of *Advances in Information Security*, pp. 153–159, Springer, New York, NY, USA, 2011.

[6] J. Sun and K. Sun, "DESIR: Decoy-enhanced seamless IP randomization," in *Proceedings of the 35th Annual IEEE International Conference on Computer Communications, IEEE INFOCOM 2016*, April 2016.

[7] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, and G. Gu, "Towards SDN-Defined Programmable BYOD (Bring Your Own Device) Security," in *Proceedings of the Network and Distributed System Security Symposium*, San Diego, Calif, USA, February 2016.

[8] T. Yu, S. K. Fayaz, M. Collins, V. Sekar, and S. Seshan, "PSI: Precise Security Instrumentation for Enterprise Networks," in *Proceedings of the 2017 Network and Distributed System Security Symposium (NDSS '17)*, San Diego, Calif, USA, February 2017.

[9] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[10] J. Sonchack, A. J. Aviv, E. Keller, and J. M. Smith, "Enabling Practical Software-defined Networking Security Applications with OFX," in *Proceedings of the Network and Distributed System Security Symposium (NDSS '16)*, San Diego, Calif, USA, February 2016.

[11] Stanford University, "Clean slate program," http://cleanslate.stanford.edu/.

[12] Open Networking Foundation, "OpenFlow switch specification," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[13] Flowgrammable Team, "Packet in messages," 2014, http://flowgrammable.org/sdn/openow/message-layer/packetin.

[14] P. Mell, K. Scarfone, and S. Romanosky, "A Complete Guide to the Common Vulnerability Scoring System Version 2.0," 2007, https://www.nist.gov/publications/complete-guide-common-vulnerability-scoring-system-version-20.

[15] E. Al-Shaer, Q. Duan, and J. H. Jafarian, "Random host mutation for moving target defense," in *Security and Privacy in Communication Networks*, A. D. Keromytis and R. Di Pietro, Eds., vol. 106 of *Lecture Notes of the Institute for Computer Sciences*, pp. 310–327, Springer, Berlin, Germany, 2013.

[16] P. Kampanakis, H. Perros, and T. Beyene, "SDN-based solutions for Moving Target Defense network protection," in *Proceedings of the 15th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM '14)*, pp. 1–6, Sydney, Australia, June 2014.

[17] G. Lyon, Network mapper, 2017, https://nmap.org/.

[18] Rapid7 LLC, Metasploit, 2009, https://www.offensive-security.com/metasploit-unleashed/vulnerabilityscanning.

[19] Tenable, Nessus, 2017, http://www.tenable.com.

[20] The CloudLab Team, CloudLab, 2014, http://www.cloudlab.us.project.

[21] D. Kewley, R. Fink, J. Lowry, and M. Dean, "Dynamic approaches to thwart adversary intelligence gathering," in *Proceedings of the DARPA Information Survivability Conference and Exposition II, DISCEX 2001*, pp. 176–185, Anaheim, Calif, USA, June 2001.

[22] R. Zhuang, S. A. DeLoach, and X. Ou, "A model for analyzing the effect of moving target defenses on enterprise networks," in *Proceedings of the 9th Annual Cyber and Information Security Research Conference (CISRC '14)*, pp. 73–76, April 2014.

[23] R. Zhuang, S. A. DeLoach, and X. Ou, "Towards a theory of moving target defense," in *Proceedings of the First ACM Workshop on Moving Target Defense (MTD '14)*, pp. 31–40, ACM, November 2014.

[24] Y. Zhang, M. Li, K. Bai, M. Yu, and W. Zang, "Incentive compatible moving target defense against VM-colocation attacks in clouds," in *Information Security and Privacy Research*, pp. 388–399, Springer, Berlin, Germany, 2012.

[25] F. Gillani, E. Al-Shaer, S. Lo, Q. Duan, M. Ammar, and E. Zegura, "Agile virtualized infrastructure to proactively defend against cyber attacks," in *Proceedings of the 34th IEEE Annual Conference on Computer Communications and Networks, IEEE INFOCOM 2015*, pp. 729–737, May 2015.

[26] F. P. Miller, A. F. Vandome, and J. McBrewster, *Address space layout randomization*, Alphascript Publishing, 2010.

[27] K. Chongkyung, J. Jinsuk, C. Bookholt, X. Jun, and N. Peng, "Address Space Layout Permutation (ASLP): Towards fine-grained randomization of commodity software," in *Proceedings of the 22nd Annual Computer Security Applications Conference, ACSAC 2006*, pp. 339–348, December 2006.

[28] H. Shacham, M. Page, B. Pfaff, E. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization," in *Proceedings of the 11th ACM conference on Computer and communications security (CCS '04)*, p. 298, Washington, DC, USA, October 2004.

[29] S. W. Boyd, G. S. Kc, M. E. Locasto, A. D. Keromytis, and V. Prevelakis, "On the general applicability of instruction-set randomization," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 3, pp. 255–270, 2010.

[30] Y. Huang and A. K. Ghosh, "Introducing diversity and uncertainty to create moving attack surfaces for web services," in *Moving Target Defense*, vol. 54 of *Advances in Information Security*, pp. 131–151, Springer, New York, NY, USA, 2011.

[31] M. Christodorescu, M. Fredrikson, S. Jha, and J. Giffin, "End-to-end software diversification of internet services," in *Moving Target Defense*, vol. 54 of *Advances in Information Security*, pp. 117–130, Springer, New York, NY, USA, 2011.