Whack-a-Mole: Software-defined Networking driven Multi-level DDoS defense for Cloud environments

Minh Nguyen*, Amitangshu Pal†, Saptarshi Debroy*
* Computer Science, City University of New York, New York, NY 10065
† Computer and Information Sciences, Temple University, Philadelphia, PA 19121
Email:{*mnguyen@gradcenter.cuny.edu, amitangshu.pal@temple.edu, saptarshi.debroy@hunter.cuny.edu*}

***Abstract*—With wider adoption of Software-Defined Networking (SDN), network obfuscation and resource adaptation within a cloud environment have emerged as cost-effective solutions against cyber attacks. In spite of their implementation simplicity, shortcomings of such one-dimensional strategies are considerable against sophisticated attacks where the attacker/s have enhanced visibility to the cloud network. In this paper, we propose '`Whack-a-Mole`', a SDN-driven cloud resource management scheme through network obfuscation that can help Cloud Service Providers (CSPs) to: a) proactively protect critical services from impending DDoS attacks and b) contribute very little service interruption footprint while doing so. '`Whack-a-Mole`' works at two levels: it employs a novel virtual machine (VM) spawning model that not only creates multiple VM-replicas of critical services to new cloud resource instances, but also assigns VM-replicas' IP addresses through address space randomization. Using numerical results, we show how such VM spawning can be optimized based on realistic cloud Service Level Agreements (SLA) without compromising its effectiveness. Finally, '`Whack-a-Mole`' is implemented through SDN/OpenFlow controllers over Open vSwitches on a GENI testbed where the efficacy and effectiveness of the scheme is evaluated. The results show '`Whack-a-Mole`' to be as effective as random obfuscation in evading attack events and more than 2x better on average in attack avoidance over other static resource adaptation based defense strategies.**

***Index Terms*—Cloud Security, Address Randomization, Moving Target Defense, DDoS Attacks, Software-defined Networking.**

I. INTRODUCTION

Network based Distributed Denial of Service (DDoS) attacks are one of the most common threats impacting all types of cloud based services ranging from finance industry to game servers. The targets of such DDoS attacks include any and all types of cloud service providers (CSP), from large enterprise clouds to small scale campus private clouds causing billions of dollars in damages to cloud providers and tenants. In most common volumetric DDoS attacks targeting cloud resources, the attacker/s achieve partial or complete disruption of regular service users' quality of experience (QoE) by flooding the virtual machine (VM) hosting the service with huge volume of packets. The packets consume the VM resources (network, compute, and storage) thus leaving the VM unresponsive to regular users' requests. Such attacks continue to bother the cloud industry as according to the State of the Internet Survey [1], overall DDoS attacks are up (14%) in 2017 Q4 in comparison to 2016 and more than 50% of all attacks are volumetric flooding attacks.

With the advent of Software-defined Networking (SDN) [2] and OpenFlow [3], the traditional 'detect-and-react' DDoS defense approaches [4] are transformed into more proactive 'Cyber Agility and Defensive Maneuver' mechanisms [5]. These are effective in consistently maintaining the Service Level Agreements (SLA) between the providers and tenants and help protect millions of dollars worth of cloud resources. Such mechanisms can be designed to: (a) tackle intelligent and sophisticated attacks by being one step ahead, (b) be agile in responding to any attack trigger indicating impending attack, (c) be cost effective in prudent utilization of cloud resources while maneuvering. Amongst the 'Cyber Agility and Defensive Maneuver' strategies, the Moving Target Defense (MTD) [6] based resource obfuscation/adaptation mechanisms empowered by SDN can be effective to protect critical cloud-hosted applications. In order to make the implementation of MTD based schemes simpler, most of the state-of-the-art mechanisms however employ one-dimensional maneuvers that are relatively easier to predict over time. This is especially true when the attacker gains enhanced visibility of the network in terms of the IP address used for the VMs hosting the applications.

In this paper, we propose `Whack-a-Mole`: a SDN-driven MTD based DDoS defense mechanism in cloud environments. Through the SDN controller, `Whack-a-Mole` works at two levels: first it proactively spawns replicas of VMs hosting critical applications where the applications are seamlessly migrated. Next, it mutates the IP addresses associated with the services by assigning the VM replicas with IP addresses belonging to different Address Spaces (ASes) where the entire cloud network is divided into different ASes. The OpenFlow switches with the help of SDN controller direct all new incoming user (i.e., clients of cloud hosted services) requests to the spawned VMs whereas the existing users are allowed to finish their sessions with the old VMs. Upon completion of the existing users' sessions, the VMs are allowed to die by freeing up the cloud resources and IP addresses that will be reused for newly spawned VMs. In `Whack-a-Mole`, we optimize the frequency of such spawning in order to keep it frequent enough to thwart an impending attack, yet not so frequent that it disrupts the cloud user QoE or satisfaction. At the same time, the address mutation is optimized to keep the new IP address selection as unpredictable as possible for the attacker to guess. As a result, `Whack-a-Mole` is able to create a network resource obfuscation process that protects critical services from DDoS attacks and upholds the SLA bounds at the same time.

We demonstrate the utility of `Whack-a-Mole` using a GENI [7] based testbed implementation and evaluation. Using GENI, we create a typical CSP and compare the performance of proposed address mutation scheme against a completely random mutation scheme and a deterministic one-dimensional state-of-the-art mutation scheme. The results show that against an intelligent attacker with partial visibility of the network and probabilistic learning capability, `Whack-a-Mole` mutation scheme is able to achieve as low probability of attack success as a random mutation scheme. Whereas, `Whack-a-Mole`'s probability of attack success improvement in comparison to a deterministic one-dimensional mutation scheme is more than 2-fold on average.

The rest of the paper is organized as follows: Section II discusses the related work. Section III presents the system and attack model. Section IV discusses the scheme and optimization details. Section V discusses the GENI testbed implementation and resilience performance evaluations. Finally, Section VI concludes the paper.

II. Related Work

The related work in MTD based DDoS attack defense can be divided into three broad classes.

A. MTD-based cloud security solutions

In recent times, cloud based MTD works are gaining momentum in tackling cloud based threats, among these, [8–10] are notable. In [8], the authors propose a MTD strategy to marginalize the attackers within a small pool of decoy VMs. Other notable work that applies MTD against cyber attacks on VMs is [10] where the authors use VM duplication with consumers redirected whenever the VMs running the critical applications are attacked. In [9], the authors propose the one of the very first efforts on network diversity based MTD for cloud security. *Most of these works are one-dimensional in their defense strategy design and thus prone to failure against sophisticated attacks with greater network visibility.*

B. SDN enabled cloud security strategies

Works such as, [11–13] proposed SDN enabled MTD cloud security schemes where either VM IP address mutation schemes or VM migration schemes use OpenFlow [3] to route cloud users to the target applications. Authors in [11] studied the benefits and overheads of SDN-enabled MTD schemes for VM migration. In [12], the authors perform an online VM migration strategy by predicting impending attacks using attack traffic signature pattern recognition. While in [13], authors use SDN to compute the ideal VM for migration based on multiple factors. *The common theme in most of such SDN enabled MTD schemes are the ease of implementation ensuring successful misdirection of attackers with limited budget and network know-how.*

C. Network address shuffling and randomization

Network address shuffling and randomization is one of the most popular implementations of MTD in cloud [14–17]. Authors in [14] analyze the effectiveness of network address shuffling by computing the probabilities and expectations of at least one vulnerable VM getting attacked. In [15, 16], the authors propose an optimal IP mutation scheme of allocating IP addresses to each VM in the system. In [17], the authors shuffle IP address among large number of virtual decoy nodes; whenever the decoys get attacked, the shuffling scheme adapts and reconfigures. *However, such schemes assume infinitely large address space and tend to be too wasteful to be implementable.*

III. System and Attack Model

In this section, we discuss the system and attack model considered for this work.

A. System model

For `Whack-a-Mole`, we assume that the entire set of available IP addresses (does not need to be contiguous) is divided into smaller address spaces (ASes) of equal size a. We also assume that the ASes are more than the number of VMs, so that there are enough redundancy. We also assume that within a cloud network, there are different subnets, each consists of a set of hosts physically connected through a switch. In such a scenario, typical IP address mutation schemes [16, 17] mutate the VM addresses within the address space of its own subnet, which reduces the space of mutation and reduces the level of unpredictability. At the same time those schemes suffer against a more visible attacker which can somehow compromise the subnet switch. To cope with this, we propose a novel model of mutating (or spawning) the VM copies across the subnets or in our case ASes.

Figure 1 shows the logical representation of `Whack-a-Mole` process of spawning VMs into different ASes aided by the SDN/OpenFlow controller. The controller coordinates the spawning process through multiple Open vSwitches (OVSes), i.e., switches that support OpenFlow along the spawning path. The SDN/OpenFlow controller governs the data flow to the VMs through the OVSes and load balancers using the control path. The VMs share their control/status information with the controller along the control path. As shown in Figure 1, the regular users access the application through the controller and OVSes along the regular path. Our scheme also assumes the presence of state-of-the-art intrusion detection system within the cloud network that can detect impending attack and direct the attacks on the target application to an old VM replica. The VMs under attack is thus treated as a quarantine and the associated IP address is not reused.

B. Attack model

In this work, we consider attacks on a target application, (i.e., on a target VM) generated from one or multiple attackers to be a Poisson point process [18, 19] with exponentially

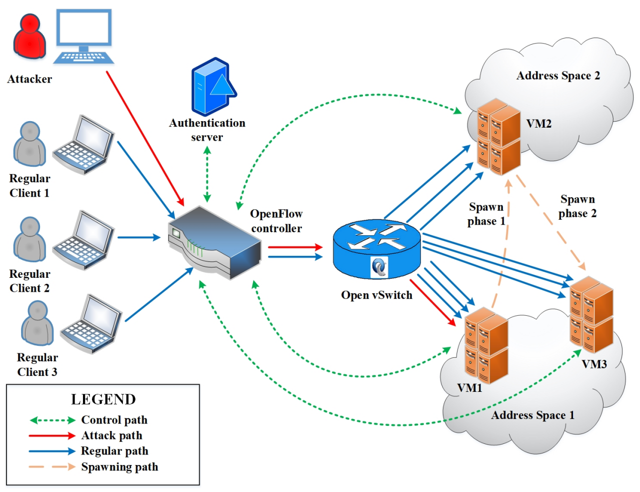


Fig. 1: The cloud system model for `Whack-a-Mole`

distributed attack inter-arrival times. In our model, every VM experiences two states: Attacked, i.e., when the target VM is either under attack or being actively probed before an attack; and Idle, when the VM is under no active attacks. As shown in Figure 1, the attacker attacks the target application, or more specifically the VM hosting the application along the *attack path*. We assume the attacker/s to have partial visibility to the cloud network, i.e., the attacker has the knowledge of IP addresses and ASes used for mutation, although the mutation process is unknown. However, we assume that the attacker can employ probabilistic learning mechanisms to guess the mutated IP address post VM spawning.

IV. `Whack-a-Mole`: Scheme Details and Optimization

In this section we describe the overview of the proposed `Whack-a-Mole` scheme. We first frame an optimization problem to determine the spawning frequency of the VMs, followed by the optimization outcome based on numerical results. We next describe the overall spawning procedure of the VMs across the subnets using the optimal spawning frequency.

A. Scheme overview

During the VM spawning procedure, `Whack-a-Mole` needs to ensure that the ongoing users and packets are least affected during the process. This can be achieved as follows. Every VM has a *lifetime*. During this time it spawns another VM (or replica), which will operate similarly (will have a lifetime and spawn), on a different IP address, maybe on a different subnet. So in this model, at any instance there will be a number of VM instances, and the new users will be directed to the newest VM. The old VMs will gradually die (or become ineffective) after their lifetime expires. Thus in this model only a small fraction of the users (only the ongoing ones) may need to be redirected (when their VM will die) to the newer VM. Also in this model *a single attack cannot affect all the users*, as the users are distributed among multiple VM replicas. Thus by dynamically spawning the VM replicas across the subnets, *this model increases the level of unpredictability from an attackers perspective, and at the same time ensures less user redirection, thus fulfilling our dual objectives of robustness and user satisfaction.*

TABLE I: Table of Notations

n	$\triangleq$	Number of VMs
M	$\triangleq$	Total number of ASes
N	$\triangleq$	Total number of subnets
I	$\triangleq$	Interval duration
a	$\triangleq$	Number of addresses in an AS
c_i	$\triangleq$	Spawning cost of VM$_i$
$\lambda_a = 1/t_a$	$\triangleq$	R.V. for average DDoS attack frequency
$\mu_i = 1/t_i$	$\triangleq$	R.V. for average idle period frequency
T_i	$\triangleq$	VM$_i$ lifetime
b^i_j	$\triangleq$	Whether AS$_j$ is assigned to VM$_i$ in the current interval
B^i_j	$\triangleq$	Whether S$_j$ is assigned to VM$_i$ in the current intervals
x^i_j	$\triangleq$	Number of times AS$_j$ is chosen for VM$_i$ in the last $\omega - 1$ intervals
X^i_j	$\triangleq$	Number of times S$_j$ is chosen for VM$_i$ in the last $\omega - 1$ intervals
y^i_j	$\triangleq$	Number of times AS$_j$ is assigned to VM$_i$ in the last ω intervals
z^i_j	$\triangleq$	Number of times AS$_j$ is assigned to S$_i$ in the last ω intervals
$R_i = 1/\gamma_i$	$\triangleq$	Spawning rate of VM$_i$

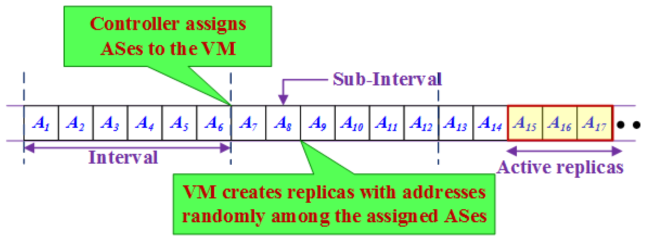

Controller assigns ASes to the VM
Sub-Interval
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
A16
A17
Interval
Active replicas
VM creates replicas with addresses randomly among the assigned ASes

Fig. 2: Illustration of intervals, sub-intervals and active replicas corresponding to a VM

We assume that time is divided into *intervals*. In each interval, the controller assigns some address spaces to each VM, so that two or more VMs are not assigned to the same address space. In the next interval, the VM is assigned a different set of ASes. Each interval is further divided into several sub-intervals. In each sub-interval, a VM *spawns* another *instance* (or *replica*) of that VM to a different IP address. It does this by uniformly randomly choosing an IP address within its own ASes that were assigned to it (by the controller) in that interval. The chosen IP address may or may not reside in the same subnet. Figure 2 shows an illustration of the active replicas corresponding to a certain VM, where $A_1 - A_{17}$ denotes different IP addresses..

Figure 3 shows the overview of the proposed scheme. In Figure 3 the spawning interval and VM lifetime are assumed to be γ_i and T_i respectively. Thus after γ_i, the replica a_1 spawns another replica a_2, which spawns another one after its spawning interval and so on. The replicas die after their lifetime, so that their addresses can be reused for future replicas. The detailed scheme is described in the following subsections. The effectiveness and efficiency of our proposed scheme, thus, relies on its ability to optimize the spawning frequency and VM lifetime that (a) does not increase the DDoS attack probability, (b) and at the same time ensures minimum user redirection and VM spawning overhead. In this section, we first outline and formalize the optimization problems for spawning frequency, and then use it to build the problem formulation of dynamic VM spawning scheme across the address spaces. Table I summarizes the notations used for our modeling.

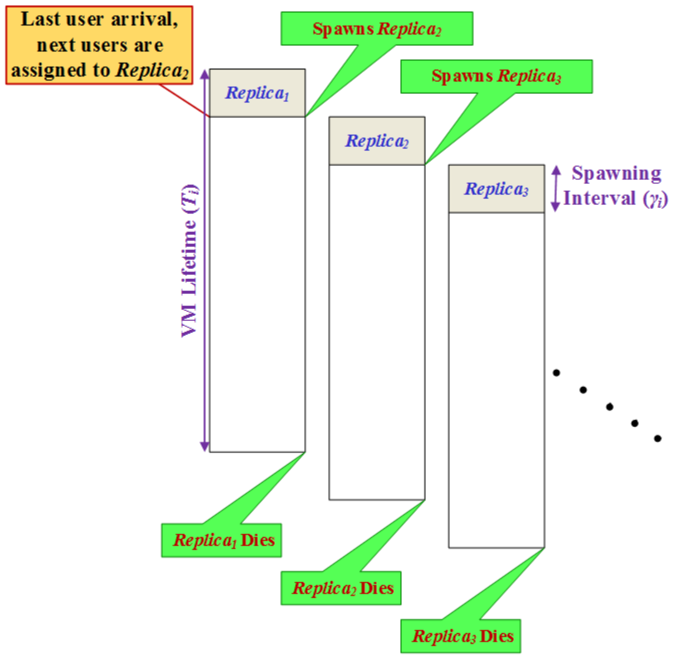


Fig. 3: Schematic of `Whack-a-Mole` VM spawning process

B. VM lifetime and spawning frequency optimization

In Figure 3 we can observe that last user/customer of a replica uses the VM for atmost $T_i - \gamma_i$ time units. Thus if the spawning interval γ_i is increased (of spawning frequency is decreased), the VM lifetime needs to be increased as well to ensure that all the users can be served by the VM. Thus the VM lifetime is proportional to the spawning interval and vice versa.

The ideal VM spawning frequency should be such that it is not too infrequent to make the VM vulnerable to DDoS attacks (as infrequent spawning frequency increases the VM lifetime). At the same time, the frequency should not be too often to waste valuable cloud network resources. To solve this frequency optimization problem, we assume the lifetime of VM$_i$ to be T_i which ideally should be infinitely large (and so is γ_i) if there is no DDoS attack, thus minimizing the network resource wastage. However, due to threats of DDoS attacks, T_i needs to be adjusted just enough so that it is less than the DDoS attack inter-arrival time. This optimization problem can be formulated as follows:

$$\begin{aligned} &\text{Minimize} \quad T_i \\ &\text{s.t.} \quad T_i \leq \text{DDoS attack inter-arrival time} \end{aligned} \tag{1}$$

The constraint in Eq. (1) ensures that the all users using the services of VM i during its lifetime do not encounter an attack. Here, T_i will be a function of mean attack period duration and idle period duration (attack inter-arrival times) t_a and t_i, i.e., in other words, T_i will be dependent on λ_a and μ_i. The exponentially distributed random variable for attack period duration x with mean $t_a = \frac{1}{\lambda_a}$ is given by

$$f_1(x) = \begin{cases} \lambda_a e^{-\lambda_a x} & \forall\ x \geq 0 \\ 0 & \forall\ x < 0 \end{cases} \tag{2}$$

Similarly, the exponentially distributed random variable for idle period duration y with mean $t_i = \frac{1}{\mu_i}$ is given by

$$f_2(y) = \begin{cases} \mu_i e^{-\mu_i y} & \forall\ y \geq 0 \\ 0 & \forall\ y < 0 \end{cases} \tag{3}$$

Let us assume that the random variable representing the attack inter-arrival time be $\mathbf{z}$ which is the sum of two independent random variables for *Attacked* and *Idle* periods $\mathbf{x}$ and $\mathbf{y}$ respectively, i.e., $\mathbf{z} = \mathbf{x} + \mathbf{y}$. Therefore, the distribution of attack inter-arrival time $\mathbf{z}$ is obtained as:

$$\begin{aligned} f_Z(z) &= f_X(x) * f_Y(y) \\ &= \int_{-\infty}^{+\infty} f_X(z-y) f_Y(y) dy \\ &= \begin{cases} \frac{\lambda_a \mu_i [e^{-\lambda_a z} - e^{-\mu_i z}]}{(\lambda_a - \mu_i)} & \forall\ \lambda_a \neq \mu_i \\ \lambda_a^2 z e^{-\lambda_a z} & \text{otherwise} \end{cases} \end{aligned} \tag{4}$$

In order to quantify the optimal T_i, we approach the problem by first calculating the probability of VM being attacked before it dies. Such a probability is expressed as:

$$\begin{aligned} &\text{Prob}\{\text{VM getting attacked within its lifetime}\} \\ &= \text{Prob}\{z \leq T_i\} \qquad \text{(VM attack being memoryless)} \\ &= \int_{-\infty}^{T_i} f_Z(z) dz \\ &= \begin{cases} \int_0^{T_i} \frac{\lambda_a \mu_i [e^{-\lambda_a z} - e^{-\mu_i z}]}{(\lambda_a - \mu_i)} dz & \forall\ \lambda_a \neq \mu_i \\ \int_0^{T_o} \lambda_a^2 z e^{-\lambda_a z} dz & \text{otherwise} \end{cases} \\ &= \begin{cases} \frac{\mu_i (e^{-\lambda_a T_i} - 1) + \lambda_a (1 - e^{-\mu_i T_i})}{\lambda_a - \mu_i} & \forall\ \lambda_a \neq \mu_i \\ 1 - e^{-\lambda_a T_i}(\lambda_a T_i + 1) & \text{otherwise} \end{cases} \end{aligned} \tag{5}$$

Now in order to optimize problem (1), probability of VM getting attacked before it dies, i.e., Prob$\{z \leq T_i\}$ needs to be minimized. This reduces the optimization problem from equation (1) to:

$$\boxed{\text{Minimize} \quad \left(\text{Prob}\{z \leq T_i\}\right)} \tag{6}$$

However, due to the asymptotic nature of exponentially distributed random variable **z**, the nature of Eq. (5) is continuously increasing and asymptotic; and thus does not have any maxima or minima. Therefore, for a particular DDoS attack scenario (i.e., with statistical λ_a and μ_i known), the optimal T_i can be evaluated by tuning the desired probability of VM getting attacked, i.e., Prob$\{z \leq T_i\}$. However, in our model the VM lifetime is also dependent on the spawning interval, user's service time and spawning costs, which gives rise to the following set of constraints.

User satisfaction constraint: We assume that the spawning interval) of VM$_i$ is given by γ_i. Also assume that the user's service time is s, which is exponentially distributed with mean μ_u. Then $T_i - \gamma_i$ (see Figure 3) should be sufficient enough so that the last customer can finish its service within the lifetime of the VM, with high probability, i.e.

$$\text{Prob}\left(s > T_i - \gamma_i\right) < \tau_1 \tag{7}$$

where τ_1 is an user defined threshold.

Spawning cost constraint: Also there is a spawning cost associated with a VM, which is different for different VMs. The spawning cost of a VM$_i$ is the sum of its snapshot cost, and the network cost of imitating/copying snapshot files to a different location. Assume that c_i is the spawning cost of VM$_i$, then the total spawning cost in an interval is $c_i I / \gamma_i$, should be less than some cost budget τ_2, i.e.

$$c_i I / \gamma_i < \tau_2 \tag{8}$$

where τ_2 is an user defined cost budget.

Thus there is a clear tradeoff between the spawning cost of a VM, its lifetime and its security under DDoS attack. If the cost of a VM spawning is high, its spawning interval is also high. Thus its lifetime should also needs to be higher to fulfill objective (6). On the other hand, the lifetime cannot be too high to make the VM vulnerable to attack.

C. Effects of VM lifetime and spawning frequency optimization

Using MATLAB based numerical results, we demonstrate how the VM lifetime and thereby VM spawning interval can be optimized based on different service and system parameters.

We first characterize Eq. (5) against T_i as shown in Figure 4(a). We can observe how the Prob$\{z \leq T_i\}$ gradually increases and achieves saturation with larger values of VM lifetime T_i. However, the slope of the probability increase is dependent on the attack budget during the attack and idle periods t_a and t_i respectively. Consequently, the T_i optimization is adaptive to: (a) the attack budget, and (b) the tolerable limit of DDoS attack success. For example, if the attack budget is high, i.e., an application is attacked every 10 seconds with 100 seconds of duration between consecutive attacks, and if the system can handle up to 1 out of 10 successful attacks, the optimal T_i should be around 50 seconds. Whereas for lower attack budgets, in order to maintain the same attack success tolerance, VM lifetime of 1000 seconds will suffice.

Figure 4(b) shows the attack success probability with different spawning duration γ_i for different attack budgets. We assume $\tau_1 = 0.1$ for Figure 4(b). The characteristics are similar to Figure 4(a) as we observe increasing Prob$\{z \leq T_i\}$ with spawning interval γ_i. This is because the spawning interval is proportional to the VM lifetime. We can also observe that in order to safeguard a VM during its lifetime, the spawning needs to be more frequent with increasing attack budget. For example, if we keep a constant spawning interval of 1000s against a 100x attack budget increase (i.e., from 10s/10000s to 10s/100s), then the attack success probability increases by more than 4x.

We next demonstrate the characterization of attack success probability against the spawning cost constraint c_i in Figure 4(c). We assume $\tau_2 = 0.1$ and $I = 30$ seconds for Figure 4(c). We can observe that for a fixed attack budget, the attack probability clearly increases with higher spawning cost. This is because higher spawning results in an increase in the spawning interval, which increases the VM lifetime and this attack success probability.

D. Spawning VMs at Unpredictable IP Addresses

With the established spawning rate and lifetime corresponding to a VM, we next formulate our MTD based VM spawning problem. The MTD is run at the SDN controller at every interval to assign the ASes to the VMs. The objective of the controller is mainly twofold. The first factor is to maximize the entropy (or unpredictability) of assigning a VM to a particular AS. The second factor is to maximize the unpredictability of assigning a VM to a particular subnet. The key purpose of this twofold objective is to ensure that the VMs not only spawns at different addresses, but also across different ASes.

The first factor can be modeled as follow. Assume that b_j^i and B_j^i are binary variables which are one if AS$_j$ and subnet S_j are assigned to VM$_i$ respectively in the current interval. Also assume that x_j^i and y_j^i is the number of times AS$_j$ is assigned to VM$_i$ in the last $\omega - 1$ and ω (including the current interval) intervals respectively. Then the number of times AS$_j$ is assigned to VM$_i$ is given by $y_j^i = x_j^i + b_j^i$. Thus the probability (or fraction of times) of assigning AS$_j$ to VM$_i$ in the last ω intervals is given by $p_j^i = y_j^i / \sum_{j=1}^{M} y_j^i$. Similarly let us assume that X_j^i and z_j^i is the number of times subnet S_j is assigned to VM$_i$ in the last $\omega-1$ and ω intervals respectively, then $z_j^i = X_j^i + B_j^i$. Then the probability of assigning VM$_j$ to a subnet S$_i$ is given by $q_j^i = z_j^i / \sum_{j=1}^{N} z_j^i$. Thus our objective function to maximize the overall unpredictability which can be modeled as

$$\boxed{\text{Maximize} \quad \sum_{j=1}^{M} p_j^i \log\left(\frac{1}{p_j^i}\right) + \sum_{j=1}^{N} q_j^i \log\left(\frac{1}{q_j^i}\right)} \tag{9}$$

The objective is to find out the optimal set of b_j^i corresponding to the VMs, such that the following constraints are satisfied.

Integrity constraint: The address spaces chosen corresponding to a particular VM as well as its chosen subnet should be consistent, i.e., if an AS is chosen corresponding to a VM, then the subnet where the AS resides should also be assigned

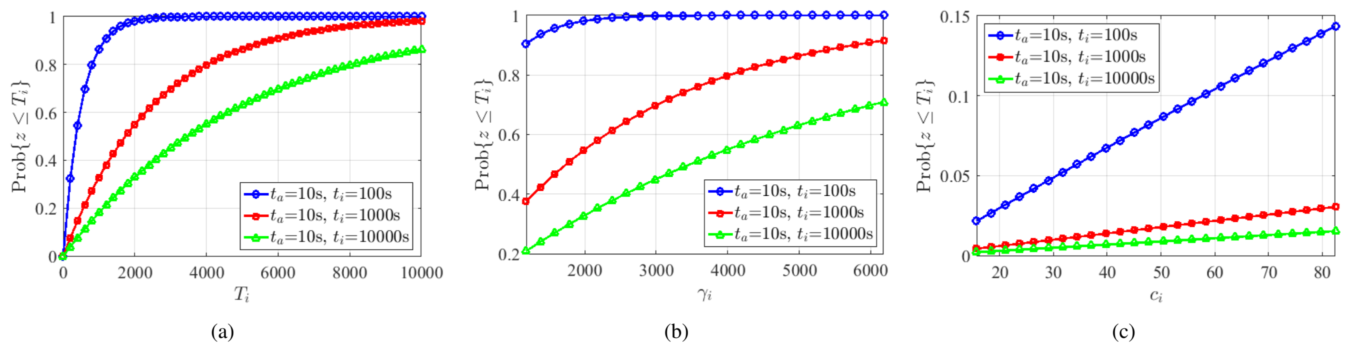


Fig. 4: Variation of attack success probability with (a) different VM lifetimes, (b) spawning intervals and (c) spawning costs.

to that VM. These give rise to the following set of constraints:

$$\begin{aligned} \sum_{j=1}^{M} b_j^i &\ge B_k^i \quad \forall AS_j \in S_k \\ L.B_k^i &\ge \sum_{j=1}^{M} b_j^i \quad \forall AS_j \in S_k \end{aligned} \tag{10}$$

where L is a very large number, which ensures that if $b_j^i = 1$, then $B_k^i = 1$ if and only if AS$_j$ resides in subnet-k. On the other hand, if $B_k^i = 1$ then at least one of the AS$_j \in S_k$ has to be 1.

Spawning rate constraint: At any interval, each VM receives enough address spaces depending on their spawning rates, i.e. the VMs with higher spawning rates receive more ASes and vice versa. If R_i is the spawning rate of VM$_i$, then

$$\max_{\forall i}\left(\frac{\sum_{j=1}^{M} b_j^i}{R_i}\right) - \min_{\forall i}\left(\frac{\sum_{j=1}^{M} b_j^i}{R_i}\right) < \delta \tag{11}$$

i.e., the ratios of the address spaces allocated to a VM and its spawning rate is more-or-less identical. Here δ is an user defined constant. Also the number of addresses assigned to a VM should be more than the addresses it needs, i.e.,

$$a.\sum_{j=1}^{M} b_j^i > \alpha I R_i \quad \forall i \tag{12}$$

where a is the number of addresses in a subspace, and α is a constant.

Other constraints: At any interval, all the ASes are assigned to a VM. This ensures the utilization of all the available ASes, which maximizes the level of unpredictability. We also need to ensure that no two VMs are assigned to the same address space, i.e.,

$$\sum_{i=1}^{n} b_j^i = 1 \quad \forall j \tag{13}$$

This constraint satisfaction problem can be solved by finding all satisfiable b_j^j using one of Satisfiability Modulo Theories (SMT) [20] solvers such as Z3 Prover [21].

Within an interval the VMs spawn their replicas at the addresses that are chosen uniformly randomly among the ASes assigned to that VM, without invoking the central controller. This is depicted in Figure 2. This makes `Whack-a-Mole` highly scalable when it comes to managing large number of VMs.

V. System Implementation and Evaluation

In this section, we describe the performance evaluation of `Whack-a-Mole` on the GENI Cloud infrastructure [7].

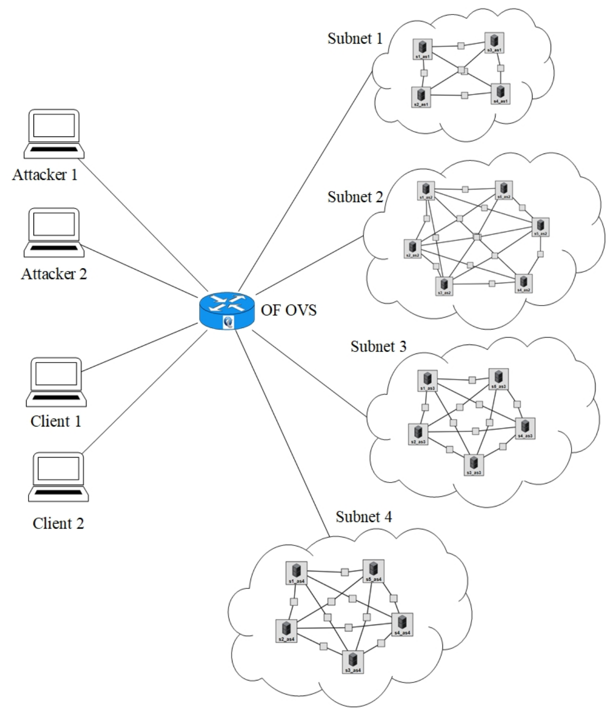


Fig. 5: GENI cloud network experimental topology.

A. Testbed setup

We setup the GENI testbed by creating multiple slices that represent ASes or subnets and VMs that represent the servers hosting applications. The overall setup is shown in Figure 5. The OVS with built-in SDN functionality is the controller; it lies on a different slice and runs `Whack-a-Mole`. All the VMs in subnet slices are connected directly to the OVS. Since GENI cloud infrastructure does not allow dynamic 'on the fly' VM generation once the resources are reserved, we create empty stand-by VMs beforehand in order to act as dynamically spawned VMs. For example, in Subnet 1, only one server among many is up at the beginning of the experiment. When the controller needs to spawn a new VM, it will dynamically activate one of inactives. Overall, we use four subnets with 20 total servers or VMs.

For the SDN functionality, we use open-source POX controller that is available on GENI platform's OVS to implement our `Whack-a-Mole` algorithm. For every flow that goes through the OVS/controller, the routing part is dealt with 'Control Plane'; while the SDN controller governs the 'Data Plane'. For the DDoS attacks, we install two attackers that use 'slowhttptest' as attack tool. It is a compilation of various DOS attack tools such as 'slowloris', 'Slow HTTP POST', and 'Slow Read attack' (based on TCP persist timer exploit) [22]. We use it on two Linux VMs acting as attackers to launch a combined Layer 3 (TCP) and Layer 7 (HTTP request) attack to the target application.

B. Experimental setup

The objective of the experiments is to compare the IP address mutation performance of `Whack-a-Mole` with two other candidate mutation schemes:

Deterministic: In this scheme the individual VMs spawn at the addresses in a deterministic, round-robin fashion. This type of scheme is employed by most one-dimensional state-of-the-art VM migration based MTD mechanisms.

Random: In this scheme the new VM-replica addresses are chosen uniformly randomly. Ideally if the addresses are chosen randomly, then it maximizes the level of unpredictability. We want to evaluate how close `Whack-a-Mole` performs in comparison to the random mutation scheme.

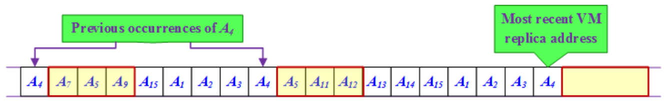

Previous occurrences of A_4
Most recent VM replica address
A_4 A_7 A_5 A_9 A_{15} A_1 A_2 A_3 A_4 A_5 A_{11} A_{12} A_{13} A_{14} A_{15} A_1 A_2 A_3 A_4

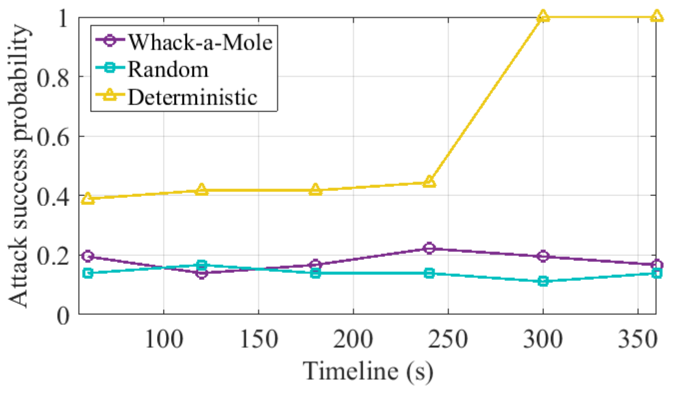
Fig. 6: Illustration of the attackers learn-ability.

For the DDoS experiments, we design attacks with varied learn-ability and visibility. At each sub-interval, the attacker chooses the address for launching his next attack for the next sub-interval to a particular VM. We assume that the VM address in the current sub-interval is i. For each address-j, the attacker calculates the probability that the VM will create a replica with address-j in the next sub-interval, conditioned on the fact that the current VM address is i. In order to achieve it, the attacker keeps track of a *window* of w sub-intervals. Thus, the probability $\mathcal{B}_j$ that j will be assigned to a VM in the next window is calculated as the ratio of the number of times address-j is assigned to that particular VM within the "look-ahead" window of size w, and the total number of times address-j is assigned to that VM. Thus the probability of attacking j in the next sub-interval is $\mathcal{B}_j/w$.

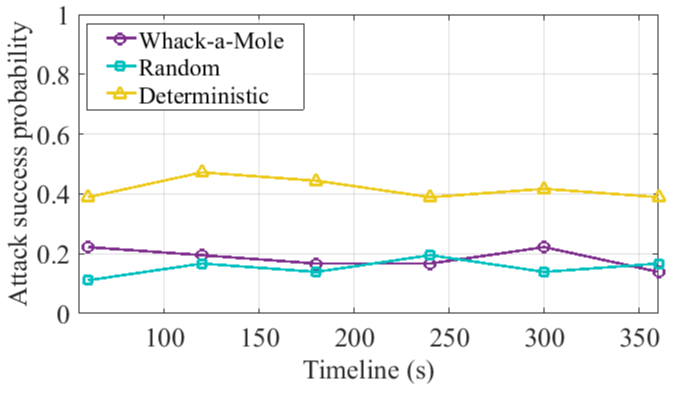
Figure 6 shows how the attacker predicts the next IP address corresponding to a particular VM. In Figure 6 we assume that the look-ahead window size $w = 3$, and assume that the current VM address is A_4. Now based on the historical spawning of that VM, we can observe that A_5 comes twice within the look-ahead window after A_4, whereas A_7, A_9, A_{11}, A_{12} appears once. Thus $\mathcal{B}_5 = 1$, $\mathcal{B}_7 = \mathcal{B}_9 = \mathcal{B}_{11} = \mathcal{B}_{12} = 0.5$. $\mathcal{B}_j$ is zero for all the other addresses. Thus the attacker chooses A_5 with a probability of $\mathcal{B}_5/w = 1/3$, whereas A_7, A_9, A_{11}, A_{12} are chosen with a probability of 1/6 each.

The look-ahead window size w is basically an abstraction that models how accurately a VM's next spawning address is predicted by an attacker. Notice that such attack strategy is effective mainly in case of deterministic (or semi-deterministic) scheme. In case of a deterministic scheme, lesser values of w results in higher attack success probability and vice versa. Ideally w can be anywhere between 1 and the number of VMs in the system. The key purpose of `Whack-a-Mole` is to improve the level of unpredictability during the spawning process so that such learning cannot help the attacker to launch a successful attack.

The performance comparison between `Whack-a-Mole`, Deterministic, and Random schemes is made in terms of 'instances of successful attack' and 'probability of successful attack'. An attack is considered successful if mutated IP address of the replica VM upon spawning is the same as the IP address of the VM under attack. Thus, these metrics measure the unpredictability of the mutation schemes.



(a)



(b)

Fig. 7: Attack success probability with (a) w = 1 and (b) w = 3.

C. Effects of VM spawning at unpredictable addresses

We first observe the attack success probability for each candidate scheme for an experiment duration of 360 seconds. The attacker generates a new IP address to attack and the system spawns a new replica with a new IP address at every 20 second. Thus, the experiment only measures `Whack-a-Mole`'s efficacy in terms of VM spawning at unpredictable IP addresses and not spawning rate efficiency. Each candidate scheme is tested independently.

Figure 7(a) illustrates the attack success probability for the total runtime of the test (i.e., 360 seconds) with the attacker's look-ahead window size of $w = 1$. Evidently, attacker's learning mechanism adapts with the Deterministic scheme's pattern after a while. In case of Deterministic scheme, the attacker's ability to precisely identify the new VM-replica addresses results in gradual increase in attack success probability. In fact the attack success probability goes to 1 after 300 seconds. However, due to the unpredictability of Random and `Whack-a-Mole` schemes, the success rate remains as low as $\sim$0.2.

We next repeat the experiment with the attacker's look-ahead window size of $w = 3$. In case of Deterministic scheme, this signifies less precise attacks. This is shown in Figure 7(b), where the attack success probability remains $\sim$0.4 throughout the span of our experiment. However, the success rate in case of `Whack-a-Mole` and Random schemes remain $\sim$0.2. We can also observe that the attack success probability is independent of the look-ahead window size in case of `Whack-a-Mole` and Random schemes, i.e. the learn-ability of the attacker does not help in launching successful attacks.

Figures 8(a)-8(b) show the number of successful attacks at individual VMs with the look-ahead window size of 1 and 3 respectively. From these figures we can observe that `Whack-a-Mole` and Random schemes exhibit near identical performances with very few instances of successful attacks for each VM. In case of Deterministic scheme, increasing the window size from 1 to 3 reduces the attack success rate by upto 3 times for some VMs. However, the performance of `Whack-a-Mole` and Random schemes remain identical with the variation in window sizes, and is signiificantly better than the Deterministic scheme.

D. `Whack-a-Mole` vs Random mutation

One relevant question in this context is the following: "If random address mutation gives optimal solution, then why do we need `Whack-a-Mole`?". The key reason of using `Whack-a-Mole` is its *scalability*. In case of random mutation, the controller needs to generate random addresses in each sub-interval to ensure that no two VMs are assigned the same address. This requires frequent intervention of the controller, and required message exchanges between the controller and the VMs. This is onerous in a large cloud environment. To cope with this, `Whack-a-Mole` invokes the controller once in an interval to assign non-overlapping address spaces to each VM while maximizing the level of unpredictability. Through experimental evaluations we claim that `Whack-a-Mole` still provides identical resiliency compared to the random mutation scheme.

VI. Conclusion and Future Work

In this paper, we proposed `Whack-a-Mole`, a SDN driven MTD based network obfuscation scheme that works at two levels: spawning of VM replicas across the subnets and the choice of the replica addresses to improve the level of unpredictability. Using numerical results, we showed how `Whack-a-Mole` optimizes spawning frequency in order to reduce attack success probability, yet limiting the user service interruption. At the same time, using cloud scale GENI testbed, we demonstrated how `Whack-a-Mole`'s replica creation and their chosen IP addresses are able to achieve close to optimal unpredictability. The results of this work can help the cyber security and CSP communities to leverage SDN programmability in employing proactive, smart, and multi-dimensional resource obfuscation based maneuvers against DDoS attacks.

As this work seeks to design an address mutation scheme that increases unpredictability, we focus on the proposed scheme's ability to reduce 'probability of attack success' rather than 'impact of successful attacks'. We believe that such metrics can be measured by the VM response time with varying attack intensity and attack budget; both of which will be addressed in future extended version of the current manuscript.

References

[1] "Q4 2017 State of the Internet Security Report" - https://www.akamai.com/uk/en/multimedia/documents/state-of-the-internet/q4-2017-state-of-the-internet-security-report.pdf"

[2] I. Monga, E. Pouyoul, and C. Guok, "Software-defined networking for Big-Data science - Architectural models from campus to the WAN," *Proc. of IEEE SCC*, 2012.

[3] "OpenFlow Switch Specification," *Open Networking Foundation*, 2015.

[4] A. Kartit, A. Saidi, F. Bezzazi, M. El Marakki, A. Radi, "A new approach to intrusion detection system," *Journal of Theoretical and Applied Information Technology*, vol. 36, no. 2, pp. 284 289, February 2012.

[5] "Extreme DDoS Defense (XD3)" - *https://www.darpa.mil/program/extreme-ddos-defense*.

[6] Cyber Security Division, "Project: Moving Target Defense," *US Department of Homeland Security - https://www.dhs.gov/science-and-technology/csd-mtd*.

[7] "NSF GENI Infrastructure" - *https://www.geni.net*.

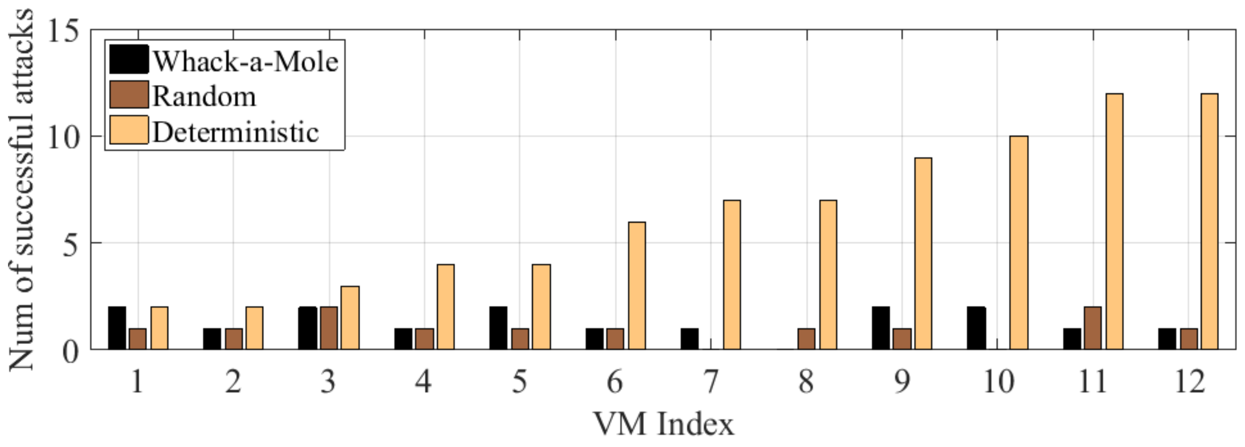
[8] Q. Jia, H. Wang, D. Fleck, F. Li, A. Stavrou, and W. Powell, "Catch me if you can: A cloud-enabled DDoS defense," in *Proc. of IEEE/IFIP DSN*, 2014.

[9] M. Pendleton, R. Garcia-Lebron, J.H. Cho, and S. Xu, "A survey on systems security metrics," *Proc. of ACM Comput. Surv.*, 2016.

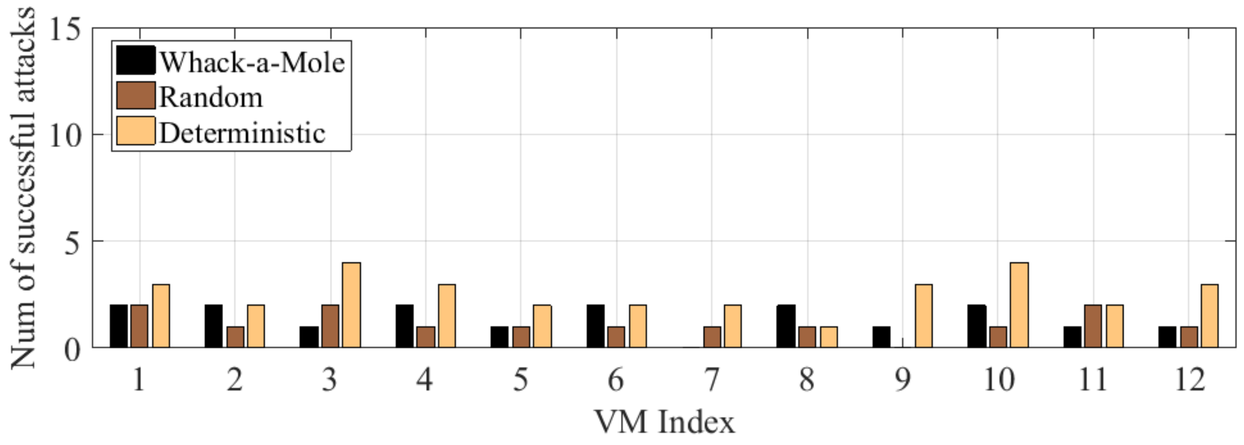
[10] W. Peng, F. Li, C. T. Huang, and X. Zou, "A moving-target defense strategy for cloud-based services with heterogeneous and dynamic attack surfaces," in *Proc. of IEEE ICC*, 2014.

[11] P. Kampanakis, H. Perros, and T. Beyene, "SDN-based solutions for moving target defense network protection," in *Proc. of IEEE WoWMoM*, 2014.

[12] R. Zhuang, S. Zhang, A. Bardas, S. DeLoach, X. Ou, and A. Singhal, "Investigating the application of moving target defenses to network security," in *Proc. of IEEE ISRCS*, 2013.



(a)



(b)

Fig. 8: Number of successful attacks at individual VMs with (a) w = 1 and (b) w = 3.

[13] S. Debroy, P. Calyam, M. Nguyen, A. Stage, and V. Georgiev, "Frequency-minimal moving target defense using software-defined networking," in *Proc. of IEEE ICNC*, 2016

[14] T. Carroll, M. Crouse, E. Fulp, and K. Berenhaut, "Analysis of network address shuffling as a moving target defense," in *Proc. of IEEE ICC*, 2014.

[15] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: Transparent moving target defense using software defined networking," in *Proc. of ACM HotSDN*, 2012.

[16] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "An Effective Address Mutation Approach for Disrupting Reconnaissance Attacks," *IEEE Transactions on Information Forensics and Security*, 2015.

[17] A. Clark, K. Sun, and R. Poovendran, "Effectiveness of IP address randomization in decoy-based moving target defense" in *Proc. of IEEE CDC*, 2013.

[18] M. E. Kuhl, J. Kistner, K. Costantini, and M. Sudit, "Cyber attack modeling and simulation for network security analysis" in *Proc. of the 39th Conference on Winter Simulation*, 2007.

[19] S. Debroy, S. De, and M. Chatterjee, "Contention based multichannel MAC protocol for distributed cognitive radio networks," *IEEE Transactions on Mobile Computing*, 2014.

[20] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T)," *Journal of the ACM*, 2006.

[21] N. Bjorner and D. Phan, "newZ: Maximal Satisfaction with Z3," *6th International Symposium on Symbolic Computation in Software Science*, 2014.

[22] S. Shekyan, "slowhttptest Wiki" - *https://github.com/shekyan/slowhttptest/wiki*.