Performance Modeling of Moving Target Defenses*

Warren Connell
Center for Secure Information Systems
George Mason University
Fairfax, VA, USA
wconnel@gmu.edu

Daniel A. Menascé
Department of Computer Science
George Mason University
Fairfax, VA, USA
menasce@gmu.edu

Massimiliano Albanese
Center for Secure Information Systems
George Mason University
Fairfax, VA, USA
malbanes@gmu.edu

ABSTRACT

In recent years, Moving Target Defense (MTD) has emerged as a potential game changer in the security landscape, due to its potential to create asymmetric uncertainty that favors the defender. Many different MTD techniques have then been proposed, each addressing an often very specific set of attack vectors. Despite the huge progress made in this area, there are still some critical gaps with respect to the analysis and quantification of the cost and benefits of deploying MTD techniques. In fact, common metrics to assess the performance of these techniques are still lacking and most of them tend to assess their performance in different and often incompatible ways. This paper addresses these gaps by proposing a quantitative analytic model for assessing the resource availability and performance of MTDs, and a method for the determination of the highest possible reconfiguration rate, and thus smallest probability of attacker's success, that meets performance and stability constraints. Finally, we present an experimental validation of the proposed approach.

CCS CONCEPTS

• **General and reference** → **Performance**; *Metrics*; • **Security and privacy** → **Systems security**; *Domain-specific security and privacy architectures*; • **Computing methodologies** → **Modeling methodologies**; *Model verification and validation*;

KEYWORDS

Moving Target Defense, performance, Markov chains

1 INTRODUCTION

Moving Target Defense (MTD) has the potential of turning the typical asymmetry of the security landscape in favor of the defender [13, 18]. Most of the current MTD technologies are designed to protect systems against a very specific set of attack vectors, such as DoS attacks [19, 20], data exfiltration [28], or SQL injection [8]. Consequently, there is a lack of shared metrics to assess their effectiveness, and, despite the substantial amount of work done in this area, gaps still exist with respect to the analysis and quantification of MTD techniques. A recent attempt at quantification is the work of Maleki *et al.* [23], which uses a discrete-time Markov chain to model the interactions between attacker and defender. Their work, unlike ours, cannot predict the performance of an individual MTD.

When deploying an MTD mechanism, as for any other security mechanism, there exists a trade-off between performance and security [25]. MTDs operate by periodically reconfiguring one or more system parameters. As the reconfiguration frequency increases, MTDs can provide better security, but they increase the overhead on the system and reduce the availability of resources, therefore affecting the overall performance. In particular, the problem of evaluating the impact of MTDs on the availability of resources has not been formally studied. This is an important problem, as resources being reconfigured are temporarily unavailable to legitimate users.

This paper addresses these critical gaps and provides the following major contribution: (i) a quantitative analytic model for assessing the availability and performance of resources that are reconfigured by MTDs; (ii) a method for determining the highest possible reconfiguration rate (corresponding to the smallest probability of attacker's success) that meets performance and stability constraints; and (iii) an experimental validation of the proposed approach.

The rest of the paper is organized as follows. Section 2 provides some background information on MTDs. Section 3 presents a model for quantitative analysis of MTDs, whereas Section 4 describes the experimental testbed used to evaluate our model and to analyze transient behaviors of MTDs. Section 5 presents numerical results obtained with the analytic model and through experimentation. Finally, Section 6 discusses related work, and Section 7 provides some concluding remarks and discusses ongoing and future research.

2 BACKGROUND ON MTDS

Cyber attacks are typically preceded by a reconnaissance phase in which adversaries collect valuable information about the target system, including network topology, service dependencies, and unpatched vulnerabilities. Because most system configurations are static – hosts, networks, software, and services do not reconfigure, adapt, or regenerate except in deterministic ways to support maintenance and uptime requirements – it is only a matter of time for attackers to acquire accurate knowledge about the target system, engineer reliable exploits, and plan their attacks.

*The work of W. Connell and M. Albanese was partially supported by Army Research Office grants W911NF-13-1-0421 and W911NF-13-1-0317, and by the Office of Naval Research grant N00014-13-1-0703. The work of D. Menascé was partially supported by the AFOSR grant FA9550-16-1-0030.

MTD'17, October 30, 2017, Dallas, TX, USA.

ACM ISBN 978-1-4503-5176-8/17/10…$15.00
https://doi.org/10.1145/3140549.3140550

To address this important problem, significant work has been done in the area of Adaptive Cyber Defense (ACD), which includes concepts such as Moving Target Defense (MTD), as well as artificial diversity and bio-inspired defenses, to the extent they involve system adaptation for security and resiliency purposes. MTD techniques are mechanisms for continuously changing or shifting a system's attack surface, thus increasing complexity and cost for the attackers [13]. A system's attack surface has been defined as the *"subset of the system's resources (methods, channels, and data) that can be potentially used by an attacker to launch an attack"* [24]. Therefore, the majority of MTD techniques operate by periodically reconfiguring one or more system parameters in order to disrupt the knowledge an attacker may have acquired about those parameters and, consequently, render the attack's preconditions impossible or unstable. Intuitively, dynamically reconfiguring a system is expected to introduce uncertainty for the attacker and make the reconnaissance effort more costly.

MTDs may be designed to address different stages of the Cyber Kill Chain [15], a framework developed by Lockheed Martin as part of the Intelligence Driven Defense model for identification and prevention of cyber intrusion activity. The model identifies what steps the adversaries must complete in order to achieve their objective: reconnaissance, weaponization, delivery, exploitation, installation, command & control, actions on objectives. Most techniques currently available are designed to address the reconnaissance phase of the cyber kill chain, as they attempt to interfere with the attacker's effort to gather information about the target system.

One of the major drawbacks of many MTD techniques is that they may introduce a costly overhead for legitimate users, as well as the potential for loss of availability. Additionally, most existing techniques are purely proactive in nature or do not adequately consider the attacker's behavior. To address this limitation, alternative approaches aim at inducing a "virtual" or "perceived" attack surface by intelligently crafting responses to an attacker's attempt to gather data: the goal is to induce the attacker to formulate incorrect inferences about the system's configuration [2]. Compared to other approaches, this approach presents the advantage of limiting the overhead by introducing uncertainty for the attackers without actually changing the system's configuration. Honeypots have also been used to divert attackers away from critical resources [1], but they have proven to be less effective than MTDs because they provide a static solution: once a honeypot or honeynet has been discovered, the attacker will avoid it.

3 QUANTITATIVE ANALYSIS OF MTDS

The computing environment we consider in this paper consists of c similar *resources* (e.g., VMs) available to serve incoming service requests that arrive at an average rate λ, join a single queue, and are served by any of the available resources, with an average service time S. A generic MTD technique consists in each resource occasionally, at random intervals, reconfiguring itself independently of the other resources. Thus, each resource handles *service requests* as well as *reconfiguration requests*. While a resource is being reconfigured, it is not available to handle service requests. Without reconfigurations, the system behaves exactly like an M/M/c queue [21].

Now, assume that each resource is reconfigured at an average rate of α. As an example, a reconfiguration could entail swapping out a VM with a clean instance, similar to how SCIT [6] operates. When the VM comes back online, it may have a new IP address – thus implementing a form of IP hopping – or even a new Operating System – thus implementing OS rotation. These reconfigurations make it more difficult for an attacker to learn about the VMs, and disrupt the attacker's persistence in the system. The attacker's success probability depends on the average reconfiguration rate. The reconfiguration rate α also affects the average number of resources available to serve requests (see Figure 1). Reconfigurations reduce resource availability and ultimately increase queuing and response time.

While these *qualitative* tradeoffs are intuitive and not surprising, there is a need for *quantitative* models for determining the impact of the reconfiguration rate on resource availability, response time of service requests, and attacker's success probability. We use Continuous Time Markov Chains (CTMC) to compute the probability distribution of the number of resources being reconfigured as a function of α and other parameters and then use that distribution to determine resource availability and response time, among other metrics. Markov chains have been used for many decades to study various aspects of computer and communication systems. The novelty in each case is in how the state of a CTMC is defined in order to represent the system to be analyzed.

Figure 2, the framework for our analytic models, shows the *reconfiguration model* at the top and, at the bottom, the *performance model*. The reconfiguration model takes as inputs the rate α at which resources are reconfigured, the average reconfiguration time S, and the number of resources c, and produces as outputs the availability of resources, the average number of resources available, and the probability distribution $\{p_k\}$ of the number of available resources. This distribution, along with the number of resources, the average arrival rate of service requests, and the average service time of requests are inputs to the performance model, which produces the probability distribution $\{P_k\}$ of the number of service requests in the system and the average response time of requests.

We analyze this generic MTD in three steps: (i) analysis of the effect of the reconfiguration rate α on the probability distribution of available resources; (ii) analysis of the effect of that availability on

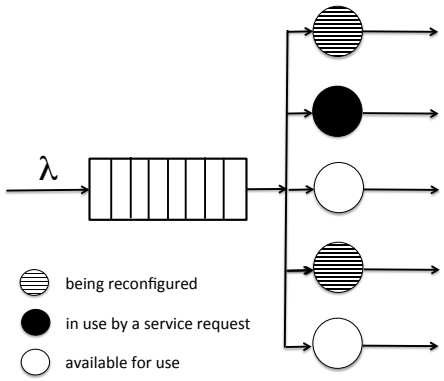

Figure 1: Queuing representation of the reference scenario

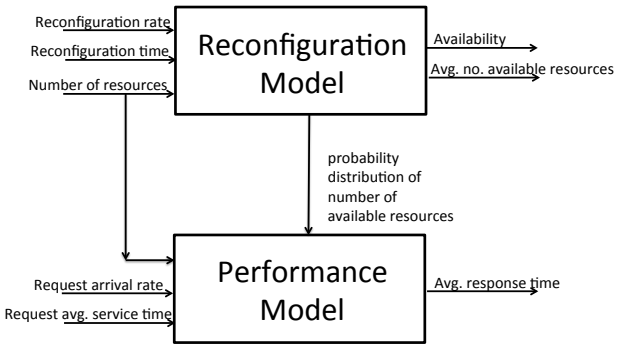

Figure 2: Analytic model framework.

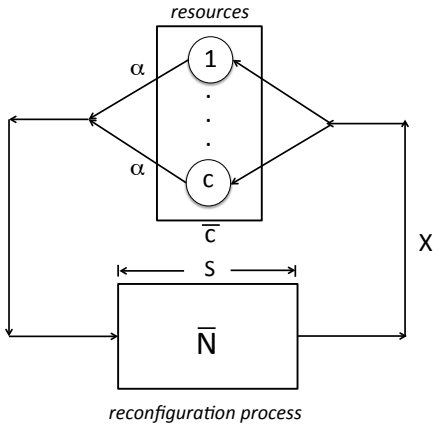

Figure 3: Reconfiguration cycle

response time; and (iii) calculation of the effective reconfiguration rate and determination of the attacker's probability of success.

3.1 Reconfiguration Model

Figure 3 helps explain the basic equations that govern an MTD process. Table 1 summarizes the names and descriptions of all variables defined here.

Consider that there are c resources (e.g., VMs) that are reconfigured at regular time intervals. Each resource cycles through a period in which it is available for use and a period in which it is being reconfigured (see Figure 3). Resources are reconfigured independently of one another at a rate of α reconfigurations per time unit. Thus, the average time a resource is available for use between the end of a reconfiguration operation and the start of the next one is $1/\alpha$. We refer to this as the average *age* of a resource, which is the primary security metric used in determining the likelihood of attacker's success, described later in Section 3.3.

Let $\bar{c}$ be the average number of resources available for use (i.e., not being reconfigured) and $\bar{N}$ be the number of resources being reconfigured. Thus,

$$c = \bar{c} + \bar{N} \tag{1}$$

Applying Little's Law [21] to the set of resources we obtain:

$$\bar{c} = X \times (1/\alpha) \tag{2}$$

where X is the system's reconfiguration throughput (or throughput for short), i.e., the aggregate rate at which resources complete their reconfiguration, which is the collective rate at which resources are reconfigured.

Let S be the average time it takes for a resource to complete the reconfiguration process. For example, a reconfiguration process could include the time to complete all running transactions in progress at a server, changing its configuration file, shutting down the server, and re-booting it. Applying Little's Law [21] to the set of resources being reconfigured, we obtain:

$$\bar{N} = X \times S \tag{3}$$

Adding Eqs. 2 and 3 and combining the result with Eq. 1, we obtain:

$$c = \bar{c} + \bar{N} = X(S + 1/\alpha) \tag{4}$$

We can rewrite Eq. 4 in order to express the reconfiguration rate α as a function of the number of resources c, the time to reconfigure S, and the throughput X:

$$\alpha = X/(c - S \times X) \tag{5}$$

We use the CTMC of Figure 4 to compute X. The state k ($k = 0, \ldots, c$) in this CTMC represents the number of resources in the reconfiguration box of Figure 3. Thus, the number of available resources is $c - k$. The arrival and departure rates are assumed to be stationary (i.e., not changing over time), as they respectively depend on the values of α – which is set by the system administrator – and S – which depends on the nature of service requests. These two parameters may need to be recomputed if the characteristics of incoming service requests change, but they cannot be altered by an attacker capable of adapting in response to the deployment of MTDs.

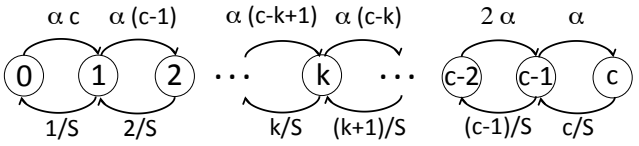

Figure 4: CTMC for the reconfiguration model

An expression for p_k ($k = 0, \ldots, c$) is obtained by using the general birth-death equation for Markov Chains [21]:

$$p_k = p_0 \prod_{i=0}^{k-1} \frac{\gamma_i}{\mu_{i+1}} \qquad k = 1, \ldots, c \tag{6}$$

$$p_0 = \left[1 + \sum_{k=1}^{c} \Pi_{i=0}^{k-1} \frac{\gamma_i}{\mu_{i+1}}\right]^{-1} \tag{7}$$

where $\gamma_k = \alpha \cdot (c - k)$ for $k = 0, \ldots, c-1$ is the aggregate rate at which resources are reconfigured when there are k resources being reconfigured and $\mu_k = k/S$ for $k = 1, \ldots, c$ is the aggregate rate at which resources complete their reconfiguration when there are k resources being reconfigured. Using the expressions for γ_k and μ_k

Table 1: Summary of variable names and descriptions

Variable	Description
$P_s(t)$	Probability that t time units are needed for a successful attack
T_s	Time needed for an attacker to succeed: $P_s(T_s) = 1$
c	Number of resources
$\bar{c}$	Average number of resources not being reconfigured
$\bar{N}$	Average number of resources being reconfigured
α	Reconfiguration rate (in reconfigurations per time unit)
S	Average time to reconfigure a resource
X	Reconfiguration throughput, i.e., aggregate rate at which resources start a reconfiguration
λ	Average arrival rate of service requests
T	Average time a request spends using a resource
R	Average response time of requests

in Eqs. 6 and 7, we obtain:

$$p_k = p_0 \prod_{i=0}^{k-1} \frac{\alpha \cdot (c-i)}{(i+1)/S} = p_0 \cdot (\alpha \cdot S)^k \binom{c}{k} \qquad k = 1, \ldots, c \quad (8)$$

An expression for p_0 is obtained by noting that the sum of all probabilities is equal to 1. Thus,

$$p_0 = \left[1 + \sum_{k=1}^{c} (\alpha \cdot S)^k \binom{c}{k} \right]^{-1} \quad (9)$$

The values of p_k can be easily computed because the summation needed to compute p_0 is finite. Given p_k and p_0 one can then compute the average throughput X as

$$X = \sum_{k=1}^{c} (k/S) \cdot p_k = \frac{1}{S} \sum_{k=1}^{c} k \cdot p_k \quad (10)$$

The average number of available resources can now be computed by combining Eqs. 2 and 10:

$$\bar{c} = \frac{X}{\alpha} = \frac{1}{\alpha \cdot S} \sum_{k=1}^{c} k \cdot p_k \quad (11)$$

The *availability* A of the set of resources is then given by the fraction of resources available for use, i.e.,

$$A = \frac{\bar{c}}{c} = \frac{X}{\alpha \cdot c} \quad (12)$$

It turns out that the availability does not depend on the number of resources, but only on the product of the reconfiguration rate and the reconfiguration time. This can be seen by combining Eqs. 2, 4, and 12.

$$A = \frac{\bar{c}}{c} = \frac{X}{\alpha} \cdot \frac{1}{c} = \frac{1}{1 + \alpha \cdot S} \quad (13)$$

When there is no reconfiguration (i.e., $\alpha = 0$), the availability is 1 as expected. Eq. 13 can be used to determine the value of the product $\alpha \cdot S$ necessary to guarantee an availability greater than or equal to some value $A_{\min}$.

$$\alpha \cdot S \leq \frac{1}{A_{\min}} - 1 \quad (14)$$

3.2 Response time model

The c resources are used for some computational purpose and requests to use any of them arrive at a rate of λ requests per unit of time and are served by any of the available resources. If no resources are available, a request has to wait in a queue. The number of resources available for service varies from 0 to c due to reconfiguration (see Figure 1). The probability that k resources are available for service is given by the probability p_{c-k} that $c - k$ resources are being reconfigured (see Eq. 8).

We use the CTMC of Figure 5 with an infinite number of states, where a state $k = 0, 1, 2, \ldots$ represents the number of service requests in the system, either using one of the available resources or waiting for one. Note that the system of Figure 1 is similar to an M/M/c queuing system with an important difference. In an M/M/c model, the rate at which transactions complete is $k\mu$ for $k = 1, \ldots, c$ and $c\mu$ for $k > c$ where μ is the average rate at which a request completes from a resource. In our case, as explained above, the transaction completion rate has to consider that resources may be in the process of being reconfigured. Thus, we follow an approach similar to the development of the results for the M/M/c queue [21], with a modification in the average transaction completion rate. Consider the following additional notation:

- P_k: probability that there are k requests in the system (either being serviced or in the waiting line).
- π_j: probability that j resources are available for use (i.e., not being reconfigured), thus $\pi_j = p_{c-j}$.
- $\mu\delta_k$: average request departure rate at state k. The value of δ_k is

$$\delta_k = \sum_{j=1}^{k} j \, \pi_j \qquad k = 1, \ldots, c \quad (15)$$

  because the departure rate is μ if only one resource is available (which happens with probability π_1), 2μ if only two resources are available (which happens with probability π_2), ..., and $k\mu$ if k resources are available (which happens with probability π_k). Note that $\delta_c = \sum_{j=1}^{c} j \, \pi_j = \bar{c}$.
- $\mu = 1/T$: average service rate of each resource.
- $\rho = \lambda/(\mu \, \bar{c})$: average utilization of the resources.

As Figure 5 shows, the transition rate from state k to $k + 1$ is λ, the average request arrival rate, and the transition rate β_k from a

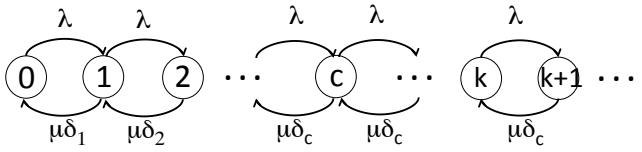

Figure 5: CTMC for the response time model

state k to state $k-1$ is given by

$$\beta_k = \begin{cases} \mu\ \delta_k & k < c \\ \mu\ \delta_c & k \geq c \end{cases} \tag{16}$$

We can now use the generalized birth-death equations (see Eqs. 6 and 7) to solve for P_k and P_0. We have to break down the expression for P_k into two parts (for $k = 1, \ldots, c$ and $k > c$) because β_k has two expressions. Hence,

$$P_k = P_0 \prod_{i=0}^{k-1} \frac{\lambda}{\mu\ \delta_{i+1}} = P_0 \frac{\left(\frac{\lambda}{\mu}\right)^k}{\prod_{i=0}^{k-1} \delta_{i+1}} \qquad k = 0, \ldots, c \tag{17}$$

and

$$P_k = P_0 \prod_{i=0}^{c-1} \frac{\lambda}{\mu\ \delta_{i+1}} \prod_{i=c}^{k-1} \frac{\lambda}{\mu\ \delta_c} = P_0 \frac{\rho^k\ \bar{c}^c}{\prod_{i=0}^{c-1} \delta_{i+1}} \qquad k = c+1, \ldots \tag{18}$$

P_0 can now be computed as:

$$P_0 = \left[1 + \sum_{k=1}^{c} \frac{\left(\frac{\lambda}{\mu}\right)^k}{\prod_{i=0}^{k-1} \delta_{i+1}} + \sum_{k=c+1}^{\infty} \frac{\rho^k\ \bar{c}^c}{\prod_{i=0}^{c-1} \delta_{i+1}} \right]^{-1} \tag{19}$$

If we move $\bar{c}^c / \prod_{i=0}^{c-1} \delta_{i+1}$ out of the infinite summation in the above expression, we are left with the following geometric series, which converges for $\rho < 1$:

$$\sum_{k=c+1}^{\infty} \rho^k = \frac{\rho^{c+1}}{1-\rho} \tag{20}$$

Note that $\rho < 1$ is a *necessary* but not sufficient condition for the system to be stable as discussed is Section 5.2. P_0 can now be written as the following easily computable expression:

$$P_0 = \left[1 + \sum_{k=1}^{c} \frac{\left(\frac{\lambda}{\mu}\right)^k}{\prod_{i=0}^{k-1} \delta_{i+1}} + \frac{\bar{c}^c}{\prod_{i=0}^{c-1} \delta_{i+1}} \frac{\rho^{c+1}}{1-\rho} \right]^{-1} \tag{21}$$

The average number of requests in the system can be computed as

$$N_s = \sum_{k=1}^{\infty} k \cdot P_k = \sum_{k=1}^{c} k \cdot P_k + \sum_{k=c+1}^{\infty} k \cdot P_k \tag{22}$$

The infinite summation in Eq. 22 can be written as

$$P_0 \frac{\bar{c}^c}{\prod_{i=0}^{c-1} \delta_{i+1}} \left[\sum_{k=c+1}^{\infty} k\ \rho^k \right] = P_0 \frac{\bar{c}^c}{\prod_{i=0}^{c-1} \delta_{i+1}} \left[\rho \frac{\partial}{\partial \rho} \sum_{k=c+1}^{\infty} \rho^k \right]$$

and is equal to

$$P_0 \frac{\bar{c}^c}{\prod_{i=0}^{c-1} \delta_{i+1}} \frac{\rho^{c+1}}{1-\rho} \left[\frac{\rho}{1-\rho} + 1 + c \right] \tag{23}$$

Thus, Eqs. 22 and 23 allow to compute N_s. Finally, using Little's Law we compute the average response time R as $R = N_s/\lambda$.

3.3 Analysis of Attack Success Probability

Estimating the time required for an attacker to gather sufficient knowledge during the reconnaissance phase is critical to assess the attacker's ability to successfully compromise a system. As our focus is on disrupting an attacker's reconnaissance effort, we can – without loss of generality – define the probability that an attacker succeeds as the probability to gather sufficient information to plan and execute an attack, which in turn is a function of the time available to complete the reconnaissance phase. In other words, we are implicitly assuming that, once accurate information is available to the attacker, the attack will always be successful. The probability $P_s(t)$ that an attacker succeeds in t time units is important in determining the required *reconfiguration rate*, i.e., the rate at which resources needs to be reconfigured.

Figure 6 shows two examples of $P_s(t)$, namely, linear and exponential functions. The linear function, $P_s(t) = t/T_s$, indicates that the probability of attack success increases linearly with time and reaches 1 (i.e., success) at time T_s. The exponential function (see for instance Eq. 24) indicates a situation in which the attacker initially accumulates knowledge at a low rate and then becomes exponentially more knowledgeable over time and succeeds at time T_s.

$$P_s(t) = 1 - \frac{1 - e^{(t-T_s)}}{1 - e^{-T_s}}. \tag{24}$$

As an example, consider an IP sweep combined with a port scan, where the attacker's goal is to discover the IP address of the machine running a specific service within target network. The attack consists in sequentially scanning all IP addresses in a given range. Assuming an IP space of n addresses and that t^* time units are required to scan a single IP, we obtain $T_s = n \cdot t^*$ and $P_s(t) = \frac{t}{T_s} = \frac{t}{n \cdot t^*}$. As another example, consider the following DoS attack. The attacker initially compromises n hosts, which takes t^* time units. Then, each of the newly compromised hosts compromises additional n hosts, which takes additional t^* time units. At any given time t, the total number of compromised hosts, including the attacker's machine, is $N(t) = 1 + n + n^2 + \ldots + n^k = \frac{1-n^{k+1}}{1-n}$, where $k = \lfloor t/t^* \rfloor$. We

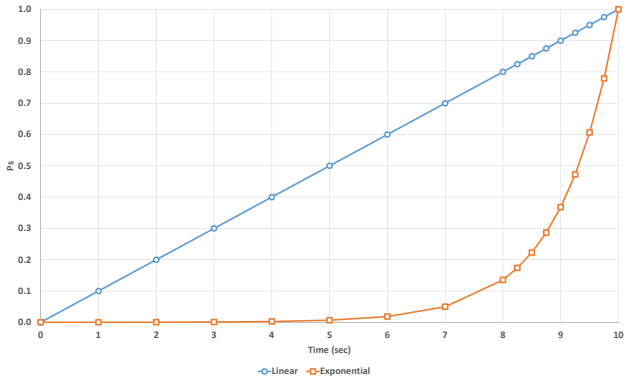

Figure 6: Probability of success P_s vs. time for $T_s = 10$

can assume that the attacker's success probability is proportional to the aggregate amount of flood traffic that compromised hosts can send to the victim, compared to the victim's capacity to handle incoming traffic. Let V denote the volume of traffic the victim can handle per time unit and let v denote the amount of traffic each compromised node can send per time unit. Then,

$$P_s(t) = \min\left\{1, \frac{N(t)\cdot v}{V}\right\} = \min\left\{1, \frac{v}{V}\cdot\frac{1-n^{\lfloor t/t^*\rfloor+1}}{1-n}\right\}$$

4 SIMULATION AND EXPERIMENTAL TESTBED

Our analytical results were validated by simulation and by experiments that implemented a generic MTD. We implemented the simulation using SimPy[1], a process-based discrete-event simulation framework based on standard Python. SimPy supports multiple processes that contend for access to a resource and automatically handles queuing of events if a resource is busy, making it ideal for our purposes. Additionally, we used SimPy as a real-time event generator to control VM reconfigurations and implement a fully operational MTD. For our VM environment, we used Citrix's open-source XenServer platform[2], which offers pooling of resources, the ability to quickly clone VMs for reconfiguration, and a command-line interface that is compatible with our simulation framework.

Our MTD controller runs on a separate server and starts an independent process for each VM – either a simulated VM or an actual VM in the XenServer pool – that generates reconfiguration requests. Our experiments show that S is normally distributed, so we used a normal distribution for S with the same mean and standard deviation that were observed in the experiments.

Reconfigurations may consist of a number of possible actions, including for instance changing the IP address or software. In our experiments, we remove a VM instance from the virtual network and replace it with a fresh copy, similar to how SCIT operates [6]. The fresh copy also has a new IP address obtained from DHCP, enabling a basic IP-hopping scheme. The reconfiguration process also collects statistics such as percentage of requests dropped and possible delays.

The MTD controller also serves as a traffic generator that creates service requests. Each service request is an independent process with exponentially distributed interarrival times with an average arrival rate equal to λ and average service time T in the simulations. For the experiments, an HTTP request is sent to an idle VM, which has a scripted delay on the HTTP response with average time T to simulate the time to process a generic service request. Each process records the time at which it was generated, began service, and completed, according to the environment's internal clock. These records are used to compute queue time, service time, and response time, and are maintained for each request.

We also collect statistics from a separate monitor process that operates at set intervals to gather information about the number of resources idle, in use, and being reconfigured, as well as the current queue length. An overview of the system and processes is shown in Figure 7.

[1]Available at https://simpy.readthedocs.io/en/3.0/.
[2]Available at https://xenserver.org/

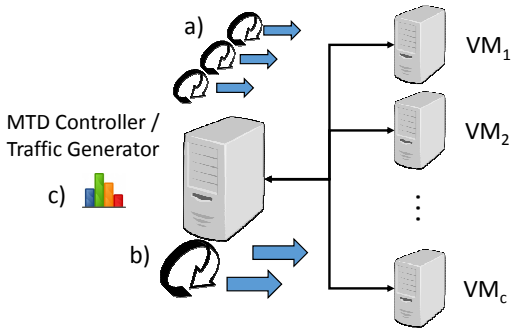

Figure 7: Experimental setup: a) c independent processes to generate reconfiguration requests (arrival rate α), b) 1 process to generate independent service requests (arrival rate λ), c) Monitor process (every 0.01 sec)

The pool of VMs is tracked using three separate states for the VMs: *idle*, *in use* (i.e., serving a request), or *being reconfigured*. All requests for a VM must first acquire a shared resource that gives them access to the pool of idle VMs. A priority queue is used, giving priority to reconfiguration requests so that reconfiguration is not unnecessarily delayed; however, reconfiguration requests for a specific VM will not preempt a request currently being served. Instead, the reconfiguration request flags that VM for reconfiguration and then releases its lock on the idle pool before waiting for that resource to appear in the pool of VMs to reconfigure. When service requests receive access to the idle pool, they remove a random VM from the pool and place it in the pool of VMs in use. Once completed the request, if that VM is flagged for reconfiguration, it is placed in the reconfiguration pool where the reconfiguration request will pick it up for reconfiguration, otherwise it is placed back in the idle pool. In the event that a service request finds no VMs in the idle pool, it waits for one to appear. This additional wait is included in the overall queue time. The overall flow of control and VM state transitions is shown in Figure 8.

Each iteration of the simulation lasted 6,000 seconds, with no statistics recorded in the first 1,000 seconds to allow the system to

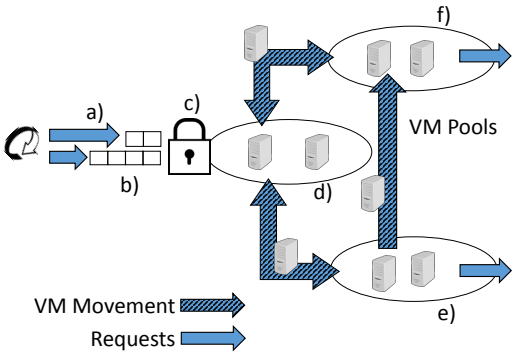

Figure 8: Control Flow and VM Movement: a) incoming requests, b) priority queue, c) resource lock on idle pool, d) idle VM pool, e) VMs in use, f) reconfiguration VMs

achieve steady state. Thirty runs were performed for values of α from 0.001 to 0.050 to obtain the mean, standard deviation, and 95% confidence intervals for the mean for each statistic. For the experiments, each run is limited to 600 seconds with statistics recorded after the first 60 seconds for select values of α. The values of the other input parameters used in the simulations and experiments are given in Table 2.

Table 2: Values of variables used in the numerical results

Variable	Description
T_s	300 sec
c	20
α	from 0.001 to 0.050 rec/sec
S	120 sec
λ	10 requests/sec
T	0.5 sec

5 NUMERICAL RESULTS AND VALIDATION

This section presents several numerical results starting with those obtained from the reconfiguration model along with validation of the model. We then cover the performance model, including some interesting findings from our implementation of the model. Finally, we show how the 2 models can be used to find an optimal value of the reconfiguration rate that considers tradeoffs between response time and security.

5.1 Reconfiguration Model

Figure 9 shows the distribution of the number k of resources being reconfigured for four values of the reconfiguration rate α, out of a total of 20 resources. The graphs show that as α increases from 0.005 to 0.04 rec/sec, the probability distribution moves to the right. The average number of resources being reconfigured is 7.50 for $\alpha = 0.005$ rec/sec, going up to 16.55 for $\alpha = 0.04$ rec/sec.

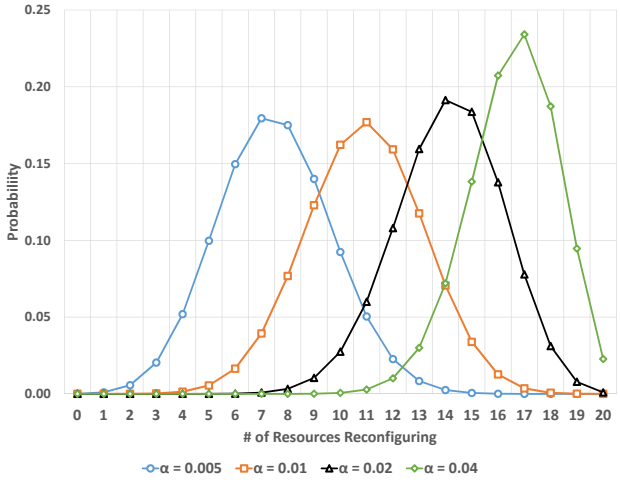

Figure 9: Distribution of the number of resources being reconfigured for $c = 20$.

The reconfiguration probabilities p_k are used to compute the availability. Figure 10 shows three availability curves as a function of the reconfiguration rate α for values of the reconfiguration time S equal to 60, 90, and 120 seconds, respectively. As the reconfiguration rate increases, the availability decreases in a non-linear fashion and, as the reconfiguration time increases, the availability decreases for the same value of α. As the reconfiguration rate tends to zero, the availability tends to 1 because all resources are available for use. Note that, as we indicated in Section 3, the availability does not depend on the number of resources c, as it is a relative measure.

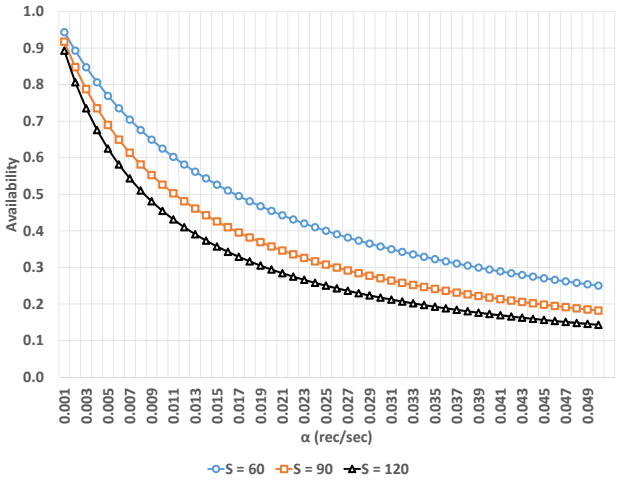

Figure 10: Availability vs. reconfiguration rate α

We validated the analytic model using the simulation and experiments, using the mean and standard deviation of the reconfiguration times measured in the experiments. We found that the probability distribution generated by the simulation closely matches that of the analytical model, as seen in Figure 11.

The theoretical availability results match very well the results obtained through the simulations and experiments as shown in Table 3, which indicates that, for the same range of values of α used in Figure 10, the percentage absolute relative error between the model and the simulation does not exceed 2.29% and the error between the model and the experimental does not exceed 9.62%.

Table 3: Comparison of availability results.

α	Model Results	Simulation Results ± 1/2 of 95% CI	Absolute % Error	Experimental Results ± 1/2 of 95% CI	Absolute % Error
0.005	0.696	0.694 ± 0.004	0.29	0.686 ± 0.015	1.46
0.010	0.467	0.466 ± 0.004	0.21	0.465 ± 0.017	0.43
0.015	0.329	0.330 ± 0.003	0.30	0.318 ± 0.017	3.46
0.020	0.253	0.255 ± 0.003	0.78	0.247 ± 0.011	2.43
0.030	0.171	0.175 ± 0.002	2.29	0.156 ± 0.007	9.62
0.040	0.132	0.133 ± 0.002	0.75	0.121 ± 0.007	9.09

5.2 Response Time Model

We now summarize our results about the response time and then explain them in detail. The key conclusions are that: (i) the response

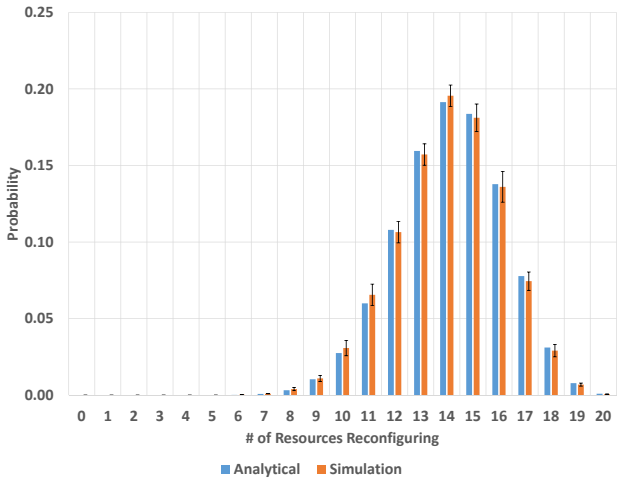

Figure 11: Comparison of the distributions of the number of resources being reconfigured (α = 0.02 rec/sec).

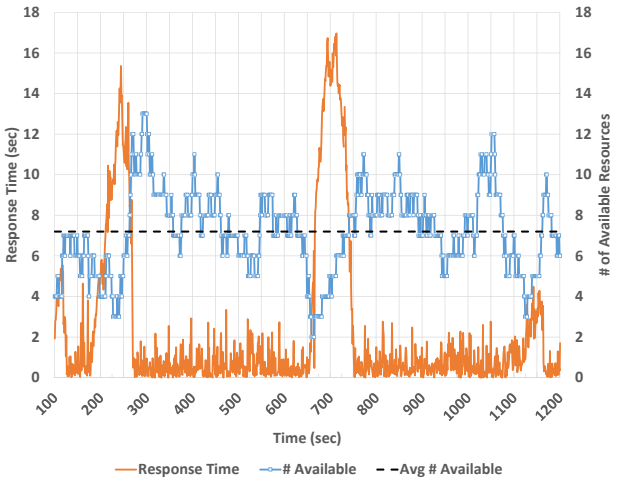

Figure 13: Number of available resources and response time (α = 0.015)

time model closely matches the simulation results for a range of values of α for which the system is stable most of the time (we formally define stability later); and (ii) for larger values of α there is a high variation of the utilization around its average $\rho = \lambda\, T/\bar{c}$, which causes the system to become unstable (i.e., $\rho > 1$) for non-negligible fractions of time: as a consequence, the queue and the response time grow infinitely.

As indicated in Section 3, ρ must be less than 1. As ρ tends to 1, the response time tends to infinity because of the term $(1-\rho)$ that appears in the denominator of the expression of the average number of requests in the system. Because λ and T are assumed constant (λ = 10 requests/sec and T = 0.5 sec), the variation of ρ depends on $\bar{c}$, which decreases with the availability, which in turn decreases as α increases (see Table 3). Thus, $\rho < 1 \Rightarrow \bar{c} > \lambda T \Rightarrow \bar{c} > 5$.

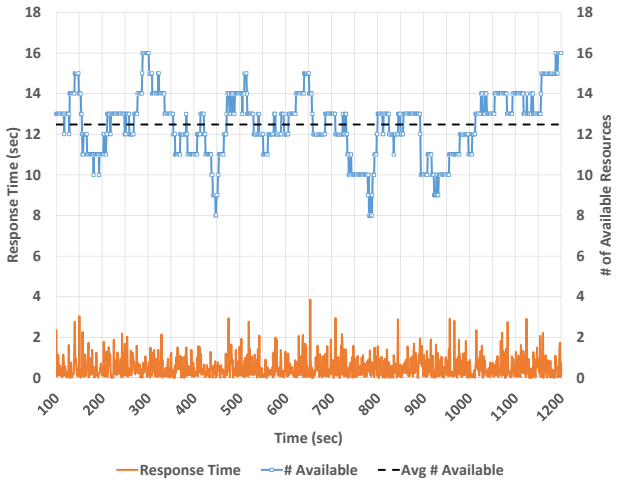

Figure 12: Number of available resources and response time (α = 0.005)

Before we present the variation of the response time as a function of α, it is instructive to compare graphs of run-time data captured for α = 0.005 and α = 0.015 in Figures 12 and 13 respectively. From this data, we observe a higher coefficient of variance (COV) for the number of available resources and for the response time. For α = 0.005, the COV for available resources is 0.123 and for response time is 1.007; but for α = 0.015, these values go up to 0.287 and 1.676, respectively. More importantly, we also notice that in both cases, $\bar{c} > 5$ as denoted by the dashed line in the graphs, thus $\rho < 1$. However, for α = 0.015, there are periods of time where there are 5 or fewer resources available, causing a spike in response time. During these periods, $\rho \geq 1$ and the queue of service requests grows infinitely. Furthermore, even as the number of available resources returns above the minimum required, there is a lagging effect on response times returning to normal as there are built-up service requests in the queue. Thus, a metric such as ρ alone does not capture well the effect of episodic instability. To better quantify this effect, we introduce a metric ω, which we call *stability*, defined as the fraction of time the system is in a stable state (i.e., $\rho < 1$):

$$\omega = \sum_{k \in \mathcal{N}} p_k \tag{25}$$

where $\mathcal{N} = \{k \in \{0, 1, \ldots, c\} \wedge \lambda T/(c-k) < 1\}$. Because ω depends on the probabilities p_k, it is a function of α, and we use $\omega(\alpha)$ to denote that relationship. Then, a system is stable for a given set of parameters if $\omega(\alpha) \approx 1$ because it is almost never in a situation where $\rho > 1$. Algorithm 1 is used to compute $\omega(\alpha)$.

Figure 14 shows ω superimposed over response time results obtained through simulation and from the analytic model. As we can see, for low values of α, ω is very close to 1 and the simulation matches the analytic results. As α increases, we observe that the response time is very sensitive to small decreases in stability. When α = 0.015 rec/sec, ω = 0.775, which means that the system is unstable 22.5% of the time with requests rapidly building up in the queue, causing a higher than expected value and variance in the response time. A possible solution is to limit the number of resources

Algorithm 1 Computation of $\omega(\alpha)$

```
Input: α, S, T, λ, c, {p_k}
ω ← 0
for k = 0 → c do
  if λT/(c − k) < 1 then
    ω ← ω + p_k
  end if
end for
return ω;
```

reconfiguring at any one time to ensure that there are sufficient resources available to handle the expected workload, which is an area we plan to investigate as part of our future work.

5.3 Optimal Reconfiguration Rate

The model presented in Section 3 allows one to answer a variety of "what-if" questions such as *"How does the resource availability vary with the time needed to reconfigure a resource?"* or *"How does the average response time of service requests vary with the average reconfiguration rate?"* Additionally, one can solve optimization problems such as maximizing the reconfiguration rate subject to the following constraints: (i) the stability must be greater than or equal to a threshold $\omega_{\min}$, and (ii) the average response time must be less than or equal to a threshold $R_{\max}$. More formally,

Maximize α
s.t. $\omega \geq \omega_{\min}$ and $R \leq R_{\max}$

Because the stability decreases monotonically with α and the response time R increases monotonically with α (see Figure 14), the maximum feasible value $\alpha_{\max}$ of α is

$$\alpha_{\max} = \min(\alpha_\omega, \alpha_R) \tag{26}$$

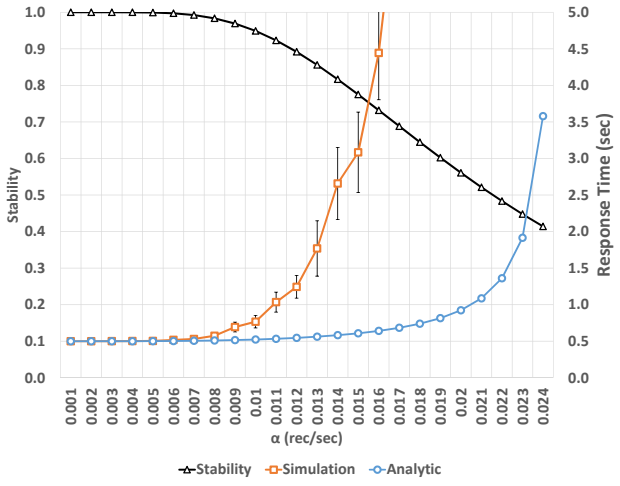

Figure 14: Response time: Simulation vs. Analytical Model with Stability

where

$$\begin{aligned} \alpha_\omega &= \text{argmax}_\alpha\{\omega \geq \omega_{\min}\} \\ \alpha_R &= \text{argmax}_\alpha\{R \leq R_{\max}\} \end{aligned} \tag{27}$$

Consider Figure 15 and $S = 60$ sec, $c = 20$, $\omega_{\min} = 0.9$, and $R_{\max} = 0.75$ sec. Then, $\alpha_\omega \leq 0.023$ must hold to satisfy the stability constraint. However, in order to satisfy the response time constraint, $\alpha_R \leq 0.036$ as illustrated in Figure 15. Therefore, $\alpha \leq \min(0.023, 0.036) = 0.023$ rec/sec. This means that each resource will be available, on average, for $1/\alpha = 1/0.023 = 43.5$ seconds before it is reconfigured.

We now consider the interplay of the maximum reconfiguration rate $\alpha = 0.023$ rec/sec obtained in the optimization example above and the probability of a successful attack. Consider a linear function for the probability $P_s(t)$ with $T_s = 100$ sec. In that case, the probability that an attacker succeeds after 43.5 sec is $43.5/100 = 0.435$. However, if the knowledge accumulation rate has the exponential form of Eq. 24, the probability of a successful attack at $t = 43.5$ is

$$P_s(43.5) = 1 - \frac{1 - e^{(43.5-100)}}{1 - e^{-100}} \approx 0 \tag{28}$$

In other words, the optimal reconfiguration rate yields a relative large probability that the attacker is successful if knowledge can be accumulated linearly but a close-to-zero probability of success when knowledge is accumulated at very slow pace initially.

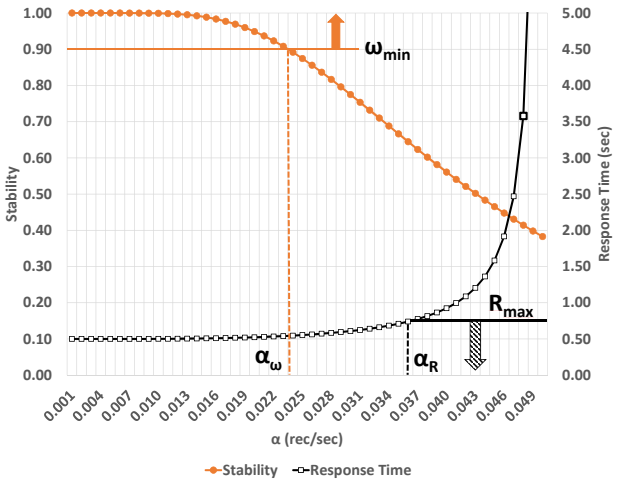

Figure 15: Optimization analysis to find the maximum feasible reconfiguration rate α for $c = 20$ and $S = 60$ sec.

6 RELATED WORK

Several MTD techniques have been proposed in recent years, with the majority of such techniques designed to protect systems against a very specific set of attack vectors such as SQL injection [8], data exfiltration [28], or distributed DoS attacks [19, 20]. A rich line of research has focused on MTD techniques to mitigate distributed DoS attacks [19, 20] by deploying proxies between clients and servers, and periodically reshuffling – either proactively or in response to

detected threats – the associations between clients and proxies in order to disrupt knowledge accumulated by adversaries.

Dunlop *et al.* [12] proposed a mechanism to dynamically hide addresses of IPv6 packets in order to achieve anonymity. In [11], Duan *et al.* presented a proactive Random Route Mutation technique to randomly change the route of network flows to defend against eavesdropping and DoS attacks. They use OpenFlow Switches and a centralized controller to define the route of each flow. Jafarian *et al.* [16] used an IP virtualization scheme based on virtual DNS entries and Software Defined Networks to hide network assets from scanners. Using OpenFlow, each host is associated with a range of virtual IP addresses and mutates its IP address within its pool. A similar identity virtualization approach was presented in [3]. An approach based on diverse virtual servers was proposed in [18] to create a dynamic and uncertain attack surface. Each server is configured with a set of software stacks, and a rotational scheme is employed for substituting different software stacks for any given request.

Deception-based approaches aim at misleading reconnaissance tools, such as *nmap* or *Xprobe2*. Such tools are able to identify a service or an operating system by analyzing packets that can reveal implementation specific details about the host [22]. Watson *et al.* [29] adopted protocol scrubbers in order to avoid revealing implementation-specific information and restrict an attacker's ability to determine the operating system of a protected host. Moreover, some proof-of-concept software and kernel patches have been proposed to alter a system fingerprint [7], such as IP Personality and Stealth Patch.

Different metrics have been proposed to quantify the effectiveness of MTDs. Some authors assess performance in terms of the attacker's success rate [10], while others also consider metrics for deterrence and detectability [17], or multiple metrics (productivity, success, confidentiality, and integrity) for the attacker and defender [30]. Previously described metrics do not evaluate more than a few select MTDs chosen for a specific study. One expert survey provides a thorough assessment of the effectiveness and cost of many techniques across the spectrum of existing MTDs [14]. However, the survey is qualitative in nature and potentially subject to reviewer bias. In [9], Cai *et al.* develop a performance evaluation and comparison model for existing MTDs based on stochastic Petri Nets. Although more general than most existing approaches, this model still has some limitations, as the authors focus on MTD techniques that can be deployed on a web server, whereas the model we propose in this paper is agnostic of the specific nature of the hosts being defended and the MTDs being deployed.

The model presented in [23] is the closest to our work and introduces a Markov-model-based framework for MTD analysis. The framework allows modeling of a broad range of MTD strategies, and presents results on how the probability of an adversary defeating an MTD strategy is related to the time/cost spent by the adversary. They also show how their approach can be used to analyze a composition of MTDs. However, differently from our model, the proposed approach does not evaluate the performance of deployed MTD techniques, which the authors list as future work. Another relevant body of work in the area of quantification is represented by efforts addressing the availability, performance and security of intrusion-tolerant systems [26, 27]. However, the scope of quantification of MTD techniques is significantly different from quantification in intrusion-tolerant systems, since MTD techniques aim at preventing, rather than tolerating, intrusions.

7 CONCLUSIONS

Moving Target Defense has been recognized as a potential game-changer in cyber security, prompting the development of a wide array of MTD techniques. Unfortunately, most such techniques address a very specific set of attack vectors, and critical gaps still exist with respect to the problem of quantifying their performance and cost. This paper represents a first important step toward addressing these gaps. In summary, we have proposed a quantitative analytic model for assessing the resource availability and performance of MTDs, and a method for the determination of the optimal reconfiguration rate that minimizes the attack success probability subject to availability and performance constraints. The results obtained so far are encouraging and indicate this is a promising research direction. As part of our ongoing future work, we are studying reconfiguration strategies that may allow users to control the trade-off between availability and security. As mentioned earlier, a possible solution to address the instability issue, is to limit the number of resources reconfiguring at any one time to ensure that there are sufficient resources available to handle the expected workload. If a reconfiguration request arrives at a time when the maximum numbers of resources that can be reconfiguring at the same time has already been reached, it has to either be dropped or queued. A similar mechanism will prevent attacks on availability, where the attacker's goal is to trigger an excessive number of reconfigurations in order to make the systems unstable and unresponsive.

Our work is also inspired by research on autonomous systems [4, 5], which change their settings to adapt to the environment. When applied to security mechanisms, this concept can be seen as a form of moving target defense (MTD). To change their settings effectively, self-protecting systems need a way to quantify both their effectiveness and cost or overhead in order to provide an accurate measure of their utility. Therefore, our work on MTD quantification is critical for enabling similar self-protection capabilities.

REFERENCES

[1] Fahim H. Abbasi, Richard J. Harris, Giovanni Moretti, Aun Haider, and Nafees Anwar. 2012. Classification of malicious network streams using Honeynets. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM 2012)*. IEEE, Anaheim, CA, USA, 891–897.

[2] Massimiliano Albanese, Ermanno Battista, Sushil Jajodia, and Valentina Casola. 2014. Manipulating the Attacker's View of a System's Attack Surface. In *Proceedings of the IEEE Conference on Communications and Network Security (CNS 2014)*. IEEE, San Francisco, CA, USA, 472–480.

[3] Massimiliano Albanese, Alessandra De Benedictis, Sushil Jajodia, and Kun Sun. 2013. A Moving Target Defense Mechanism for MANETs Based on Identity Virtualization. In *Proceedings of the IEEE Conference on Communications and Network Security (IEEE CNS 2013)*. IEEE, Washington, DC, USA, 278–286.

[4] Mourad Alia, Marc Lacoste, Ruan He, and Frank Eliassen. 2010. Putting Together QoS and Security in Autonomic Pervasive Systems. In *Proceedings of the 6th ACM Workshop on QoS and Security for Wireless and Mobile Networks (Q2SWinet 2010)*. ACM, Bodrum, Turkey, 19–28.

[5] Firas Alomari and Daniel A. Menascé. 2012. An Autonomic Framework for Integrating Security and Quality of Service Support in Databases. In *Proceedings of the 6th International Conference on Software Security and Reliability (SERE 2012)*. Gaithersburg, MD, USA, 51–60.

[6] Anantha K. Bangalore and Arun K. Sood. 2009. Securing Web Servers Using Self Cleansing Intrusion Tolerance (SCIT). In *Proceedings of the 2nd International Conference on Dependability (DEPEND 2009)*. IEEE, Athens/Glyfada, Greece, 60–65.

[7] David Barroso Berrueta. 2003. A practical approach for defeating Nmap OS-Fingerprinting. (2003). http://nmap.org/misc/defeat-nmap-osdetect.html

[8] Stephen W. Boyd and Angelos D. Keromytis. 2004. SQLrand: Preventing SQL Injection Attacks. In *Proceedings of the 2nd International Conference on Applied Cryptography and Network Security (ACNS 2004) (Lecture Notes in Computer Science)*, Markus Jakobsson, Moti Yung, and Jianying Zhou (Eds.), Vol. 3089. Springer, Yellow Mountain, China, 292–302.

[9] Guilin Cai, Baosheng Wang, Yuebin Luo, and Wei Hu. 2016. A Model for Evaluating and Comparing Moving Target Defense Techniques Based on Generalized Stochastic Petri Net. In *Proceedings of the 11th Conference on Advanced Computer Architecture (ACA 2016)*. Springer, Weihai, China, 184–197.

[10] Thomas E. Carroll, Michael Crouse, Errin W. Fulp, and Kenneth S. Berenhaut. 2014. Analysis of Network Address Shuffling as a Moving Target Defense. In *IEEE International Conference on Communications (ICC 2014)*. IEEE, Sydney, Australia, 701–706.

[11] Qi Duan, Ehab Al-Shaer, and Haadi Jafarian. 2013. Efficient Random Route Mutation Considering Flow and Network Constraints. In *Proceedings of the IEEE Conference on Communications and Network Security (IEEE CNS 2013)*. IEEE, Washington, DC, USA, 260–268.

[12] Matthew Dunlop, Stephen Groat, Randy Marchany, and Joseph Tront. 2012. Implementing an IPv6 Moving Target Defense on a Live Network. In *Proceedings of the National Moving Target Research Symposium*. Annapolis, MD, USA.

[13] Executive Office of the President, National Science and Technology Council. 2011. Trustworthy Cyberspace: Strategic Plan for the Federal Cybersecurity Research and Development Program. http://www.whitehouse.gov/. (December 2011).

[14] Katheryn A. Farris and George Cybenko. 2015. Quantification of Moving Target Cyber Defenses. In *Proceedings of SPIE Defense + Security 2015*. Baltimore, MD, USA.

[15] Eric M. Hutchins, Michael J. Cloppert, and Rohan M. Amin. 2010. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. Lockheed Martin Corporation. (2010).

[16] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. 2012. OpenFlow Random Host Mutation: Transparent Moving Target Defense using Software Defined Networking. In *Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks (HotSDN 2012)*. ACM, Helsinki, Finland, 127–132.

[17] Jafar Haadi Jafarian and Ehab Al-Shaer amd Qi Duan. 2014. Spatio-temporal Address Mutation for Proactive Cyber Agility Against Sophisticated Attackers. In *Proceedings of the 1st ACM Workshop on Moving Target Defense (MTD 2014)*. ACM, Scottsdale, AZ, USA, 69–78.

[18] Sushil Jajodia, Anup K. Ghosh, Vipin Swarup, Cliff Wang, and Xiaoyang Sean Wang (Eds.). 2011. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats* (1st ed.). Advances in Information Security, Vol. 54. Springer.

[19] Quan Jia, Kun Sun, and Angelos Stavrou. 2013. MOTAG: Moving Target Defense Against Internet Denial of Service Attacks. In *Proceedings of the 22nd International Conference on Computer Communications and Networks (ICCCN 2013)*. Nassau, Bahamas.

[20] Quan Jia, Huangxin Wang, Dan Fleck, Fei Li, Angelos Stavrou, and Walter Powell. 2014. Catch me if you can: a cloud-enabled DDoS defense. In *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2014)*. Atlanta, GA USA, 264–275.

[21] Leonard Kleinrock. 1975. *Queueing Systems. Volume 1: Theory*. Wiley-Interscience.

[22] Gordon Fyodor Lyon. 2009. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure.

[23] Hoda Maleki, Saeed Valizadeh, William Koch, Azer Bestavros, and Marten van Dijk. 2016. Markov Modeling of Moving Target Defense Games. In *Proceedings of the 3rd ACM Workshop on Moving Target Defense (MTD 2016)*. ACM, Vienna, Austria, 81–92.

[24] Pratyusa K. Manadhata and Jeannette M. Wing. 2011. An Attack Surface Metric. *IEEE Transactions on Software Engineering* 37, 3 (May 2011), 371–386.

[25] Daniel A. Menascé. 2003. Security Performance. *IEEE Internet Computing* 7, 3 (May/June 2003), 84–87.

[26] Paulo Sousa, Nuno Ferreira Neves, Paulo Veríssimo, and William H. Sanders. 2006. Proactive Resilience Revisited: The Delicate Balance Between Resisting Intrusions and Remaining Available. In *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems (SRDS 2006)*. IEEE, Leeds, United Kingdom, 71–82.

[27] Luís Teixeira d'Aguiar Norton Brandão and Alysson Neves Bessan. 2012. On the reliability and availability of replicated and rejuvenating systems under stealth attacks and intrusions. *Journal of the Brazilian Computer Society* 18, 1 (March 2012), 61–80.

[28] Sridhar Venkatesan, Massimiliano Albanese, George Cybenko, and Sushil Jajodia. 2016. A Moving Target Defense Approach to Disrupting Stealthy Botnets. In *Proceedings of the 3rd ACM Workshop on Moving Target Defense (MTD 2016)*. ACM, Vienna, Austria, 37–46.

[29] David Watson, Matthew Smart, G. Robert Malan, and Farnam Jahanian. 2004. Protocol Scrubbing: Network Security Through Transparent Flow Modification. *IEEE/ACM Transactions on Networking* 12, 2 (April 2004), 261–273.

[30] Kara Zaffarano, Joshua Taylor, and Samuel Hamilton. 2015. A Quantitative Framework for Moving Target Defense Effectiveness Evaluation. In *Proceedings of the 2nd ACM Workshop on Moving Target Defense (MTD 2015)*. ACM, Denver, CO, USA, 3–10.