Analysis of VM Migration Scheduling as Moving Target Defense against insider attacks

Matheus Torquato
University of Coimbra, CISUC, DEI
Coimbra, Portugal
mdmelo@dei.uc.pt
Federal Institute of Alagoas, Campus Arapiraca
Arapiraca, Brazil
matheus.torquato@ifal.edu.br

Paulo Maciel
Centro de Informática, Universidade Federal de Pernambuco (CIn-UFPE)
Recife, Brazil
prmm@cin.ufpe.br

Marco Vieira
University of Coimbra, CISUC, DEI
Coimbra, Portugal
mvieira@dei.uc.pt

ABSTRACT

As cybersecurity threats evolve, cloud computing defenses must adapt to face new challenges. Unfortunately, due to resource sharing, cloud computing platforms open the door for *insider* attacks, which consist of malicious actions from cloud authorized users (e.g., clients of an Infrastructure-as-a-Service (IaaS) cloud) targeting the co-hosted users or the underlying provider environment. Virtual machine (VM) migration is a Moving Target Defense (MTD) technique to mitigate *insider* attacks effects, as it provides VMs positioning manageability. However, there is a clear demand for studies quantifying the security benefits of VM migration-based MTD considering different system architecture configurations. This paper tries to fill such a gap by presenting a Stochastic Reward Net model for the security evaluation of a VM migration-based MTD. The security metric of interest is the probability of attack success. We consider multiple architectures, ranging from one physical machine pool (without MTD) up to four physical machine pools. The evaluation also considers the unavailability due to VM migration. The key contributions are i) a set of results highlighting the probability of *insider* attacks success over time in different architectures and VM migration schedules, and ii) suggestions for selecting VMs as candidates for MTD deployment based on the tolerance levels of the attack success probability. The results are validated against simulation results to confirm the accuracy of the model.

CCS CONCEPTS

• **Security and privacy** → **Distributed systems security**; • **Computing methodologies** → **Model development and analysis**; • **Computer systems organization** → *Availability*;

KEYWORDS

Moving Target Defense, VM migration, Migration-based Dynamic Platform, Availability, Stochastic Petri Nets

SAC '21, March 22–26, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8104-8/21/03…$15.00
https://doi.org/10.1145/3412841.3441899

ACM Reference Format:
Matheus Torquato, Paulo Maciel, and Marco Vieira. 2021. Analysis of VM Migration Scheduling as Moving Target Defense against insider attacks. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3412841.3441899

1 INTRODUCTION

Despite being widely studied in the last years, recent surveys show that cloud computing security still requires further improvements [11][24]. One of the most challenging issues in cloud computing security is the *asymmetric advantage* of the attackers. The economic cost for building robust defenses is (usually) higher than the cost for conducting a cybersecurity attack [5]. Moreover, the attackers can exploit a single system vulnerability, while the defenders should protect the system from all possible attack vectors.

Moving Target Defense (MTD) [15] reduces the *asymmetric advantage* by applying dynamic environment reconfiguration aiming at defending or thwarting attacks [7]. The key idea is to continuously shift the attack surface to increase complexity for the environment reconnaissance or react to attacks dynamically [21].

A usual strategy for MTD in the cloud is VM migration [28]. The standard approach is to move VMs to prevent them from attacking co-resident VMs, or the underlying physical host [31][22]. However, there is a need for evaluation approaches to quantify the security benefits and availability impact when adopting different scheduling of VM migration as MTD against *insider* attacks (i.e., MTD timing problem [23]).

Wang et al. [31] proposed an algorithm for selecting proper MTD timing to minimize the associated costs. Their work provided insights on how to evaluate different MTD timing approaches. Penner and Guirguis [22] proposed a set of MTD mechanisms against Multi-Armed Bandit (MAB) policy attacks. Our work considers a similar threat model, but instead of MAB policies, we consider a multi-stage attack (i.e., the attacker may first reconnoiter the environment before launching the attack). Connell et al. [9] presented a Markov model for performance evaluation of an MTD deployment in a virtualized environment. Their work provided insights on i) how to model the MTD problem and ii) validation through simulation. Unlike these works, we decided to follow a straightforward approach of applying the usual VM migration scheduling as MTD (i.e., without requiring specific MTD implementations of third-party

software). Moreover, our analysis aims at comparing different *environmental configurations* (i.e., different system architectures and VM migration scheduling), showing their benefits and drawbacks regarding availability and probability of attack success. Furthermore, we propose *tolerance level* metric in our analysis. Based on the probability of attack success tolerated levels, *tolerance levels* support the selection of VMs as *candidates* for MTD deployment based on their expected runtime.

This paper presents a Stochastic Reward Net (SRN) model for the evaluation of the probability of *insider* attack success in an Infrastructure-as-a-Service (IaaS) cloud with MTD based on VM migration scheduling. The main goal is to evaluate the availability and security of the MTD deployment in different scenarios. These scenarios comprise different system architectures (i.e., set of available physical machine pools) and different VM migration scheduling policies.

The considered MTD aims at moving the *attacker's* VM between the available physical machine pools. Each physical machine pool has a unique hypervisor. Our threat model considers that the hypervisor is the target of the *insider* attack. Therefore, each time we move the VM, the *attacker* needs to reconnoiter the hypervisor variant before proceeding.

The following research questions guided this research:

- RQ_1*:* What is the attack success probability reduction when using different system architectures?
- RQ_2*:* What is the availability and attack success probability impact due to different VM migration scheduling?
- RQ_3*:* What is the time required for the system to reach a specific attack success probability (i.e., *tolerance level*) considering different architectures and VM migration scheduling?

To address these questions, we evaluated two case studies. The first focuses on reducing the attack success probability when enlarging the system architecture from one physical machine pool to four physical machine pools. The second case study investigates the reduction of the attack success probability varying the VM migration scheduling. We considered policies with *30 minutes*, *1 hour*, *6 hours*, *12 hours* and *24 hours* between VM migrations. Our results highlight the tradeoff between availability and security. For example, applying the policy *30 minutes* in a system with four physical machine pools, the probability of attack success at 24 hours is less than 1%. When applying the policy *12 hours* in the same conditions, the probability is 23%. However, the system downtime due to VM migrations at 24 hours is about 8 seconds for the policy *12 hours* and 3 minutes for the policy *30 minutes*.

Up to our knowledge, this is the first paper to investigate how different *environmental configurations* affect the security and availability levels of an IaaS cloud with VM migration scheduling as MTD against *insider* attacks. VM migration scheduling is already used in other contexts, such as software rejuvenation [3], load balancing [13], and sustainability [12]. Our contribution may be of interest to system managers who intend to design multi-objective VM migration scheduling policies.

The subsequent parts of this paper are organized as follows. Section 2 presents the assumptions behind this research and Section 3 discusses the SRN model proposed. Section 4 presents the case studies. Section 5 presents the model's validation using simulation. Section 6 presents some threats for validity and limitations of our work. Section 7 discusses related work. Finally, Section 8 presents conclusions and future work directions.

2 ASSUMPTIONS

The assumptions of this work are divided into three groups. The first (Section 2.1) is related to system architecture. The second (Section 2.2), is the threat model considered. The last (Section 2.3) is related to the Moving Target Defense approach considered.

2.1 System Architecture

We consider a set of architectures, ranging from *1N* (with one physical machine pool - ***baseline***) up to *4N* [1] (with four different physical machine pools). Figure 1 displays the system configuration for a *2N* architecture. The considered architecture has the following characteristics:

(1) Each physical machine pool has its unique hypervisor variant (hypervisor is the target of the *attacker's* insider attack).
(2) It is possible to migrate VMs between the physical machine pools.
(3) Each physical machine pool has at least one physical machine available to receive migrations.

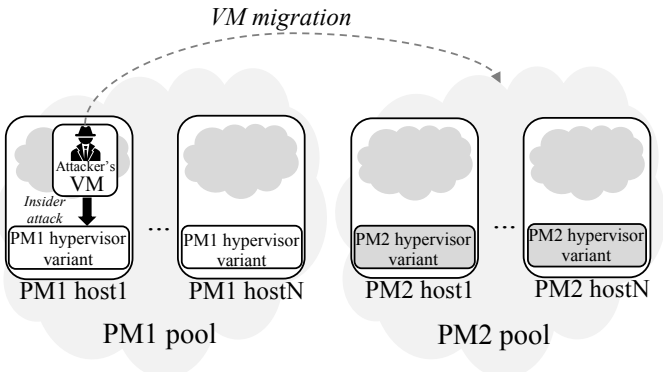


Figure 1: System configuration - *2N* architecture

2.2 Threat Model

The *attacker* has authorized control of one VM in the environment. The *attacker* goal is to perform an *insider* attack targeting the underlying hypervisor[2], which is the middleware between the VMs and the physical host.

When assuming the control of the hypervisor, the *attacker* can monitor or compromise co-resident VMs. We assume that the *attacker* will not resign until the attack success. The *insider* attack has two phases: i) *reconnaissance* - when the *attacker* tries to identify the hypervisor variant of the host; and ii) *attack* - when the *attacker* starts to perform malicious actions against the host hypervisor. In the *attack* phase, the *attacker* adopts a try-and-error approach.

[1] We decided to scale the model up to *4N* because we found that there are four major hypervisors (i.e., the target of the attack) used in server environments - Xen, Hyper-V, ESXi, KVM.
[2] Note that some research works may refer to this attack as VM escape [34].

Consequently, the longer the time spent on the same physical host, the greater the chance of attack success. Besides, once the *attacker* reconnoiters the host's hypervisor, it is possible to continue the try-and-error approach, ignoring the failed attempts (i.e., the *attacker* accumulates knowledge).

2.3 MTD defense

The adopted MTD consists of a time-based VM migration policy that moves the *attacker's* VM between the available physical machine pools[3]. The MTD goal is to reduce the probability of attack success through dynamic changes in the attack target (i.e., hypervisor variant of the physical host). In particular, we consider the VM migration between heterogeneous hypervisors [16].

Focusing on simplifying the MTD deployment, the VM migration policy follows a circular approach for any of the considered architectures. The circular approach consists of migrating the VM from one physical machine (PM) pool to the next physical machine pool. For example, let us suppose that the *attacker* is in the PM1 pool in a *3N* architecture. We first migrate the VM from the PM1 pool to the PM2 pool. Then, from PM2 to PM3, in the next migration round. Finally, from PM3 to PM1, restarting the VM migration cycle. The adoption of other migration schemes (e.g., random) is one of our future works.

For illustration purposes, consider the following example. Figure 2 shows an attack and defense flow in a *2N* architecture. The ***Stage 1*** is the initial state. We suppose that the VM of the attacker is in the PM1 pool. After the *reconnaissance* phase, the attacker starts the attack phase. The time spent in the attack phase is counted as attack progress. Then, the VM migration schedule arrives, moving the VM of the attacker from the PM1 pool to the PM2 pool, thus starting ***Stage 2***.

In ***Stage 2***, the attacker follows the same approach of ***Stage 1***. As before, the attacker has no accumulated knowledge about the PM2 pool environment and has to start the attack from the beginning. Finally, when the schedule for VM migration arrives again, the VM is moved back to the PM1 pool (***Stage 3***). In ***Stage 3***, after reconnaissance, the attacker leverages from the accumulated knowledge to finish the attack.

We are aware that the detection of an attacker in the environment is a rather difficult task. Besides, if the attacker is detected, it is more straightforward to delete the *attacker's* VM instead of migrating it. However, our strategy supposes that the *attacker* is undetected, has privacy rights, and has authorized access to a VM. Thus, a concern may arise on deciding which VMs should follow the MTD policy. System managers may leverage our *tolerance levels* results to select VMs as *candidates* for MTD deployment based on their expected runtime. Nevertheless, this MTD deployment may have a non-negligible impact on system performance and availability. We studied the availability impact on Section 4.2. The performance impact is yet to be studied.

The proposed MTD brings benefits as it is easy to use (assuming that VM migration is a common task in IaaS cloud management). However, unless we have very frequent migrations, which may affect VM availability, the *attacker* will eventually be successful. Hopefully, the MTD may provide valuable extra time to enhance the defensive mechanisms against *insider* attacks.

[3]In other papers (e.g., [33]), this approach is named as Migration-based Dynamic Platform.

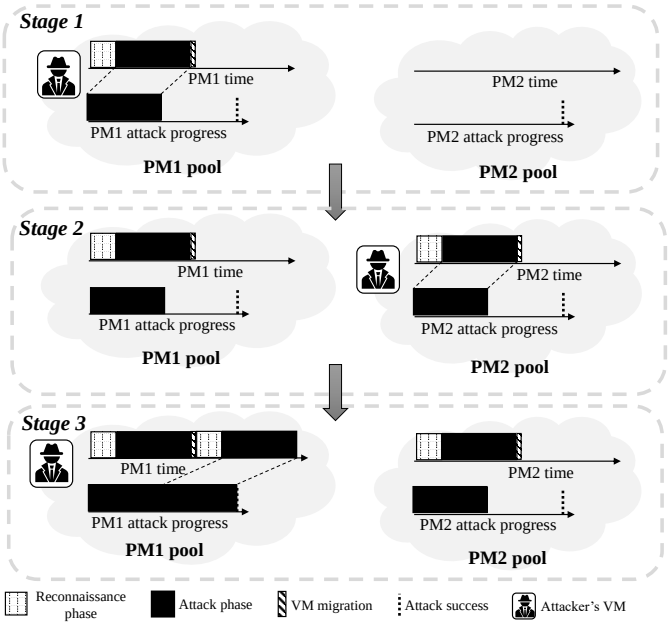


Figure 2: Example of attack and defense flow - *2N* architecture

3 MODEL

The proposed model has two submodels: a) Clock model and b) System model (see Figure 3). The models interact through the guard functions described in Table 1[4]. For simplicity, this section presents only the models related to the *2N* architecture. Nevertheless, we consider up to the *4N* architecture in our evaluations. We emphasize that each architecture has its model. All the models follow the same approach used in the *2N* architecture model. The only difference is that the other architectures *3N* and *4N* have three and four physical machine pools, respectively. *1N* architecture has only one physical machine pool and does not support VM migration. Nevertheless, we added the results from *1N* architecture as ***baseline*** results.

Table 1: Guard functions

Guard	Enabling function
$Gf1$	`(#VMMigPM1_2>0) OR (#VMMigPM2_1>0)`
$Gf2$	`(#Schedule>0)`

The Clock model represents the system component responsible for triggering VM migration. To assure that the model represents the predefined VM migration scheduling, transition `Trigger` adopts a deterministic firing delay. Its firing delay is the configured delay

[4]Guard functions are Boolean expressions evaluated based on the net current marking. They disable the associated transition when the boolean expression returns *false*. `(#P>0)` expression returns *true* if the number of tokens in the place `P` is greater than zero (otherwise, it returns *false*).

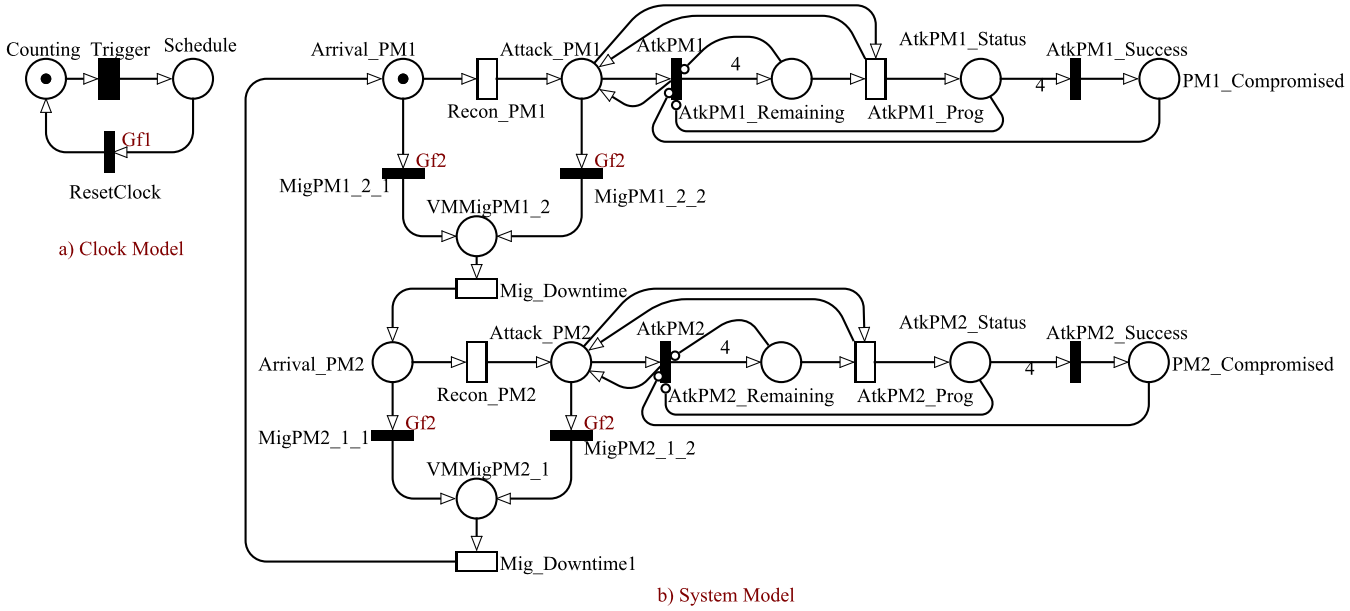


Figure 3: SRN model for probability of attack success evaluation (*2N* architecture)

between VM migrations. Thus, transition `Trigger` firing represents that the system reached the time interval for VM migration. Transition `Trigger` firing removes a token from place `Counting` to the place `Schedule`, thus activating guard function $Gf2$.

$Gf2$ enables the firing of transitions related to VM migration (i.e., `MigPM1_2_1`, `MigPM1_2_2`, `MigPM2_1_1` and `MigPM2_1_2`). These transitions firing put a token in place `VMMigPM1_2` or place `VMMigPM2_1`, indicating that the VM migration is in progress. In this situation, the system clock starts the time counting for the next migration (i.e., transition `ResetClock` firing). Transition `ResetClock` firing moves the token from place `Schedule` to the place `Counting` restarting the cycle of Clock model.

The System model in Figure 3 represents the *2N* architecture. Transitions and places related to the PM1 pool have an explicit mention to `PM1`. Transitions and places related to the PM2 pool have `PM2` in their names.

In this model, we consider that the *attacker's* VM is initially in the PM1 pool (place `Arrival_PM1` with one token). From this state, we have two possibilities. The first is when the VM migration schedule arrives before the attacker finishes the *reconnaissance* phase. In this case, obeying $Gf2$, the token from place `Arrival_PM1` is moved to place `VMMigPM1_2` through transition `MigPM1_2_1` firing. The second possibility is that the attacker finishes the *reconnaissance* phase (transition `Recon_PM1` firing). Transition `Recon_PM1` firing removes the token from place `Arrival_PM1` and puts a token in place `Attack_PM1`.

Following the same approach of our previous work [27], we model the *attack* phase using a four-phase Erlang sub-net. The Erlang sub-net has the immediate transition `AtkPM1`, transition `AtkPM1_Prog`, and the places `AtkPM1_Remaining` and `AtkPM1_Status`. There are two reasons for this choice. First, we need to preserve the attack progress even after a VM migration. Thus, we need a model with a *enabling memory policy* [18]. The Erlang sub-net (specifically the place `AtkPM1_Status`) serves as a *memory* of the attack progress. Secondly, a four-phase Erlang sub-net can represent an Increasing Failure Rate (IFR) [29]. As mentioned before, the attacker adopts a try-and-error approach. Thus, the attack success probability increases as long as the attacker stays in a specific physical machine pool. Therefore, we approximate this increasing probability using the four-phase Erlang sub-net. It is worth highlighting that previous works also adopted hypoexponential distributions to represent IFR [17][26].

Transition `AtkPM1` immediately fires when place `Attack_PM1` receives a token. Transition `AtkPM1` firing *swaps*[5] the token in the place `Attack_PM1`, and puts four tokens (representing the number of phases) in the place `AtkPM1_Remaining`. This event represents the start of the *attack* phase. Each transition `AtkPM1_Prog` firing represents that the attack is progressing. Note that the transition `AtkPM1_Prog` is only enabled when we have tokens in the place `AtkPM1_Remaining` (representing that the *attack* phase is not over) and place `Attack_PM1` (indicating that the VM of the attacker is still in the PM1 pool). Transition `AtkPM1_Prog` moves tokens from place `AtkPM1_Remaining` to place `AtkPM1_Status`.

Transition `AtkPM1_Success` immediately fires when the place `AtkPM1_Status` receives the fourth token. Transition `AtkPM1_Success` firing collects four tokens from place `AtkPM1_Status` and puts one token in the place `PM1_Compromised`. This event denotes that the attacker successfully compromised a physical machine in the PM1 pool.

We put an inhibitor arc[6] from place `PM1_Compromised` to transition `AtkPM1`. This inhibitor arc denotes that the *attacker* does not

[5] Receives and gives back
[6] An arc terminating in a circle instead of an arrowhead

need to pass the try-and-error approach again once the system is compromised.

Timely VM migrations delay the attack success. Lets suppose that the attacker is on the *attack* phase in PM1 pool (place `Attack_PM1` with one token). As transition `MigPM1_2_1`, the transition `MigPM1_2_2` also has embedded guard function $Gf2$. Therefore, when the system reaches the time for VM migration, it moves the *attacker's* VM from a physical machine of PM1 pool to a physical machine in the PM2 pool. This VM migration interrupts the *attack* phase progress. Transition `MigPM1_2_2` firing moves the token from place `Attack_PM1` to the place `VMMigPM1_2`. After the VM migration downtime (transition `Mig_Downtime`), the VM arrives in the PM2 pool (place `Arrival_PM2` receives a token from transition `Mig_Downtime` firing).

The attacking approach in the PM2 pool follows the same procedure described above. When the system triggers another VM migration, the attacker's VM is moved back to the PM1 pool. The attacker can leverage the obtained knowledge from his first attempts in the previous *attack* phase in the PM1 pool.

In the context of this paper, we used the proposed model to evaluate VM migration as MTD. However, the approach used in the model design may be valuable to evaluate other related MTD mechanisms. For example, the use of IP address shuffling against a persistent attacker.

Metrics computation. The main metric of interest is the **probability of attack success**. Assuming a *2N* architecture, we compute this metric by observing the transient probability of token presence in place `PM1_Compromised` or place `PM2_Compromised`. We compute **availability** by observing the steady-state probability of token presence in any of the following places: `Arrival_PM1`, `Attack_PM1`, `Arrival_PM2`, or `Attack_PM2`. Note that this evaluation only covers the availability impacts due to VM migration scheduling. Other dependability events as failures and repairs are out of the scope.

4 CASE STUDIES

This section presents two case studies. The first one estimates the probability of attack success in each proposed system architecture (Section 4.1). The second shows the results considering different VM migration scheduling policies (Section 4.2).

We used the TimeNet tool [35] for model design and evaluation. Table 2 presents the default values used for our evaluations. We obtained these values from the recent papers [26][6]. Note that these values are only for reference and should be adapted whenever measurement-based results are available. Nevertheless, we find these values reasonable to represent the considered threat model and MTD defense based on previous papers published in reputed journals. We discuss this topic more in Section 6.

4.1 CS#1 - Varying number of available physical machine pools

RQ1: What is the reduction in the probability of attack success when using different system architectures?

Table 2: Parameters used in the timed transitions

Transition	Description	Delay
`Trigger`	Time for VM migration	30 minutes
`Recon_PM1`, `Recon_PM2`	*Reconnaissance* phase	30 minutes
`AtkPM1_Prog`, `AtkPM2_Prog`	*Attack* phase (Erlang)	6 hours*
`Mig_Downtime`, `Mig_Downtime1`	VM migration downtime	4 seconds
* As we have four Erlang phases, *attack* phase total delay is of 24 hours		

Observing the threat model and proposed MTD (Section 2.1), we can conclude that the higher is the number of available physical machine pools, the longer is the delay for an attack success. However, this evaluation aims to quantify the reduction of attack success when using the proposed system architectures. In practice, this case study has two goals. The first is to evaluate how long the system survives the attack. The second is to quantify the benefits of enlarging the architecture.

Figure 4 presents the results of the probability of attack success in different architectures. These results comprise the first fifteen days (360 hours) of *attacker* presence in the environment.

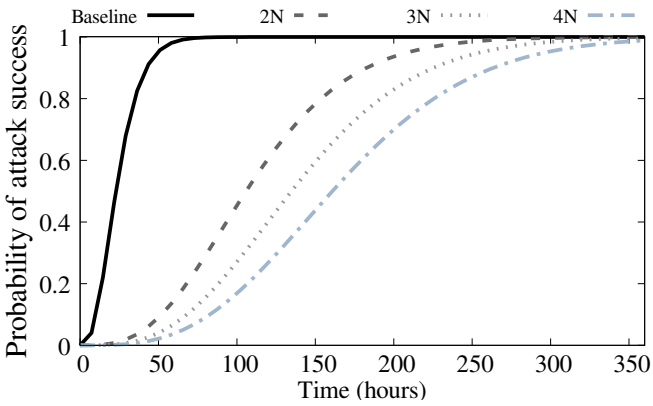


Figure 4: Probability of attack success - varying number of available physical machine pools

The results from the architectures with MTD (i.e, *2N*, *3N*, and *4N*) are significant better than the results from the baseline architecture (i.e., *1N*). As expected, the enlargement of the system architecture produces a flattening effect in the probability of attack success curve. However, the difference between the architectures with MTD is less prominent.

For comparison purposes, Table 3 presents when the system reaches 1%, 50%, and 90% of probability of attack success (i.e., *tolerance levels*) in each proposed architecture. These findings may help to perceive the benefits of MTD deployment. The probability of an attack success decays significantly in the MTD-enabled environment. We noticed that the smaller architecture with MTD (*2N*) is

about three times more *secure*[7] than the baseline architecture in the first week of attack presence (up to 168 hours).

Table 3: When the system reaches probability of attack success of 1%, 50%, and 90% (tolerance levels)

Arch.	1%	50%	90%
Baseline	6 hours	23 hours	41 hours
2N	25 hours	106 hours	182 hours
3N	34 hours	135 hours	224 hours
4N	41 hours	161 hours	263 hours

RQ3: What is the time required for the system to reach a specific attack success probability considering different architectures and VM migration policies?

The results presented in Table 3 also provide a partial answer for RQ_3. Depending on the business model, system managers may be more or less concerned about security. Thus, the system manager may define appropriated tolerance levels of probability of attack success for their respective environments. For example, in a business with a high associated security risk, the system manager may set the tolerance level at 1%. In this situation, assuming our evaluation's scope, the system should flag the VMs with expected runtime above 6 hours as candidates for MTD deployment. After further verification of other relevant aspects (e.g., verification of what the client started the VM), the system manager can decide whether or not such a VM should follow the VM migration scheduling.

Figure 5 presents the results of the reduction in the probability of attack success. This reduction is the difference between the probability of attack success of each MTD architecture (*2N*, *3N*, and *4N*) and the ***baseline*** architecture. As expected, *4N* architecture provides a more significant reduction than the others. The interesting conclusion is that, after fifteen days (i.e., 360 hours) of attacker presence in the environment, the reduction is less than 1%. Therefore, in scenarios with VMs that impose long-running execution times (above fifteen days), a *4N* architecture is not enough to reduce the probability of attack success. In such scenarios, we encourage the system managers to apply alternative policies to mitigate the vulnerabilities related to *insider* attack. The deployment of periodical software rejuvenation and routines to clean up the hypervisor may improve system security in such scenarios.

4.2 CS#2 - Varying VM migration schedule - *4N* architecture

RQ2: What is the availability and probability of attack success impact due to different VM migration scheduling policies?

In the previous case study, we noticed security improvements due to the MTD deployment. However, as the MTD is based on VM migration scheduling, MTD policies with frequent VM migrations may also affect system availability. Clark et al. [8] show that, even in live migration mode, each VM migration has an associated downtime. In some cases, the accumulated VM migration downtime may be unacceptable. Thus, it is essential to evaluate different VM migration schedules to verify MTD policies' availability impact.

[7]By *secure* we mean with lower probability of attack success.

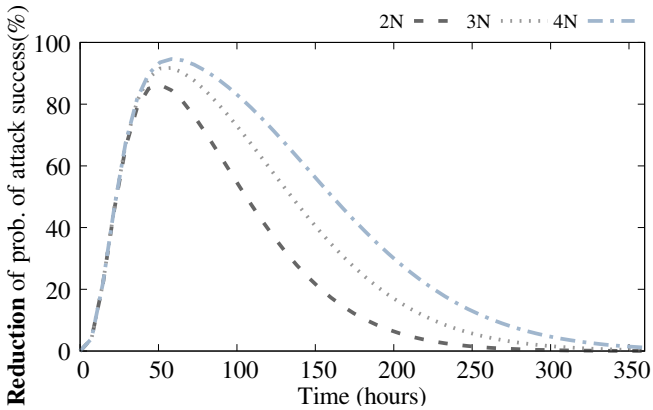


Figure 5: Reduction of probability of attack success due to the number of available physical machine pools

The threat model and MTD defense impose a tradeoff between security and availability. Therefore, more frequent migrations may delay the attack success longer but also imposes more system downtime. Moreover, less frequent migrations may reduce the system downtime due to MTD policy, increasing the probability of attack success.

Figure 6 presents the probability of attack success results for different VM migration trigger intervals. Figure 7 presents the reduction due to each VM migration scheduling policy. Due to space limitations, this evaluation only presents the results of *4N* architecture.

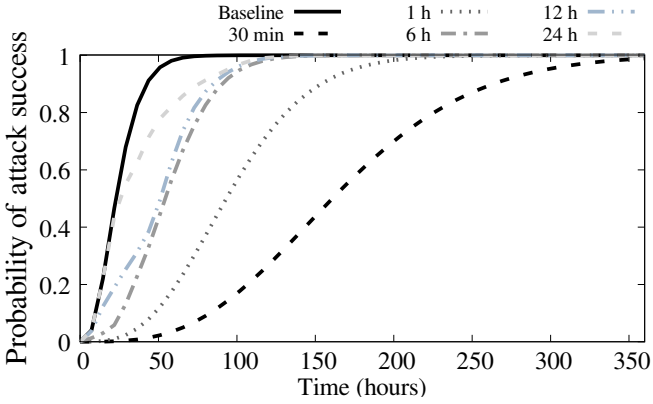


Figure 6: Probability of attack success - varying VM migration trigger - *4N* architecture

As expected, more frequent migrations (e.g., VM migration scheduling of *30 minutes* and *1 hour*) produce a more noticeable reduction in the probability of attack success. In the first hours of attacker presence, less frequent migration policies provide no significant decrease in attack success probability. Besides that, Figure 7 shows that less frequent migrations produce only a slight reduction effect on the probability of attack success.

System managers may be interested in two aspects of the VM migration scheduling when designing the MTD policy: i) **System**

Table 4: Tolerance levels and system unavailability - Different VM migration schedules - *4N* architecture

Trigger	TL *1%*	TL *50%*	TL *90%*	Availability considering VM migrations	Downtime in the first fifteen days (due to VM migrations)
30 minutes	41 hours	161 hours	263 hours	0.997778	48 minutes
1 hour	24 hours	94 hours	152 hours	0.998889	24 minutes
6 hours	6 hours	54 hours	89 hours	0.999815	4 minutes
12 hours	6 hours	52 hours	83 hours	0.999907	2 minutes
24 hours	6 hours	23 hours	71 hours	0.999954	1 minute

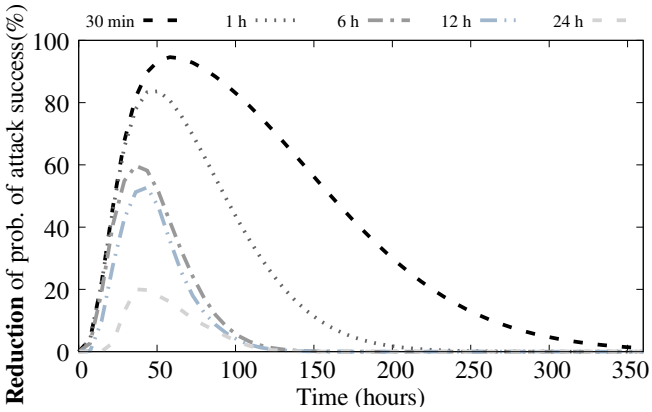


Figure 7: Reduction of probability of attack success due to the variation on VM migration trigger

availability, to understand how the VM migration scheduling affects system downtime; and ii) when does the system reach the **tolerance level** (TL) of the probability of attack success. We summarized these results in Table 4.

In systems with tolerance level of 1%, the managers may adopt MTD policies with more frequent migrations (e.g., VM migration trigger of *30 minutes* or *1 hour*). Because the other ones provide no security improvement when compared to the *baseline* results.

Availability results reveal a significant system downtime in scenarios with frequent migrations. On the other hand, the downtime in the first fifteen days (i.e., 360 hours) in below 4 minutes when using VM trigger above *6 hours*. However, we highlight that the downtime of each VM migration is usually short [8]. The VM memory state is preserved during VM migration. Therefore, VM migration downtime may produce a more severe impact in systems with an intense load of external requests. In scenarios without a network buffer, these requests will be lost due to environment unavailability during migrations. Nevertheless, there may be scenarios where this downtime is acceptable (e.g., systems that do not require high availability).

5 VALIDATION WITH SIMULATION RESULTS

Following the same idea of Connell et. al. [9], we implemented a simulation environment using SimPy[8]. SimPy provides a simulation framework using standard Python language. We used SimPy to simulate the attack progress and the interruptions due to VM migration occurrence. We implemented the simulation script by hand (i.e., without using automation or conversion frameworks). The proposed model guided the simulation implementation, where we used variables to store the system state and events to represent the attack progress and VM migration.

Figure 8 presents: i) the comparison of model (black line) and simulation (gray dashed line) results and ii) *error* - difference between the model and simulation results. The simulation results dotted lines represent the 95% confidence interval. The results below are only from the *2N* architecture. However, the comparisons between simulation and model results for the other architectures have similar results.

The *error* is higher in scenarios with more frequent VM migrations. In scenarios with more frequent VM migrations, the simulation environment has to generate more events leading to more accumulated errors. Nevertheless, in scenarios with less frequent VM migrations, the *error* results are relatively low.

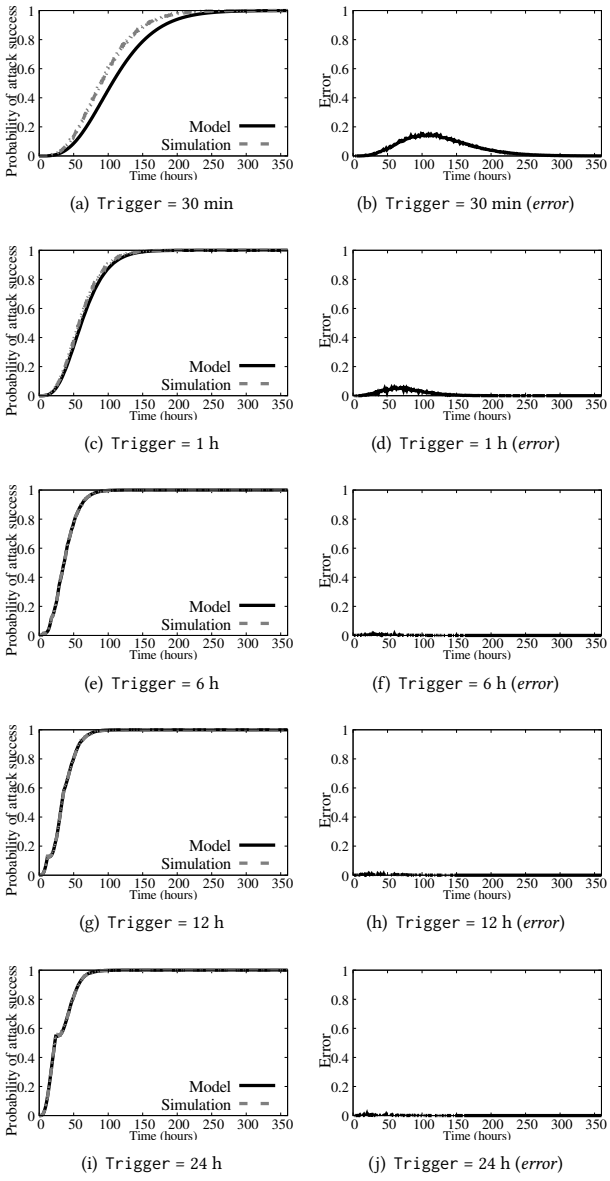
The *error* results remains under 0.2 in all considered scenarios of VM migration scheduling. The maximum *error* for the policy *30 minutes* is 0.151552, and the maximum *error* for the policy *1 hour* is 0.051828. For all the other scenarios, the maximum *error* is about 0.01. Figures 8(e), 8(g), and 8(i) show that the simulation results are nearly the same of the model results.

6 THREATS TO VALIDITY AND LIMITATIONS

Model-based evaluations are less accurate than measurement-based evaluations [14]. However, model-based evaluation suits our needs as we aim to evaluate different architectures, which may be a significant challenge for measurement-based evaluation. Besides that, as presented in [25] and [20], model-based evaluation is helpful when combining security and dependability metrics. As our research line evaluates the probability of attack success and availability, Stochastic Reward Net models seem to be a reasonable evaluation method.

The presented availability evaluation is limited. A more comprehensive availability evaluation of VM migration effects is needed to understand a VM migration scheduling policy's real impacts. Two factors are missing in our evaluation. Firstly, other relevant dependability events as failures, crashes, and software aging. Secondly, the VM migration failure probability. Despite these aspects' relevance, note that this work's focus is the security evaluation (i.e., probability of attack success evaluation). The inclusion of other relevant aspects of VM migration scheduling may raise difficulties (e.g., largeness [30]) in the model evaluation. An approach to address this problem in the future is hierarchical compositions or interacting SRN models [30].

[8]Available at: https://simpy.readthedocs.io/en/latest/



(a) Trigger = 30 min (b) Trigger = 30 min (*error*)

(c) Trigger = 1 h (d) Trigger = 1 h (*error*)

(e) Trigger = 6 h (f) Trigger = 6 h (*error*)

(g) Trigger = 12 h (h) Trigger = 12 h (*error*)

(i) Trigger = 24 h (j) Trigger = 24 h (*error*)

Figure 8: Model and simulation results - 2N architecture

We assumed default values for the attack and reconnaissance phases duration. In the best scenario, we should have obtained these parameters through experimentation. However, such experimentation has a non-negligible associated cost. An approach is to hire a *red team* [10] to test an real testbed. We decided to follow a lower cost approach by collecting the parameters from published papers. We noticed that similar approaches were used in several papers of model-based security evaluations [19][32][1].

7 RELATED WORKS

Mendonça et al. [19] presented a model for performability evaluation of a system with a time-based Moving Target Defense. The proposed MTD leverages Software-Defined Networking (SDN) to perform the environment modifications. The authors neglect the security improvement assessment due to MTD deployment. Unlike their work, we focus on the security evaluation regarding the probability of attack success.

Chen et al. [6] presented an SRN model for job finish time evaluation of a system with MTD based on VM Migration. The paper offered relevant insights into our model design and parameterization. The presented results focus on the job finish time under different conditions. Unlike their work, we focused on evaluating the MTD *effectiveness* under different architectures and VM migration scheduling policies. By *effectiveness*, we mean a reduction in the probability of attack success.

Cai et al. [4] presented a Petri Net (PN) model for MTD evaluation and comparison. Their research was one of the first efforts to use PN in the context of Moving Target Defense. Their work focuses on the MTD deployment in a Web Server. Their metrics of interest are related to performance (e.g., the average delay of service, system throughput, and operational efficiency). The authors decided to present their results showing the aspects of the model (e.g., number of tokens on a place, throughput of timed transitions, and sojourn times for tangible states). By adopting these results, it is possible to extract the desired metrics. Unlike this approach, we decided to deliver the results directly, instead of showing the model's aspects. We hope that this approach reduces the obstacles to understanding our results. Finally, different from the authors, we present a security and availability evaluation instead of a performance evaluation.

Connell et al. [9] presented comprehensive models for availability, security, and performance evaluation of an environment with MTD. As our work, the authors also covered the probability of attack success and availability in their evaluations. However, we select a different modeling strategy. First, we explicitly used a multi-stage attack as our threat model. Furthermore, we decided to use an Erlang sub-net to represent the IFR related to the attack phase.

Chang et al. [33] provided an SRN model for job completion time evaluation in an MTD system that considers a virtualized platform with Software Defined Networking capabilities. They also computed the availability and probability of attack success. The main goal of their paper is to investigate the MTD impact on job protection and performance. The significant difference between our approach and theirs is the threat model. In their work, they consider the attacker to select the attack targets without accumulating knowledge. In their threat model, the attacker is trying to compromise a specific job. Besides, different from their work, we also present the *tolerance levels* results, aiming to support managers' decision-making.

Alavizadeh et al. [2] provided models for evaluating Shuffle and Diversity MTD techniques in cloud computing environments. The authors consider the probability of attack success but neglect availability evaluation. Besides that, their models adopt Hierarchical Attack Representation Models (HARM). HARM models focus on the security evaluation. Unlike their approach, we decided to use SRN models, as they allow us to compute dependability and security metrics using the same model.

8 CONCLUSIONS AND FUTURE WORKS

This paper presented an SRN model to evaluate the probability of attack success and an MTD system's availability based on VM migration scheduling policies. We showed a set of case studies to exercise our model. The obtained results show a tradeoff between the probability of attack success and availability when using the proposed MTD. The model is validated against simulation results.

Using the proposed *tolerance levels*, it is possible to select VMs as *candidates* for MTD deployment. Moreover, the set of results quantify the benefit of enlarging system architectures. In some scenarios, the benefit of using large architectures is negligible.

As future works, we intend to investigate the tradeoffs between security, availability, and performability when applying MTD based on VM migration scheduling. Besides that, we aim to expand the evaluations for other scenarios and migration schemes (e.g., more prominent architectures and different VM migration scheduling). Finally, we intend to analyze scenarios with different attack duration phases duration.

ACKNOWLEDGMENTS

This work has been partially supported by Portuguese Foundation for Science and Technology (FCT), through the PhD grant SFRH/BD/146181/2019, within the scope of the project CISUC - UID/CEC/00326/2020. This work is also funded by the European Social Fund, through the Regional Operational Program Centro 2020.

This work also received support from AIDA: (Adaptive, Intelligent and Distributed Assurance Platform) project, funded by Operational Program for Competitiveness and Internationalization (COMPETE 2020) and FCT (under CMU Portugal Program) through grant POCI-01-0247-FEDER-045907. And, from project TalkConnect funded by COMPETE 2020 trough grant POCI-01-0247-FEDER-039676.

REFERENCES

[1] Hooman Alavizadeh, Jin B Hong, Julian Jang-Jaccard, and Dong Seong Kim. 2018. Comprehensive security assessment of combined MTD techniques for the cloud. In *Proceedings of the 5th ACM Workshop on Moving Target Defense*. 11–20.

[2] Hooman Alavizadeh, Dong Seong Kim, and Julian Jang-Jaccard. 2019. Model-based evaluation of combinations of Shuffle and Diversity MTD techniques on the cloud. *Future Generation Computer Systems* (2019).

[3] Jing Bai, Xiaolin Chang, Fumio Machida, Kishor S Trivedi, and Zhen Han. 2020. Analyzing Software Rejuvenation Techniques in a Virtualized System: Service Provider and User Views. *IEEE Access* 8 (2020), 6448–6459.

[4] 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 *Conference on Advanced Computer Architecture*. Springer, 184–197.

[5] Samrat Chatterjee, Mahantesh Halappanavar, Ramakrishna Tipireddy, Matthew Oster, and Sudip Saha. 2015. Quantifying mixed uncertainties in cyber attacker payoffs. In *2015 IEEE International Symposium on Technologies for Homeland Security (HST)*. IEEE, 1–6.

[6] Zhi Chen, Xiaolin Chang, Zhen Han, and Yang Yang. 2020. Numerical Evaluation of Job Finish Time Under MTD Environment. *IEEE Access* 8 (2020), 11437–11446.

[7] Jin-Hee Cho, Dilli P Sharma, Hooman Alavizadeh, Seunghyun Yoon, Noam Ben-Asher, Terrence J Moore, Dong Seong Kim, Hyuk Lim, and Frederica F Nelson. 2020. Toward proactive, adaptive defense: A survey on moving target defense. *IEEE Communications Surveys & Tutorials* 22, 1 (2020), 709–745.

[8] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. 273–286.

[9] Warren Connell, Daniel A Menasce, and Massimiliano Albanese. 2018. Performance modeling of moving target defenses with reconfiguration limits. *IEEE Transactions on Dependable and Secure Computing* (2018).

[10] Yuri Diogenes and Erdal Ozkaya. 2018. *Cybersecurity??? Attack and Defense Strategies: Infrastructure security with Red Team and Blue Team tactics*. Packt Publishing Ltd.

[11] Ghanshyam Gagged and SM Jaisakthi. 2020. Overview on Security Concerns Associated in Cloud Computing. In *Smart Intelligent Computing and Applications*. Springer, 85–94.

[12] Chaima Ghribi, Makhlouf Hadji, and Djamal Zeghlache. 2013. Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 671–678.

[13] Jinhua Hu, Jianhua Gu, Guofei Sun, and Tianhai Zhao. 2010. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In *2010 3rd International symposium on parallel architectures, algorithms and programming*. IEEE, 89–96.

[14] Raj Jain. 1990. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons.

[15] Sushil Jajodia, Anup K Ghosh, Vipin Swarup, Cliff Wang, and X Sean Wang. 2011. *Moving target defense: creating asymmetric uncertainty for cyber threats*. Vol. 54. Springer Science & Business Media.

[16] Pengcheng Liu, Ziye Yang, Xiang Song, Yixun Zhou, Haibo Chen, and Binyu Zang. 2008. Heterogeneous live migration of virtual machines. In *International Workshop on Virtualization Technology (IWVT'08)*.

[17] Fumio Machida, Dong Seong Kim, and Kishor S Trivedi. 2013. Modeling and analysis of software rejuvenation in a server virtualized system with live VM migration. *Performance Evaluation* 70, 3 (2013), 212–230.

[18] M Ajmone Marsan, Gianfranco Balbo, Gianni Conte, Susanna Donatelli, and Giuliana Franceschinis. 1998. Modelling with generalized stochastic Petri nets. *ACM SIGMETRICS performance evaluation review* 26, 2 (1998), 2.

[19] Júlio Mendonça, Jin-Hee Cho, Terrence J Moore, Frederica F Nelson, Hyuk Lim, Armin Zimmermann, and Dong Seong Kim. 2020. Performability analysis of services in a software-defined networking adopting time-based moving target defense mechanisms. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. 1180–1189.

[20] Tuan Anh Nguyen, Dugki Min, and Eunmi Choi. 2020. A Hierarchical Modeling and Analysis Framework for Availability and Security Quantification of IoT Infrastructures. *Electronics* 9, 1 (2020), 155.

[21] Department of Homeland Security. 2020. Moving Target Defense. https://www.dhs.gov/science-and-technology/csd-mtd

[22] Terry Penner and Mina Guirguis. 2017. Combating the bandits in the cloud: A moving target defense approach. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 411–420.

[23] Sailik Sengupta, Ankur Chowdhary, Abdulhakim Sabur, Adel Alshamrani, Dijiang Huang, and Subbarao Kambhampati. 2020. A survey of moving target defenses for network security. *IEEE Communications Surveys & Tutorials* (2020).

[24] Hamed Tabrizchi and Marjan Kuchaki Rafsanjani. 2020. A survey on security challenges in cloud computing: issues, threats, and solutions. *The Journal of Supercomputing* (2020), 1–40.

[25] Matheus Torquato, Paulo Maciel, and Marco Vieira. 2019. A Model for Availability and Security Risk Evaluation for Systems With VMM Rejuvenation Enabled by VM Migration Scheduling. *IEEE Access* 7 (2019), 138315–138326.

[26] Matheus Torquato, Paulo Maciel, and Marco Vieira. 2020. Availability and reliability modeling of VM migration as rejuvenation on a system under varying workload. *Software Quality Journal* (2020), 1–25.

[27] Matheus Torquato, Paulo Maciel, and Marco Vieira. 2020. Security and Availability Modeling of VM Migration as Moving Target Defense. In *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE.

[28] Matheus Torquato and Marco Vieira. 2020. Moving Target Defense in Cloud Computing: A Systematic Mapping Study. *Computers & Security* (2020), 101742.

[29] Kishor Shridharbhai Trivedi. 1982. *Probability and statistics with reliability, queuing, and computer science applications*. Vol. 13. Wiley Online Library.

[30] Kishor S Trivedi and Andrea Bobbio. 2017. *Reliability and availability engineering: modeling, analysis, and applications*. Cambridge University Press.

[31] Huangxin Wang, Fei Li, and Songqing Chen. 2016. Towards cost-effective moving target defense against ddos and covert channel attacks. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*. 15–25.

[32] Yuanzhuo Wang, Jingyuan Li, Kun Meng, Chuang Lin, and Xueqi Cheng. 2013. Modeling and security analysis of enterprise network using attack–defense stochastic game Petri nets. *Security and Communication Networks* 6, 1 (2013), 89–99.

[33] x. Chang, Y. Shi, z. zhang, Z. xu, and K. Trivedi. 2020. Job Completion Time under Migration-based Dynamic Platform Technique. *IEEE Transactions on Services Computing* (2020), 1–1.

[34] Su Zhang. 2012. *Deep-diving into an easily-overlooked threat: Inter-VM attacks*. Technical Report. Technical Report). Manhattan, Kansas: Kansas State University.

[35] Armin Zimmermann. 2017. Modelling and performance evaluation with TimeNET 4.4. In *International Conference on Quantitative Evaluation of Systems*. Springer, 300–303.