# The SDN Shuffle: Creating a Moving-Target Defense using Host-based Software-Defined Networking

Douglas C. MacFarland and Craig A. Shue
Dept. of Computer Science, Worcester Polytechnic Institute
Worcester, MA, USA
dcmacfarland@cs.wpi.edu, cshue@cs.wpi.edu

## ABSTRACT

Moving target systems can help defenders limit the utility of reconnaissance for adversaries, hindering the effectiveness of attacks. While moving target systems are a topic of robust research, we find that prior work in network-based moving target defenses has limitations in either scalability or the ability to protect public servers accessible to unmodified clients. In this work, we present a new moving target defense using software-defined networking (SDN) that can service unmodified clients while avoiding scalability limitations. We then evaluate this approach according to seven moving-target properties and evaluate its performance. We find that the approach achieves its security goals while introducing low overheads.

## 1. INTRODUCTION

With moving target defenses, an organization can limit the ability of adversaries to perform reconnaissance on the assets within the organization's network. This information disadvantage can force adversaries to attack blindly, which may be intractable with a large search space or easily noticed with proper defensive instrumentation. The moving target concept has created a rich body of research [14], with work spanning from the popular ASLR strategy [17] to network-based defenses.

With network-based defenses, a few prominent classes of protections have emerged: techniques which modify both communicating end-hosts to coordinate their movements, such as the MT6D approach [6], and techniques that modify network infrastructure, including the DNS capabilities [18] and OpenFlow Mutation [9] approaches. Each of these approaches come with inherent tradeoffs. For example, when both end-hosts must be modified, organizations cannot protect public-facing server infrastructure without requiring clients to install software. In network-centric approaches, it can be difficult to distinguish between legitimate and malicious clients. Further, the infrastructure necessarily becomes stateful, introducing possible denial-of-service risks.



*MTD'15 October 12 2015, Denver, CO, USA*



ISBN 978-1-4503-3823-3/15/10...$15.00

DOI: http://dx.doi.org/10.1145/2808475.2808485

To focus our work, we ask two research questions: 1) *How can we provide both a scalable and flexible moving-target system?* and 2) *What features can assist a moving target system with distinguishing trustworthy and untrustworthy clients?*

To answer these questions, we embrace the "dumb network, smart hosts" paradigm. We take the functionality from the DNS capabilities and OpenFlow mutation approaches and integrate it into server infrastructure. We then accept two types of clients: unmodified clients and those that can employ additional measures to facilitate the approach. In doing so, we allow a deploying organization to protect publicly available servers while providing heightened protections for internal traffic and for remote trusted clients.

Our contributions are the following:

- **Host-Based Moving Target Defense:** We describe and implement our approach to hinder adversarial reconnaissance with moving targets at the datalink, network, and transport layers. The approach is scalable and effective with unmodified clients, but provides additional security features when both hosts deploy it.
- **A Security and Performance Evaluation of the Defense:** We evaluate our approach according to seven security properties and determine the performance of our proof-of-concept implementation. Our results show the approach provides each of the security properties and yields performance overheads that are acceptable for practical deployment.

## 2. BACKGROUND AND RELATED WORK

We now provide background on the software-defined networking (SDN) paradigm and the OpenFlow protocol. We then describe other work to hinder attacker reconnaissance and how our work differs.

### 2.1 SDN and OpenFlow

The SDN paradigm creates a separation between data-plane processing, which forwards packets, from control-plane processing, which determines how to populate forwarding tables. The OpenFlow protocol [12] acts as an API between network switches and a logically centralized decision maker, called the OpenFlow controller. In this model, network switches cache data-plane flow rules. When a switch receives a packet and does not know how to forward it according to its cached rules, the switch sends an "elevation" request containing the original packet and a request for guidance to the controller. The controller examines the packet and sends a set of rules that the switch should add to the data plane cache for use in forwarding packets.

The OpenFlow Mutation [9] approach uses fine-grain flow rules, which match only a single connection between a source and destination host, to perform network address translations on each packet's source and destination addresses. The approach can obscure the real IP addresses of the hosts on the network, preventing network mapping. Unfortunately, prior work has shown that fine-grain flows in OpenFlow's data plane controls do not scale to large networks [5]. Studies on OpenFlow-compatible switches show that some switches are only able to handle 150 new flows per second while others handle 750 flows per second [20]. Other work has found that some commonly-used switches have high-speed TCAM memory limits of 2000-4000 entries and that, in some cases, memory swapping between TCAM and slower-speed memory can reduce the switch's new flow capacity to only 12 flows per second [11]. These limitations can cause performance bottlenecks and denial-of-service conditions even with benign traffic; further, adversaries can induce switch thrashing to create network outages [16].

## 2.2 Network-Based Anti-Reconnaissance

Reconnaissance is the first phase in any network intrusion [4]. By denying an adversary the ability to gather information on potential targets, defenders can hinder the adversary's efforts to spread to other hosts in the network.

One technique is network address space randomization (NASR) [2]. Similar to the address space layout randomization (ASLR) technique used to protect against buffer overflow attacks [17], NASR works by changing the address of a machine randomly. This limits or removes an adversary's ability to build up knowledge of IP address over time, and has been shown to negatively impact scanning malware [1]. NASR can be implemented in several different ways, depending on the type of desired randomization.

Prior work focused on defeating hit-list scanning malware utilized DHCP to change the IP address of the host over time [3]. However, this change can disrupt existing connections. The authors suggested placing an intermediate NAT-like box in the network to transparently transition to the new address over time. The NAT-like device provided similar address translating behavior, with additional logic to preserve old addresses while they are still in use by previously established connections.

Our own work utilized DNS capabilities and an intermediary NAT device [18]. Rather than changing out the DHCP lease to achieve randomization, we randomly rotated the IP address contained in the DNS reply and notified the NAT device. The NAT device then allows new connections with that IP to reach the requested host by mapping it to the host's fixed internal address. This approach reuses proven technology and avoids the need for special tools.

Our approach is distinct from each of these in that it does not require mapping state in network switches or NAT devices, eliminating potential scalability concerns. This is particularly important for a security approach where adversaries may intentionally try to induce a denial of service condition by targeting resource-constrained infrastructure.

## 2.3 Host-Based Anti-Reconnaissance

Dunlop *et al.* [6] created the MT6D ("Moving Target IPv6 Defense") system in 2011. In it, the client and target share a symmetric key out-of-band and use these keys to determine the IPv6 addresses the hosts will use. To compose their IPv6 addresses, the hosts construct a hash using the shared key, a value derived from the host's MAC address, and a timestamp. The MT6D approach is only effective when both the client and the server can be modified, which prevents its use when protecting public infrastructure for legacy clients.

At a lower level, the wireless frequency hopping strategy used to avoid jammers [13] is an example of a moving target defense. This approach attempts to evade an adversary that cannot feasibly block all frequencies simultaneously and uses channel hopping to avoid the active blocking attempts. While this strategy uses a different entropy space and identifiers (namely a chosen frequency within the range of available 802.11 frequencies), its considerations are quite similar to the MT6D approach, which also requires agreement between hosts on hopping patterns.

The port knocking approach [10] is designed to allow a defender to only authorize connections to a server after a special sequence of packets, to specified ports, are received from a client. The approach essentially uses transport layer ports as moving target space for establishing a connection. This approach requires both the client and the server to previously share a key (in this case, used to determine the ports to knock) to establish a connection. Accordingly, the approach can be analyzed similarly to the MT6D approach.

In cases where both the client and server deploy our approach, we achieve similar outcomes to the MT6D and the others. However, our approach allows a legacy mode that supports unmodified clients while still providing a moving target defense. This allows the approach to be deployed on public-facing systems.

# 3. THREAT MODEL

We include the SDN controller, the DNS server, and the operating system and hardware of the protected infrastructure in our trusted computing base (TCB). In cases where the client deploys the approach, such as communication within a LAN, we also include the client's hardware and operating system in the TCB. This prohibits any administrative-level compromises, which reflects the best practice of "least user privilege" [15] under which adversaries will only be able to run within a compromised account's privileges. This is a common assumption among host-based defense systems, such as anti-virus, firewall, and host-based intrusion detection software. Finally, we exclude active man-in-the-middle attackers which could hijack an authorized connection.

This threat model does allow normal user-level accounts on client machines and protected infrastructure to be compromised. It further allows passive network monitoring outside of a host, such as hosts that promiscuously tap network connections. However, we note that adversaries that have compromised an account on a modified client or server will not have access capture network packets on that machine, since that action requires administrative privileges.

# 4. APPROACH

Once an adversary establishes a foothold within an organization, the adversary's goal is to perform network reconnaissance [4]. Adversaries can use the knowledge gained from reconnaissance to pivot from their foothold and compromise more hosts. Adversaries can identify hosts within a network using IP addresses and MAC addresses. Our approach op-

erates by utilizing synthetic addressing information in place of the real addressing information at the data link and network layers. This synthetic information can be considered to be chosen at random within certain validity constraints. By creating a MTD that protects both these values, we can prevent the adversary from progressing through the cyber security kill chain [8] and contain breaches.

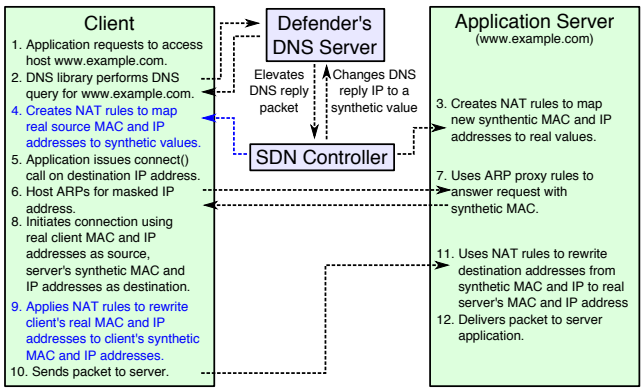


**Figure 1: Anti-reconnaissance process. Steps 4 and 9, in blue, are optional and are used only when the client has been modified to employ the approach.**

To avoid mapping system scalability concerns in network-centric MTD systems, we use a host-based approach. We essentially create an OpenFlow agent on the responding host, typically a server, that will perform packet elevation and apply rules like a normal OpenFlow agent. This provides the benefits of network-based systems while enabling scalability, since the hosts must already manage per-flow state to manage the connection, as in TCP connections. Clients may optionally deploy the approach as well, which helps distinguish between trustworthy and untrustworthy clients. Even when clients do not employ the approach, we can still effectively achieve the goals of a MTD.

In Figure 1, we visually depict the steps in our approach. The process begins when the client application requests a DNS resolution of a host name to an IP address. A DNS library on the client requests the resolution from its configured DNS resolver. Eventually, the resolution request will reach the defending organization's authoritative DNS server. The DNS server application will answer the response, as usual, and will set an extremely low time-to-live (TTL) value on the response (e.g., five seconds or less). However, before the server's operating system transmits the packet back to the requestor, the DNS server diverts the response to the SDN controller for consideration.

The SDN controller then generates a synthetic IP address and MAC address for the server associated with the requested host name to use. It orders that application server to install NAT rules that 1) translate the synthetic IP address into the server's real IP address and 2) translate the synthetic MAC address into the real MAC address for the host. If the client deploys the approach as well, the SDN controller will likewise generate synthetic IP and MAC addresses for the client and send similar NAT rules to the client that will take effect when the client attempts to communicate to the specified synthetic server addresses. The SDN controller then rewrites the IP address in the DNS reply to reflect the server's synthetic IP address. The SDN controller returns the DNS reply packet to the DNS server, which then returns it to the DNS resolver that issued the query.

The client delivers the DNS response to the client application. That application then attempts to establish a connection, which creates a packet that is transmitted to the client's operating system. The client's operating system then sends the packet to the default gateway, if the server is outside the subnet, or otherwise uses the address resolution protocol (ARP) to determine the destination machine's MAC address. The server responds to the ARP query with its synthetic MAC address due to the rules it previously installed when ordered by the SDN controller. Upon receiving the ARP response, the client prepares to transmit the packet using its own real MAC address and IP address for the source and using the synthetic MAC and IP addresses it learned as the destination fields. If the client deploys the approach and received an order from the SDN to create NAT rules, the client applies the NAT rules. Those rules rewrite the source MAC and IP addresses to the synthetic variants that the SDN controller created for the client. The client then sends the packet.

Once the server receives the client's packet, it uses the NAT rules from the SDN controller's previous order to translate the synthetic MAC and IP addresses into their real equivalents for the host. The packet is then delivered to the server's application. If the server application chooses to reply, it will reverse the source and destination addresses for its reply packet. The server will then apply the inverse of the NAT rules, rewriting the source addresses from the server's real values back to their synthetic equivalents. It will then send the packet using the client's supplied address. If the client is modified, it will employ the inverse of its NAT rules to translate the destination addresses from the synthetic values back to the client's real values and supply them to the client application. Subsequent messages in the connection will follow the same NAT'ing pattern for communication.

This approach achieves a powerful outcome: each client receives synthetic addresses for the server, which the SDN controller can rotate for each new DNS resolution. This allows the SDN controller to provide a different, moving IP address for each client and the short TTL for the client ensures that the client will re-issue DNS requests for new connections, allowing the server to again change the addresses.

In the case where the client deploys the approach, the client also uses synthetic addresses, preventing an adversary on the server from being able to learn the identities of deploying clients. Further, when both systems deploy, the SDN controller can update both the client and server NAT rules to transition the connection to a new IP address and MAC address combination, preventing an adversary from being able to track flows across movements, providing both anonymity and unlinkability[1]. These properties are similar to the MT6D approach, but avoid the need for IP tunneling.

## 5. IMPLEMENTATION

Our reference implementation uses Python scripts on the Ubuntu Linux operating system. While the details of the approach will vary across operating systems, the concepts are consistent and similar functionality may be available. To

[1]Similar transport layer translations can prevent adversaries from distinguishing connections by inspecting ports.

enable communication between agents and the controller, we used asynchronous messaging with the Twisted framework [19]. These components were not optimized for performance and thus are conservative estimates of what would be possible in a production implementation.

Our DNS server is a standard BIND9 installation on an Ubuntu server with a sample DNS zone to be queried. To enable diversion to the SDN controller, we created a program that uses the Linux kernel's `netfilter_queue` library to tell the kernel that it should intercept any DNS response packets originating locally. That program then transmits a copy of the DNS response packet to the SDN controller for review via a separate connection to the controller. When the agent receives the SDN controller's response, it instructs the kernel to re-queue the packet, but specifies the modified packet rather than the original packet, queuing the altered version.

Our SDN controller is a Python script that receives these messages from the DNS and parses them. It then generates random IP and MAC addresses for the client and server and sends orders to the those machines to install the appropriate rules for handling the packets. It then replies to the DNS server with alterations to the responses.

The client and server run agents as root that await commands from the SDN controller. To implement the NAT rules supplied by the network controller, the agents use `iptables` for network layer translations and `ebtables` for datalink layer translations.

This approach provides a basic moving target defense that forces clients to engage with the DNS server to actually reach the legitimate server. The approach provides malicious clients with no persistent knowledge about the server and prevents a network observer from being able to understand the activity on the network. However, a malicious application running at the user-level on the client or server could learn the synthetic addresses of the communicating counterpart and attempt to establish a concurrent connection in order to launch an attack. In the Linux operating system, unprivileged users can use the `netstat` tool to discover established connections.

# 6. EVALUATION

To determine the viability of our approach, we evaluate the security properties of the system and performance characteristics of the reference implementation.

## 6.1 Security Evaluation

We evaluate the moving target defense using the properties identified by Green *et al.* [7]:

**Unpredictability:** The defense must move its protected assets so that a client cannot guess the new destination of any given asset unless the client has an active authorization to that asset. We achieve this using a cryptographically sound pseudorandom number generator for the lower-order bits in the IP prefix.

**Vastness:** The destination space of the defense must be sufficiently large so that it is intractable for a client to gain access to an asset by exhaustively enumerating all possible destinations. Since the protected hosts do not listen for connections on its real IP address, and synthetic IP addresses are created on demand and only allow a single remote IP address to access them at a time, an attacker will not gain access by exhausting the IP space.

**Periodicity:** The assets must be moved with enough regularity that any reconnaissance collected by untrusted clients expires quickly. We provide this property by using a short DNS TTL and changing the addresses for each DNS lookup.

**Uniqueness:** The defense must individually authorize a client and prevent that authorization from being shared with any other client. We simply delete the synthetic record (or specific flows associated with the record) to remove access.

**Revocability:** The defense must be able to terminate or expire a prior authorization without causing collateral damage or disruption to other clients. Our removal of flow-specific and client-specific records meets the revocability property.

**Availability:** The defense must not introduce any new availability constraints or denial-of-service vulnerabilities that would prevent an authorized client from reaching a protected asset. Since our approach uses state already present on end-hosts, we introduce no new availability constraints and thus meet the availability property.

**Distinguishability:** The defense must distinguish trustworthy clients from untrustworthy clients in order to only authorize the former. We can provide distinguishability by essentially encoding passwords (e.g., `[password].example.com`) inside the host names that the clients must provide in their DNS queries. By sharing the password only with legitimate users, organizations can ensure distinguishability.

From this evaluation, we have determined the approach meets the required properties of a moving-target defense.

## 6.2 Performance Evaluation

The approach reuses existing infrastructure, such as the DNS and NAT processes, that are already commonly used in the Internet and yield acceptable performance. Accordingly, we only need to evaluate the performance of two new elements: 1) the time required to divert a packet from an end-host to the SDN controller, which captures the overhead of the communication between the DNS server and the SDN controller, and 2) the time required for a host to apply NAT rules in an order from the SDN controller. Accordingly, we evaluate each of these overheads empirically.

Our experiments run on a VM server running a KVM hypervisor with 16 cores operating at 2.8 GHz and 64 GBytes of RAM. It runs two VMs: a VM representing the SDN controller and a VM representing the controller's counterpart, which is the DNS server in the first experiment and is the application server in the second experiment. The SDN controller has two cores and 2 GBytes of RAM and the application server has a single core and 512 MBytes of RAM. Both machines run the Ubuntu 14.04 Server OS. We use NTP to synchronize the VM clocks for timing analysis. The hosts ignore ICMP redirect messages to ensure proper packet escalation to the SDN controller. We conducted $1,000$ trials of each experiment.

In the first experiment, we measure the time to intercept a packet, divert it to a controller, make a decision at the controller, and return the packet to the DNS server. Across our $1,000$ trials, the median latency of this process was 12.74 milliseconds with roughly a 1 millisecond standard deviation. This short delay is unlikely to be noticed by users.

Our second experiment measures the time to apply a set of NAT rules using the `iptables` command. The median

latency was 3.98 milliseconds, with a 0.49 millisecond standard deviation. As a result, the SDN controller could order the server (and, in parallel, the client, if it implements the approach) to apply NAT rules while only introducing a small delay. The rules for the NAT entries, using `ebtables`, would likely take the same amount of time. With kernel libraries, such as `libiptc`, the hosts could directly manipulate the kernel's NAT entries and avoid the substantial overhead of forking a new process to execute the `iptables` command.

From these experiments, we note that all the manipulations required would take around 20 milliseconds to perform and these overheads would only occur during the DNS lookup process. After the connection is established, the approach only introduces the minimal time to perform NAT translations on packets. As a result, we find that the performance of the approach would be barely perceptible to users.

# 7. CONCLUSION

We explored techniques to provide a scalable moving target system that supports unmodified clients while distinguishing trustworthy clients from untrustworthy ones. We proposed a new moving target defense using SDN techniques that provides key security properties while yielding acceptable performance. Our approach allows defenders to distinguish between trustworthy and untrustworthy clients using pre-shared keys, using cryptographic MACs, or simply embedding passwords in to standard hostnames to provide access control for legacy clients.

## Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 1422180. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

# 8. REFERENCES

[1] M. Abu Rajab, F. Monrose, and A. Terzis. On the impact of dynamic addressing on malware propagation. In *ACM Workshop on Recurring Malcode*, pages 51–56. ACM, 2006.

[2] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis. Defending against hitlist worms using network address space randomization. *Computer Networks*, 51(12):3471–3490, 2007.

[3] S. Antonatos and K. G. Anagnostakis. Tao: Protecting against hitlist worms using transparent address obfuscation. In *Communications and Multimedia Security*, pages 12–21. Springer, 2006.

[4] S. Barnum. Standardizing cyber threat intelligence information with the structured threat information expression (stix$^{TM}$). *MITRE Corporation*, page 11, 2012.

[5] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 254–265, New York, NY, USA, 2011. ACM.

[6] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront. MT6D: A moving target IPv6 defense. In *Military Communications Conference*, pages 1321–1326. IEEE, 2011.

[7] M. Green, D. C. MacFarland, D. R. Smestad, and C. A. Shue. Characterizing network-based moving target defenses. In *ACM CCS Workshop on Moving Target Defense (MTD)*, October 2015.

[8] E. M. Hutchins, M. J. Cloppert, and R. M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1:80, 2011.

[9] J. H. Jafarian, E. Al-Shaer, and Q. Duan. OpenFlow random host mutation: Transparent moving target defense using software defined networking. In *Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 127–132, New York, NY, USA, 2012. ACM.

[10] M. Krzywinski. Port knocking from the inside out. *SysAdmin Magazine*, 12(6):12–17, 2003.

[11] A. Lazaris, D. Tahara, X. Huang, E. Li, A. Voellmy, Y. R. Yang, and M. Yu. Tango: Simplifying SDN control with automatic switch property inference, abstraction, and optimization. In *ACM International on Conference on Emerging Networking Experiments and Technologiesmerging Networking Experiments and Technologies*, pages 199–212. ACM, 2014.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[13] V. Navda, A. Bohra, S. Ganguly, and D. Rubenstein. Using channel hopping to increase 802.11 resilience to jamming attacks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 2526–2530. IEEE, 2007.

[14] H. Okhravi, T. Hobson, D. Bigelow, and W. Streilein. Finding focus in the blur of moving-target techniques. *Security & Privacy, IEEE*, 12(2):16–26, 2014.

[15] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. Proceedings of the IEEE, 1975.

[16] S. Scott-Hayward, G. O'Callaghan, and S. Sezer. Sdn security: A survey. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7, Nov 2013.

[17] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh. On the effectiveness of address-space randomization. In *ACM Conference on Computer and Communications Security*, pages 298–307. ACM, 2004.

[18] C. A. Shue, A. J. Kalafut, M. Allman, and C. R. Taylor. On building inexpensive network capabilities. *ACM SIGCOMM Computer Communication Review*, 42(2):72–79, 2012.

[19] Twisted Framework Developers. Python twisted framework. https://twistedmatrix.com/.

[20] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen. Scotch: Elastically scaling up SDN control-plane using vswitch based overlay. In *ACM International on Conference on Emerging Networking Experiments and Technologies*, pages 403–414. ACM, 2014.