The Journal of Systems and Software 219 (2025) 112222

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Evaluation of time-based virtual machine migration as moving target defense against host-based attacks☆

Matheus Torquato [a,c,*], Paulo Maciel [b], Marco Vieira [d,a]

[a] University of Coimbra, CISUC, DEI, Coimbra, Portugal
[b] Centro de Informática, Universidade Federal de Pernambuco (CIn-UFPE), Recife, Brazil
[c] Federal Institute of Alagoas, Campus Arapiraca, Arapiraca, Brazil
[d] University of North Carolina at Charlotte, NC, US

ARTICLE INFO

Keywords:
PyMTDEvaluator
Moving target defense
Petri nets
VM migration
Dynamic platform technique

ABSTRACT

Moving Target Defense (MTD) consists of applying dynamic reconfiguration in the defensive side of the attack-defense cybersecurity game. Virtual Machine (VM) migration could be used as MTD against specific host-based attacks in the cloud computing environment by remapping the distribution of VMs in the existing physical hosts. This way, when the attacker's VM is moved to a different machine, the attack has to be restarted. However, one significant gap here is how to select a proper VM migration-based MTD schedule to reach the desired levels of system protection. This paper develops a Stochastic Petri Net (SPN) model to address this issue. The model leverages empirical knowledge about the dynamics of the attack defense in a VM migration-enabled setup. First, we present the results of an experimental campaign to acquire knowledge about the system's behavior. The experiments provide insights for the model design. Then, based on the model, we propose a tool named *PyMTDEvaluator*, which provides a graphical interface that serves as a wrapper for the simulation environment of the model. Finally, we exercise the tool using Multi-Criteria Decision-Making methods to aid the MTD policy selection. Hopefully, our results and methods will be helpful for system managers and cybersecurity professionals.

1. Introduction

Virtual Machine (VM) migration appears among the relevant cloud computing Moving Target Defense (MTD) techniques (Torquato and Vieira, 2020). VM migration-based MTD is usually easy to deploy and manage as migrations are a usual cloud management task. Besides, the migration feature is widely available *off-the-shelf* in most virtualized systems.

VM migration-based MTD aims to counteract *host-based* attacks through VMs remapping in the available physical hosts. *Host-based* attacks are attacks from VMs targeting their own physical host (e.g., resource starvation attack) or co-hosted VMs. One common strategy of *host-based* attacks is to abuse the shared resources to affect co-hosted VMs (e.g., RowHammer attacks) (Patil and Modi, 2019). In the scope of this paper, we consider *host-based* attacks that aim to impair the victim's availability (i.e., availability attack[1]).

One of the main challenges for an effective VM-migration based MTD deployment is the timing function (i.e., when to deploy MTD) (Sengupta et al., 2020). A common strategy to address this issue is to adopt a schedule with regular intervals between MTD actions (Alhozaimy and Menascé, 2022). However, it is essential to set up proper evaluation methods to verify how the variation in the MTD schedule affects its effectiveness. Furthermore, it is essential to consider other important attributes to provide a more comprehensive evaluation approach. For example, high availability is often necessary for cloud computing environments (Buyya et al., 2018).

The context mentioned above leads us to the following research question: *What is the impact of the variation of MTD scheduling on the system availability and probability of attack success?* To address this question, this paper proposes a Stochastic Petri Net (SPN) model (Marsan et al., 1998) for the security and availability evaluation of a system with time-based MTD. The considered defensive posture consists of scheduling

☆ Editor: Prof. Raffaela Mirandola.

* Corresponding author at: University of Coimbra, CISUC, DEI, Coimbra, Portugal.
E-mail addresses: mdmelo@dei.uc.pt, matheus.torquato@ifal.edu.br (M. Torquato), prmm@cin.ufpe.br (P. Maciel), marco.vieira@uncc.edu (M. Vieira).
URL: https://www.matheustorquato.com (M. Torquato).

[1] We prefer to use *availability attack* instead of Denial of Service (DoS), as the latter is usually adopted in scenarios where the attacker is in the network, but in another device.

https://doi.org/10.1016/j.jss.2024.112222
Received 25 March 2024; Received in revised form 28 August 2024; Accepted 17 September 2024
Available online 20 September 2024

MTD actions (i.e., VM migration) at regular intervals. The approach is as follows. First, we study the effectiveness of VM migration as MTD through an experimental campaign. In this campaign, we instantiate the attack and defense model in a real testbed. Then, we use the knowledge we obtained about the system behavior in the model design. Finally, we exercise the model using hypothetical parameters to get the metrics of interest — availability, probability of attack success, capacity, and cost.

Since the proposed parameter values are illustrative and specific for a hypothetical case, we must update them accordingly to satisfy other scenarios. Besides that, for a comprehensive comparison, we need to run the model several times to obtain the metrics results for each studied MTD schedule. Despite the variety of Petri Net tools available,[2] the model evaluation process may be time-consuming and inconvenient in some cases. So, to ease the burden for the model analysis, we propose the *PyMTDEvaluator* tool. *PyMTDEvaluator* acts as a graphical interface for the proposed model. The tool provides a broad set of metrics in its output and allows the comparison of different MTD schedules.

The highlights of this paper are:

- **Experimental campaign to verify the effectiveness of VM migration as MTD.** We evaluate time-based VM migration against a specific host-based threat *Memory Denial of Service (MemDoS)* (Zhang et al., 2017). The obtained set of results highlights that timing plays a critical role in MTD effectiveness.
- **Stochastic Petri Net model to evaluate the probability of attack success and availability of a system under attack and MTD.** Based on the experimental campaign results and on prior knowledge, we present an SPN model to represent the attack-and-defense behavior. The proposed model can be expanded or adapted for similar scenarios.
- **Simulation tool to facilitate the evaluation process.** We implement the model in a simulation environment. The produced application allows the users to input their own parameters to exercise the proposed model.
- **Support to Multi-Criteria Decision Making to aid the selection of VM migration scheduling.** This paper also tries to address the question: *Which alternative is the best?* As the answer to this question heavily depends on the user's preference for each metric, the proposed tool supports the decision through Multi-Criteria Decision Making methods.

This paper extends our previous work (Torquato et al., 2021b), which presented the first version of *PyMTDEvaluator*. In that first version, the tool description paper presented a limited experimental background of MTD actions against host-based attacks. Besides that, the original version only briefly describes the tool output. Compared to the first version, we highlight the following improvements.

***(I)* the experiment now features the back-and-forth movement of the VM** (instead of a one-way movement presented in Torquato et al. (2021b)). This way, Section 3.2.2 presents an entirely new set of experimental results for the attack-and-MTD experimentation when compared to Torquato et al. (2021b). In a given deployment of time-based MTD, depending on the architecture, it is probable that the resulting scenario after the MTD action will repeat over time. In these circumstances, assuming a VM migration-based MTD, the attacker's VM eventually will return to a previously visited host. The results of the new experiment clarify how the system reacts to a second wave of attack. Besides that, they also support verifying whether the attack effects remain during continuous MTD actions.

***(II)* we extend *PyMTDEvaluator* with a new feature: Multi-Criteria Decision-Making (MCDM) to aid the selection of MTD scheduling alternatives.** *PyMTDEvaluator* allows the comparison of multiple scheduling MTD alternatives. The output has four main metrics: system availability, system security in terms of probability of attack success, system capacity to measure the impact on the service quality during the attack-defense scenario, and monetary cost due to accumulated MTD actions over time. The selection of a specific MTD schedule may vary depending on the importance that the decision-maker assigns to each one of these metrics. Therefore, in an attempt to provide a more well-rounded solution, *PyMTDEvaluator* now features two MCDM methods: Weighted Sum and Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS). Through the MCDM methods, the user receives a ranking of the evaluated alternatives based on the selected metrics weight configuration.

***(III) PyMTDEvaluator* now features multiple user interfaces.** The tool now provides three interface options for the users. *Classical* — which reproduces the original design of *PyMTDEvaluator*. *Modern* — improved aesthetics, tooltips to explain each one of the parameters, and options to save and load parameters from a file. *Upload XML file* — option to conduct evaluations directly from a saved parameter configuration file. This last option facilitates multiple evaluations of previously studied scenarios. More details of the interfaces are in the *PyMTDEvaluator* manual available at https://github.com/matheustor4/PyMTDEvaluator2/blob/main/GET_STARTED.md These extensions improve the previous version *PyMTDEvaluator* and add to the state-of-the-art as these methods are missing from the specific context of VM migration-based MTD. Finally, different from the previous paper (Torquato et al., 2021b), this paper also presents an extensive set of results obtained from *PyMTDEvaluator* output. Hopefully, the presented scenario sheds light on how to interpret and analyze the tool results.

The remainder of this paper is as follows. Section 2 presents the basic concepts of Stochastic Petri Nets. Section 3 delimits the scope of the considered threat and defense model. It also presents empirical evidence of the effectiveness of time-based VM migration as MTD. Section 4 presents the proposed SPN model. Section 5 showcase *PyMTDEvaluator*, a tool built around the proposed model to facilitate the evaluation process. Section 6 proposes an illustrative case study providing model analysis and results for hypothetical scenarios. Section 7 discusses the limitations and threats to validity. Section 8 compares the proposed approach with related works. Section 9 presents the conclusions and future works.

2. Stochastic Petri Nets

Stochastic Petri Net (SPN) is a subtype of Petri Net. SPN extends Petri Nets by including the notion of timed transitions with exponentially distributed firing delays (Marsan et al., 1998). Specifically, we use the extended deterministic SPNs, which allow guard functions, inhibitor arcs, immediate transitions, and timed transitions with deterministic delays. TimeNET (Zimmermann, 2017), Mercury (Silva et al., 2015) and SPNP (Ciardo et al., 1989) tools provide support for SPN design and analysis.

We present the SPN basics through the example in Fig. 1. For further reading, we recommend (Marsan et al., 1998; Trivedi and Bobbio, 2017; Maciel, 2023).

Fig. 1 presents an SPN availability model. The net has two places (represented by circles): P_{up} and P_{dw}; two transitions (represented by rectangles): T_{fail} and T_{repair}; one token (represented by a black circle inside the places); and arcs connecting places and transitions.

Each transition has an associated firing delay to represent the delay in event occurrence. Each transition firing moves tokens from the input places to the output places. In this example, transition T_{fail} firing moves the token from place P_{up} to place P_{dw}. And, transition T_{repair} firing moves the token from place P_{dw} to place P_{up}.

This net has only two states. The state at the top of Fig. 1 represents that the system is available. The state at the bottom means that the system is unavailable. Therefore, we can compute system availability by observing the probability of token presence in place P_{up}.

[2] A list here: https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html.

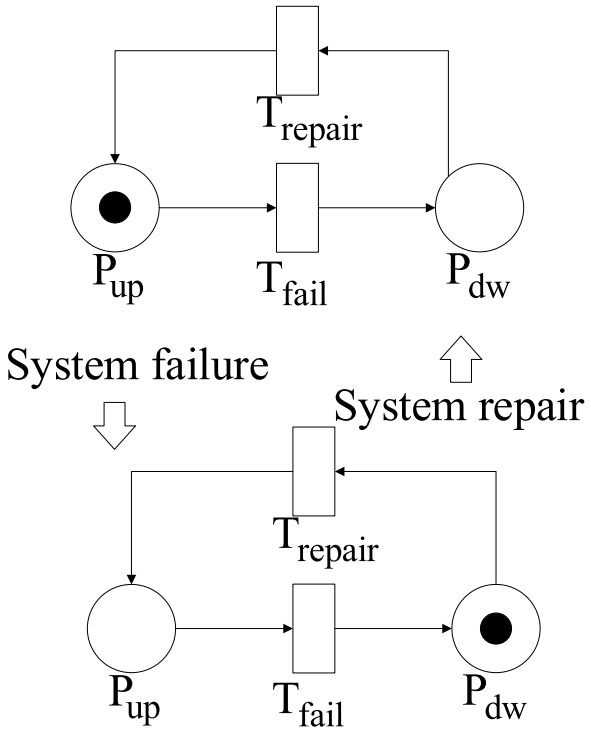


Fig. 1. An example of SPN model.

In this paper, we also applied four more concepts of the extended deterministic SPNs: (1) Guard functions — express additional conditions for transitions firing in the form of Boolean expressions; (2) Immediate transitions (represented by thin black rectangles) — transitions with associated delay equal to zero; (3) Deterministic transition (represented by a thick black rectangle) — transition with deterministic delay (instead of exponential); and (4) Inhibitor arc (represented by arcs terminating in a circle instead of an arrowhead) — which enables transition firing in the absence (instead of presence) of tokens in their input places.

3. Problem definition

This section presents the considered threat model (Section 3.1) and the obtained results from the experimental campaign (Section 3.2). Section 3.2 has three subsections: Section 3.2.1, which presents a preliminary experiment to verify the severity of the proposed security threat, Section 3.2.2 presenting the results from the attack and MTD experiment, and Section 3.2.3 which provides a preliminary root cause analysis an observed permanent failure in the *TeaStore* service.

3.1. Attack and defense model

The attacker controls one VM (*Attacker VM*) in the environment. Leveraging the VM access, the attacker launches attacks against the underlying physical host (i.e., host-based attack). The attacker aims to abuse the shared resources to disturb the *Victim VM* (Zhang and Lee, 2017). One example of an availability-oriented host-based attack is the Memory Denial of Service (*MemDoS*) attack. Roughly, *MemDoS* aims to evict the cache content to increase misses from other VMs. An alternative approach of *MemDoS* is to overload the system with memory bus LOCK signals to make the bus unavailable to other VMs (Zhang et al., 2017). The effect of *MemDoS* in the system is similar to software aging effects (Macêdo et al., 2010; Matias et al., 2012). The main difference lies in the mechanism used. While software aging is mainly due to memory leak problems (i.e., problems in the memory allocation),

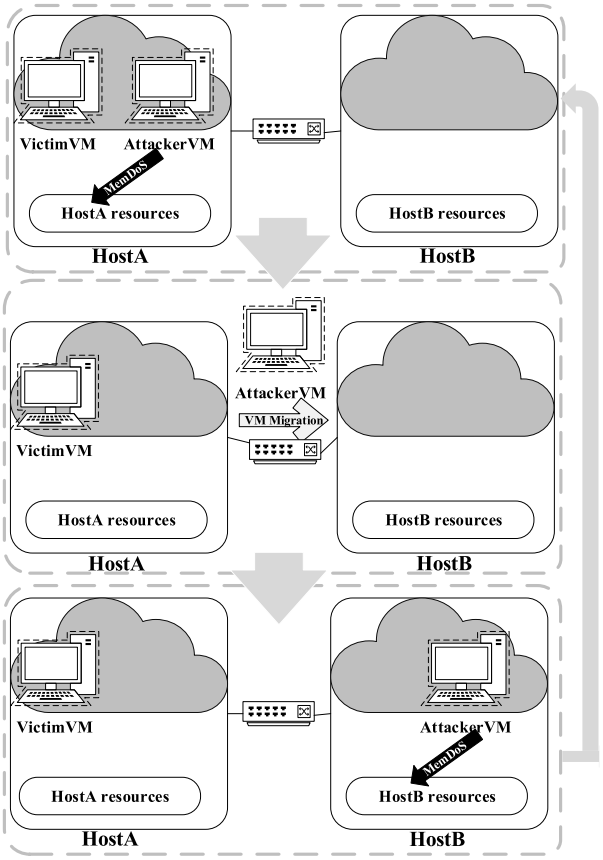


Fig. 2. Threat and defense model.

the mechanics of *MemDoS* induces more cache misses, as the bus is overloaded with LOCK signals.

The proposed MTD consists of time-based *Attacker VM* migration. The system periodically changes the position of the *Attacker VM* to increase the *Victim VM's* resistance to the incoming attack. Fig. 2 depicts the threat and defense model in a system with two hosts. In the two-host MTD setup, the *Attacker VM* ends up returning to the previous host in the second migration round. This way, the system returns to the original state before the first migration. It may be counterintuitive, as the *Attacker VM* returns sharing the same host as the *Victim VM*. However, it is worth noting that the critical factor in the considered threat model is the attack continuity. The main goal of the MTD is to increase resistance through timely VM migrations. In this scenario, the migrations break the attack continuity, thus possibly retarding *MemDoS* attack success as the system recovers itself after each migration. In the considered threat model, VM migration is not able to cease the attack (i.e., the attacker keeps attacking until it reaches success regardless of VM migration). Therefore, unless we have very frequent migrations (which may impair system performance and dependability), the attacker will eventually compromise the system.

One critical point of the MTD methodology is the *Attacker VM* selection (i.e., detection) method. We highlight that the *Attacker VM* selection is similar to a process of intrusion detection. The goal of this selection is to flag a specific VM with the status of *MTD-target*. Then applying the proposed MTD scheduling for the *MTD-target*. Since there is extensive literature on the topic of intrusion detection (Bace, 2000), we leave it to the manager to decide which method is better for each scenario. Therefore, the VM selection method is outside this paper's scope. We can highlight two methods to aid the *Attacker VM* selection. (1) Classical intrusion detection methods — verify VM resource consumption levels and flag suspicious ones for MTD deployment. (2)

Methods based on expected VM runtime — in the case of *job-oriented* VMs (i.e., VMs with an expected runtime created only to execute specific jobs and then terminated), it is possible to analyze VMs timeline. In this scenario, we can select VMs with longer execution times for MTD deployment. The idea is to avoid longer periods of potential attacker presence in the same physical host. Nevertheless, applying the proposed MTD method without attacker detection through *cyclic reconfiguration* of VMs position is possible. The *cyclic reconfiguration* of VMs assures that the resulting set of VMs inside a host is different after each migration round. We added more details about *cyclic reconfiguration* in Section 7.

In the scope of this experiment, we narrow the MTD deployment to act against *MemDoS* attacks. However, the same approach could be useful for other host-based threat scenarios. Patil and Modi's (2019) paper presents an interesting mapping of possible threats in virtualized scenarios. From their study, it is possible to notice the following: in scenarios with multiple hypervisor variants across the available physical machines, the scheduled VM migrations potentially defend against *VM-to-hypervisor* attacks as *VM escape, hypervisor information leak, or code execution on hypervisor*. Additionally, assuming that each migration round results in a different set of VMs inside the same physical host, the same strategy potentially defends against *VM side channel, VM covert channel, or RowHammer attacks.* This list is non-exhaustive, and potentially, there are other threats sensitive to VM migration-based MTD.

3.2. Experiments

Our first step is to evaluate the effectiveness of time-based MTD against *availability attacks*. We conducted an experiment to obtain knowledge about the attack-defense flow behavior in the time-based MTD context. Such knowledge supported the design of the SPN model (see Section 4).

To reach our goal, we conducted an *availability attack* campaign in a real testbed. This *availability attack* campaign has two experiments:

- **Attack severity experiment** — to investigate the *availability attack* impact in a system without MTD. For comparison purposes, we added **Golden run** results. *Golden run* observes the system baseline behavior (i.e., without attack and MTD).
- **MTD experiment** Aims to investigate the system behavior with time-based MTD enabled. We applied different scheduling policies of MTD against an ongoing *availability attack*. Also, for comparison purposes, we added the results for the system without MTD (*OnlyAttack*).

The experiment details and results are in the following sections. Although the results below refer to a single run of each experiment, we performed additional runs to confirm the results. At the time of this paper writing, we ran ten runs of *attack severity experiment* and ten runs of attacks in an MTD-enabled environment.

3.2.1. Attack severity experiment

To investigate the system behavior under an *availability attack*, we exercise our testbed with a *MemDoS* from an insider Virtual Machine (VM). We aim to observe the impact of the attack in a co-resident VM (i.e., victim) running a standard client–server application. The experiment testbed details are below.

Fig. 3 presents the experimental testbed, which includes three physical machines: Stresser (Intel i5 8250U + 16 GB of RAM) — responsible for stressing the Victim's VM; Source Node (Intel Xeon E5-2620 2.00 GHz + 16 GB of RAM with Error Correction Code enabled) — main host of the Victim's VM and Attacker's VM; Target Node (Intel Core i7-9700 3.00 GHz + 16 GB of RAM) — host for receiving VM migration.

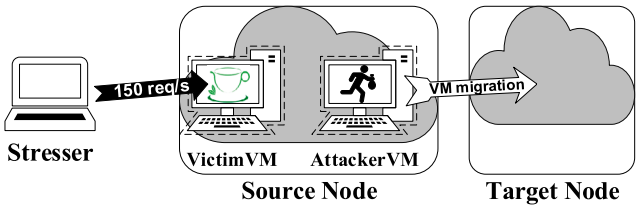


Fig. 3. Experiment testbed.

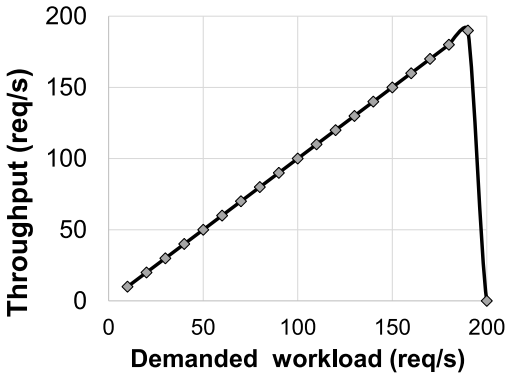


Fig. 4. Victim's VM TeaStore capacity assessment results.

Attacker's VM and Victim's VM are Kernel Virtual Machine (KVM)[3] VMs with a homogeneous configuration: single-core processor + 3 GB of RAM. The Source Node and Target Node run Ubuntu Server 20.04.2 with kernel 5.4.0–72 and KVM 4.2.1.

The Victim's VM runs the TeaStore microservice architecture (Von Kistowski et al., 2018), which emulates a basic web store. Our first step in the *attack severity experiment* is to define a proper workload for Victim's VM. Using the httperf tool (Mosberger and Jin, 1998), we conducted a Victim's VM capacity assessment. Fig. 4 presents the results, where the X-axis corresponds to the demanded request rate, and the *Y*-axis corresponds to the system reply rate.

As 190 requests per second (req/s) is the threshold for a performance drop, we selected a lighter workload to provide room for expected performance oscillations due to the attacks. Thus, we arbitrarily configured the Stresser to send a steady workload of 150 req/s to Victim's VM.

With knowledge about Victim's VM capacity, we can set up the attack load of the *attack severity experiment*. As mentioned earlier, the specific *availability attack* is a *MemDoS* (Zhang et al., 2017) attack from the Attacker's VM running inside the Source Node. In practice, the Attacker's VM runs an infinite loop of unaligned atomic accesses (Zhang et al., 2017) (*unalignAttk*) against its memory. We then analyze the possible impact of *unalignAttk* in the Victim's VM TeaStore service.

We performed the *attack severity experiment* for 30 min. The *golden run* results are included for comparison. Fig. 5 presents the TeaStore throughput results, and Fig. 6 presents the number of observed errors. The errors are related to connection timeout (i.e., the server is offline).

We can see an immediate drop in the throughput when the Attacker's VM is running the *unalignAttk*. After five minutes, we notice a substantial increase in observed errors, suggesting server unavailability. Therefore, we can conclude that after five minutes, the *unalignAttk* successfully compromises the TeaStore availability.

As one may argue that this is expected behavior because of the *unalignAttk* resources consumption overhead, we performed an experiment run with a benign heavy workload, namely an infinite loop of

[3] https://www.linux-kvm.org/.

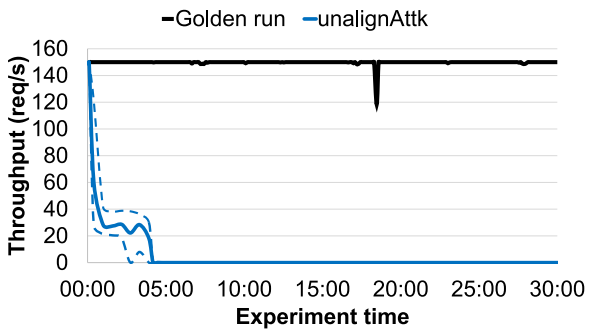


Fig. 5. TeaStore - Throughput (req/s) - attack severity experiment.

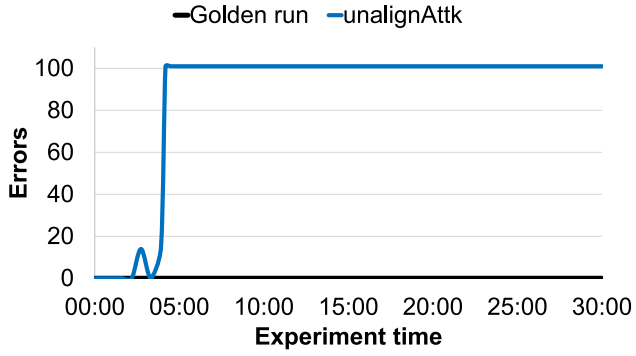


Fig. 6. TeaStore - Errors - attack severity experiment.

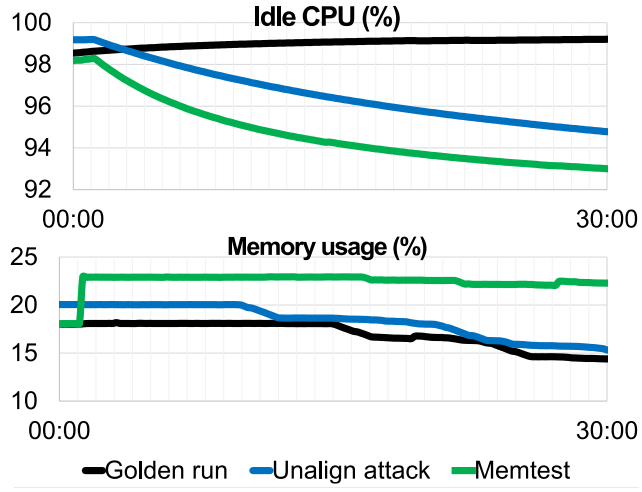


Fig. 7. Comparison of *Golden run*, *memtest* and *unalignAttk* - Source Node resources.

Linux *memtest*.[4] In this experiment, the Attacker's VM was running *memtest* instead of *unalignAttk*. Figs. 7 and 8 present a brief comparison of *memtest*, *unalignAttk* and *Golden run*. The plots on Fig. 7 refer to the Source Node resources consumption — Idle CPU and Memory usage. The plots on Fig. 8 refer to TeaStore service results — Throughput and Errors.

The results in Fig. 8 show that even using more resources than *unalignAttk*, the *memtester* infinite loop cannot compromise TeaStore availability, causing only throughput drop without error accumulation. These results highlight that the *MemDos* based on *unalignAttk* should be considered a security threat because it is capable of causing co-resident VM unavailability.

[4] https://linux.die.net/man/8/memtester.

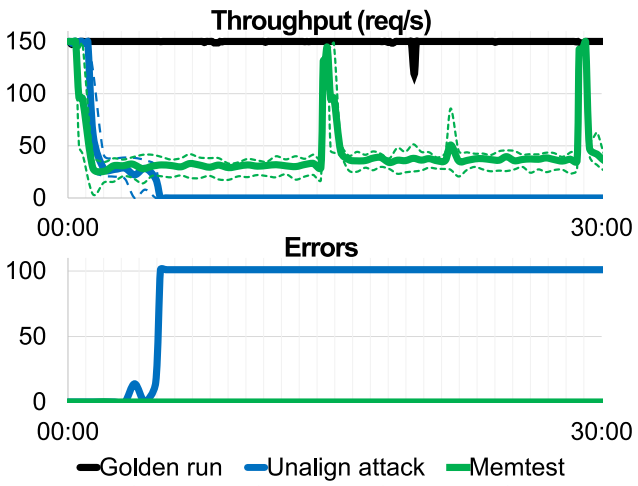


Fig. 8. Comparison of *Golden run*, *memtest* and *unalignAttk* — TeaStore metrics.

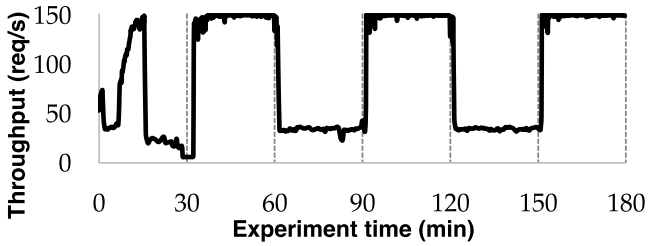
3.2.2. MTD experiment

We instantiate the attack and defense model in a real testbed. The goal of this experiment is to verify the validity of the proposed approach. This experimentation is a continuation of the previous works (Torquato and Vieira, 2021; Torquato et al., 2021b). During these last works, we neglected the effect of a series of VM migrations in the environment, as they considered only a one-way migration in each experiment. In this paper, we studied the back-and-forth movement of the attacker's VM. Remember that our experiment only comprises two physical hosts. Therefore, considering one Victim's VM and the *cyclic reconfiguration* (i.e., the VM's neighborhood changes after each migration triggering), the cycle only has two possibilities: under attack and without attack.

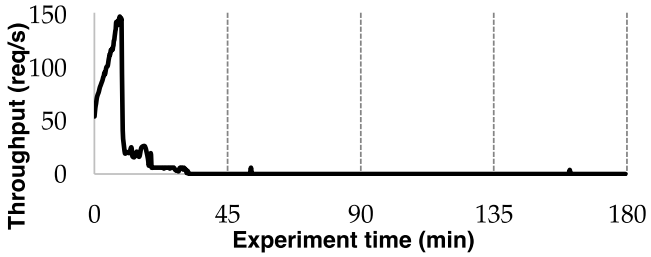
We consider four scenarios in the experimentation: 30 min between migrations, 45 min between migrations, and 60 min between migrations and the system without MTD. Here, we observe the TeaStore server throughput while receiving a constant workload of 150 req/s. The results are in Fig. 9. Each MTD-enabled experiment plot presents vertical lines highlighting the VM migration trigger event. Although the presented results come from a single experiment run, we performed confirmation runs (at least two per scenario) to verify the results. The obtained results in all the experiment runs are similar. Note that the scheduling adopted here differs from our previous paper (Torquato et al., 2021b) as the experiment goals differ. In Torquato et al. (2021b), we aimed to verify the effectiveness of VM migration as MTD. Therefore, quick one-way migrations were enough. Now, the experiment goal is to verify the validity of using the scheduling of VM migrations. The experiment also investigates whether there are residual effects of accumulating VM migrations over time.

Whenever possible, we aim to reduce the system workload exposure. The goal is to avoid excessive resource consumption. So, we intend to find the larger interval between migrations that is able to keep the system alive during the attack. We decided to increase the number of VM migrations gradually. This way, we arbitrarily start the experiment with 60 min between migrations with a 15-min decrease step. The 30-min scheduling presented the desired behavior.

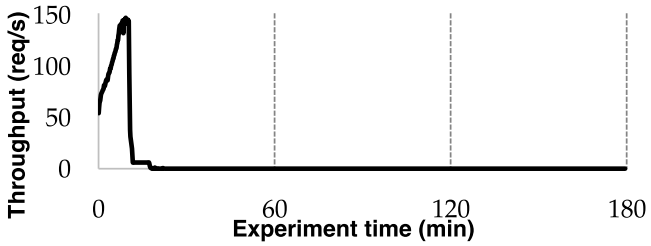
First, we highlight that the attack effects only take place after the TeaStore warm-up phase. In some preliminary runs, we noticed that early attack triggering might cause premature failures. Therefore, we first start the services. Then, after the services launch completion, we start the webserver workload. We wait for the system's expected throughput rate (i.e., 150 req/s) before the attack initiation. In the system architecture we used for the experiment, the average time to reach a throughput of 150 req/s is 15 min.. In all scenarios, we noticed



(a) VM migration schedule = 30 minutes



(b) VM migration schedule = 45 minutes



(c) VM migration schedule = 60 minutes

(d) System without MTD

Fig. 9. Experimental results — VM migration scheduling as MTD.

an immediate and severe drop in the throughput upon the attack triggering.

Fig. 9(a) confirms the theoretical expected result — a squared wave of throughput based on attack presence or absence. In the intervals of attack absence, the service maintains the expected throughput. On the other hand, in the intervals of attack presence, the service delivers degraded throughput. However, the results for the less frequent migrations (Figs. 9(b) and 9(c)) contradict the theoretical expected result. In these cases, the system does not recover its expected throughput even with VM migration. These results are comparable to the system without MTD (Fig. 9(d)). The reason for this persistent failure is presumably due to a crash after resource exhaustion. In an attempt to clarify this behavior, Section 3.2.3 presents a preliminary root cause analysis.

We highlight the following conclusions: (1) Timing plays a crucial role in the VM migration-based MTD effectiveness. (2) In some scenarios, as in our experiment, the system does not recover even after MTD

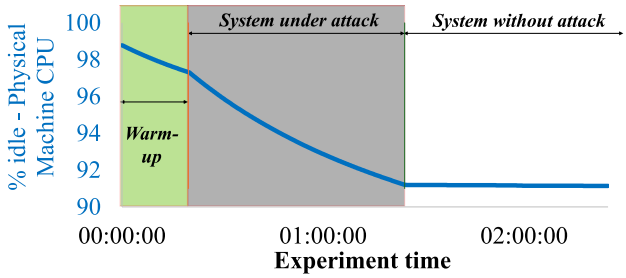


Fig. 10. Physical machine CPU — % idle.

deployment. (3) There is a specific schedule that minimizes the VM migration frequency while keeping the system alive during the attack. In our case, this specific schedule is between 30 and 45 min of VM migration interval. However, we neglect the search for this optimal MTD schedule through experimentation. We present the step-by-step guidance to find the optimal scheduling using *PyMTDEvaluator* tool in Section 6.8.

3.2.3. Preliminary root cause analysis for TeaStore permanent failure

The results presented below come from a series of ten additional runs of the MTD experiment with migration at 60 min. All the experiments last about two hours and fifteen minutes. The first fifteen minutes represent the *warm-up* phase of the service. The remaining two hours are the period of the attack-defense scenario. The flow is as follows. First, one hour of attack, then *Attacker VM* migration, concluding with one hour of the system without attack. The plots below highlight the experiment phases — green box for the *warm-up* phase, gray box for the attack period, and white box for the attack-less period. As the results from all the experiments were similar, the results below are from one of them.

We decided to keep the root cause analysis below simple to avoid deviating too much from the paper's main focus. Therefore, a deeper analysis is required to clarify the nuances of the failure. Nevertheless, the obtained results are enough to shed light on the possible causes of the permanent failure behavior. The primary guideline for our analysis comes from the root cause analysis principle of *"a structured investigation that aims to identify the real cause of a problem and the actions necessary to eliminate it"*. Andersen and Fagerhaug (2006).

The proposed root cause analysis follows a hierarchical approach, starting from the analysis of the physical machine resources, followed by an analysis of the *Victim VM*, concluding with an analysis of the *TeaStore* service running inside the VM. We also outline the strategies that were trialed to resolve the problem.

In summary, the results below show that the **TeaStore service goes down after failure due to the Denial of Service attack**. Simple restart of the containers or memory cache cleanup actions were not enough to recover the service. In our case, the working **solution was to completely reboot the *Victim VM*, start the services in an isolated (i.e., without web load exposure) environment, and only then expose the system to the web server load.** We present a more detailed analysis below.

Physical machine analysis. The goal of this analysis is to answer the question: "What happens to the physical machine resource consumption while the system presents the observed persistent failure?". We conduct system resource monitoring during the attack and defense experiment. The result from the CPU usage is in Fig. 10. The results from memory usage are in Fig. 11. The considered machine here is the Source Node (Intel Xeon E5-2620 2.00 GHz + 16 GB of RAM with Error Correction Code enabled).

The CPU usage is heavier during the attack. We noticed a steeper decay in the idle curve in the *system under attack* phase (gray box in

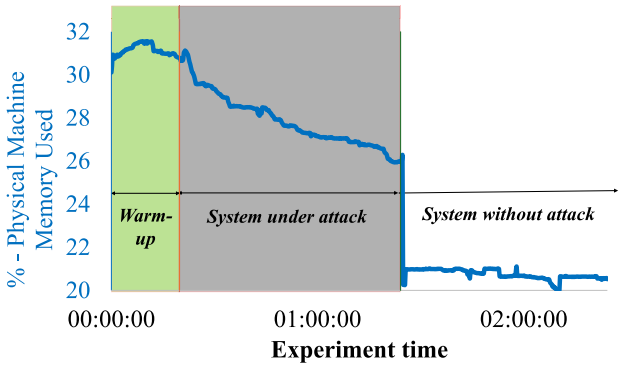


Fig. 11. Physical machine % memory used.

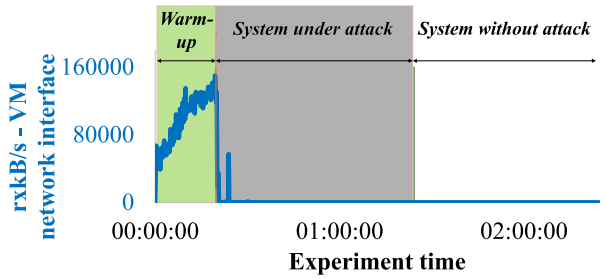


Fig. 12. VM network interface — received network traffic in kB/s.

the plot of Fig. 10). Nevertheless, the system is also freeing memory space during the same period (gray box of the plot in Fig. 11). In general, the results of the physical machine show that the system can run the attack load without main problems. During the attack phase, the average idle percentage of the CPU is 94% of the time, while the memory usage average is 28%. Although the effects on resource consumption are considerable, they are way below critical levels.

The interesting behavior in the physical machine results is the one after *Attacker VM* departure (i.e., *system without attack*) phase. We notice that the produced residual effects. After the attack ceases, it is expected that the system will recover. In other words, we expect the CPU idle curve to increase and the memory usage curve to drop right after migration. Instead, the system stays steadily in a degraded (i.e., worse than the optimal/fresh) state after the attack. This behavior is similar to software aging accumulation (Torquato and Vieira, 2019). However, further investigation is required to confirm that these effects will last for longer periods.

Victim VM analysis. Since the physical machine monitoring results show that the machine has plenty of resources after the attack, the hypothesis of generalized Denial of Service is out of the question. Therefore, we need to go further into the evaluation. The second level of evaluation is to verify the *Victim's VM* resource usage. The results of CPU and RAM monitoring show that the VM is under a heavy workload during the experiment. However, the most interesting result comes from monitoring the network interface. Fig. 12 presents the received network traffic in the docker interface.

Network traffic results show an immediate effect of the attack. The network traffic promptly reaches zero right after the attack. The result must be carefully analyzed as it may mislead to the conclusion that the VM suffered a complete crash. However, this was not the case. A secure shell connection was alive even after the service interruption. Docker software failure is also discarded as the docker daemon was responsive after the attack. Therefore, the specific affected component discovery required further investigation.

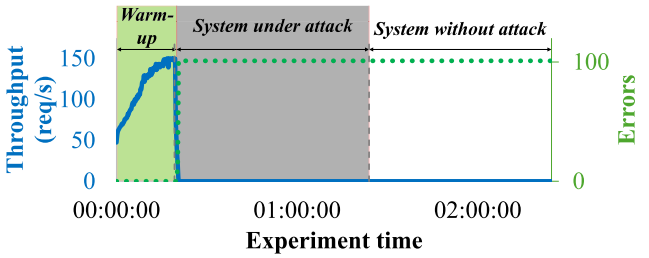


Fig. 13. TeaStore web service — Throughput (req/s) and errors.

TeaStore service analysis. In the final phase of the root cause analysis, we investigate the TeaStore microservice architecture. First, we collect the results from the webload exposure (Fig. 13). The results are similar to the previous results from the Docker network interface. The plot in Fig. 13 also includes a secondary y-axis showing the number of errors. The results show errors skyrocketing after the attack launch.

A more deep inspection shows that all the microservices in TeaStore, namely *WebUI, Auth, Persistence, Recommender, and Image* are running during the *warm-up* phase. Once the attack started, the services went down. *WebUI* service resists for a little longer, only failing after three minutes of attack.

One important question arises from this data. *If all the services went down right after attack launch, how was the VM migration schedule presented in Fig. 9(a) was able to preserve system availability?* The answer to this question comes from a log analysis of TeaStore-related containers. Log analysis reveals that, despite the immediate impact, the services survive for a while. The services keep trying to connect even with continuous `timeout` errors. Therefore, we infer that a VM migration during this survival time brings the system back to normal levels.

Failure recovery. The recovery process of this section aims to bring the system back to activity after the *Attacker VM* migration. Therefore, the process here was executed while the system was without attack and under a web server workload. First, we try to remove the software aging-like status with memory cache cleanup. The cleanup alleviates memory usage in the VM. However, the TeaStore stayed unavailable. Then, we completely restarted the TeaStore docker containers. The system returned to availability with poor levels of service quality (the throughput levels were 15 req/s instead of 150). We infer that, as the webserver workload persisted during the entire experiment, the containers had problems in the services startup. Finally, the complete recovery only comes after a full reboot in an isolated environment (i.e., without attack and web server workload). Then, after services startup, the system can handle the workload of 150 requests per second again.

4. Model

The model presented here comes from our previous paper (Torquato et al., 2021b). As this paper focuses on presenting new experimental results and the improved features of the tool, the model remains unaltered. Note that, as the model is the core of the simulation module of *PyMTDEvaluator*, any modification in its design will require proper reimplementation. Although the reimplementation process is straightforward (i.e., including the new variables to represent the new transitions), it should be performed with caution as the variables related to the current model design must be updated accordingly. In that scenario, we will also need to run a new round of tool validation to ensure its correctness. An improved version of the current model is in our plans for the near future.

Fig. 14 presents the proposed model, which has two submodels: (a) SCHEDULE MODEL and (b) MTD MODEL. The former represents the behavior

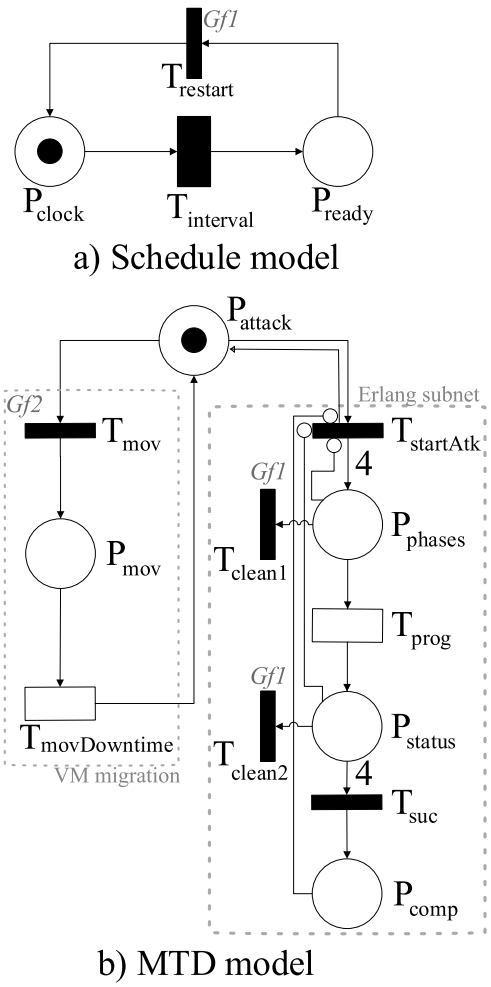


Fig. 14. Stochastic Petri Net model.

of the system component in charge of triggering VM migration, and the latter comprises the attack and MTD effect. The model embeds two guard functions[5]: $Gf1 = \#(P_{mov}) > 0$ and $Gf2 = \#(P_{ready}) > 0$. The guard functions define additional conditions for transition firing in the form of boolean expressions.

Schedule model has a deterministic timed transition — $T_{interval}$ to represent the fixed time between VM migrations. Transition $T_{interval}$ firing moves the token from place P_{clock} to place P_{ready}. Then, the model activates $Gf2$, a guard function that provokes the VM migration effects in the MTD model. The Clock model ends its cycle through $T_{restart}$ firing, representing the clock restart after the triggering of VM migration.

MTD model applies an Increasing Failure Rate distribution to represent the increasing probability of attack success over time. Following the same approach of previous works (Torquato et al., 2021b,a), we use a 4-phase Erlang subnet to represent the attack progress. Distefano et al. (2020) also assumes a 4-phase attack model. The first transition to fire in the Erlang subnet is the immediate transition $T_{startAtk}$. Its firing puts four tokens in place P_{places}, representing the remaining phases of the attack. Transition T_{prog} applies the exclusive server semantics, meaning that its firing removes the tokens one by one from place P_{phases} to place P_{status}. Then, place P_{status} token accumulation represents the attack progress. Transition T_{suc} fires immediately after depositing the fourth token in the place P_{status}. Finally, the place P_{comp} receives a token from transition T_{suc}, representing the attack success. We use several inhibitor arcs[6] to prevent multiple firing of the Erlang subnet transitions. The inhibitor arcs enable transition firing upon the absence of tokens in the input places (instead of token presence in the regular arcs).

From our experiment conclusions, a timely MTD deployment prevents attack success. We embed the MTD effects on the attack progress in the transitions T_{clean1} and T_{clean2}. These transitions remove all the tokens from places P_{phases} and P_{status} upon the VM migration triggering event.

VM migration-based MTD also impacts the system availability, as each VM migration has an associated downtime (Clark et al., 2005). The transition $T_{movDowntime}$ has an associated delay representing the system interruption during VM migration. Note that this transition firing depends on the token presence in the place P_{mov}. Place P_{mov} receives the token after $Gf2$ activation (i.e., clock reaches the interval for VM migration triggering).

Two primary metrics of interest are steady-state availability (A) and transient probability of attack success (PAS). We compute A observing the steady-state probability of token presence in the place P_{attack}. We compute PAS through the transient probability of token presence in place P_{comp}. We also derive the system capacity by observing the place P_{phases}. Specifically, each token presence in the place P_{phases} sums up 25% in system capacity.

5. Model simulation environment

This section focuses on the details of *PyMTDEvaluator* tool. It has two subsections: Section 5.1 introduces the tool implementation and features. Section 5.2 compares the tool and model results for validation purposes.

5.1. PyMTDEvaluator tool

In this section, we present *PyMTDEvaluator* tool. *PyMTDEvaluator* was first presented in the paper (Torquato et al., 2021b). This explanation suppresses implementation details from the original paper to avoid excessive overlap. In summary, we use the SimPy[7] framework to implement the model presented at Section 4. *PyMTDEvaluator* tool provides a graphical interface for the proposed model. This way, the users can run several evaluations to compare scenarios with different MTD policies. The main goal is to facilitate the evaluation process while providing useful features to help the analysis.

The tool's `Modern` interface option is in Fig. 15. Besides the `modern` interface, the user can also select the `classical` — same aesthetics as the previous version of *PyMTDEvaluator* (Torquato et al., 2021b) or the `Upload XML file` interface, in which the user feeds the parameters using an XML file instead of the graphical interface. More details about the user interfaces are in the *PyMTDEvaluator* manual available at https://github.com/matheustor4/PyMTDEvaluator2.

From the implementation perspective, *PyMTDEvaluator* uses Python language along specific libraries as `simpy` (Scherfke et al., 2020) for the simulation environment, `scikit-criteria` (Cabral et al., 2016) for the Multi-Criteria Decision Making methods, `matplotlib` (Tosi, 2009) for plot generation, `tkinter` (Shipman, 2013) for the user graphical interface, and `reportlab` for PDF report generation.[8] *PyMTDEvaluator* source code in the following link.[9] For the user's convenience, *PyMTDEvaluator* is also available in a Docker[10] container.

The use case scenario is as follows. The users fill the interface fields with the parameter values of their interest. `Downtime per movement (min)` — expected downtime due to each MTD action;

[5] $\#(P_{place})$ refers to the number of tokens in the place P_{place}.

[6] Arcs terminating in a circle instead of an arrowhead.

[7] https://simpy.readthedocs.io/en/latest/.

[8] https://pypi.org/project/reportlab/.

[9] https://github.com/matheustor4/PyMTDEvaluator2.

[10] https://www.docker.com/.

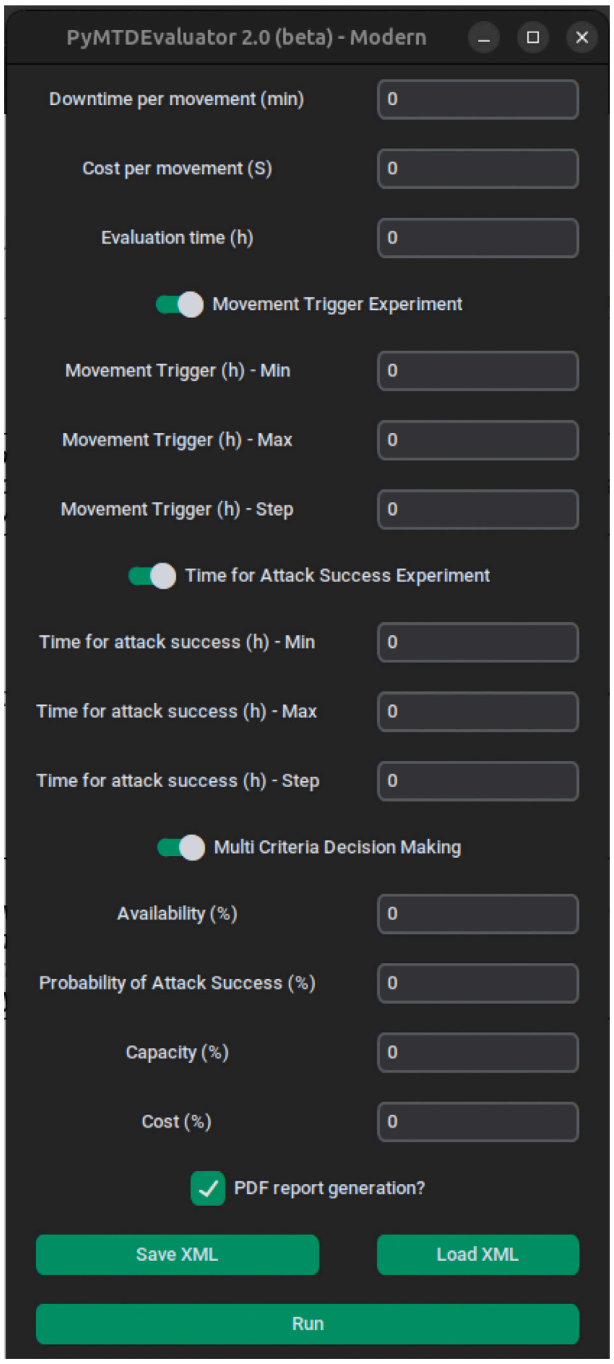


Fig. 15. *PyMTDEvaluator* modern interface.

`Cost per movement ($)` — monetary cost related to the deployment of each MTD action; `Movement Trigger (h)` — time between MTD actions deployment (i.e., in our case study — VM migration schedule); `Time for attack success (h)` — expected time for attack success; `Evaluation time (h)` — Time target for the evaluation.

PyMTDEvaluator also supports the multiple scenario evaluation through the feature Experiment. Experiment feature automates the evaluation process while considering variation for the parameters `Movement Trigger` and `Time to attack success`. Besides that, it is also possible to compile the evaluation results in a PDF file. Besides the availability and security metrics, *PyMTDEvaluator* also incorporates the concept of *System Capacity*. *System Capacity* relates to the system's ability to deliver the service while under attack and MTD. *System Capacity* is obtained by observing the token distribution in places related to the attack progress. This metric is helpful, for example, in understanding Quality of Service (QoS) degradation due to the availability attack. Our previous paper (Torquato et al., 2021b) presents more details of *PyMTDEvaluator* implementation.

Extending our previous work (Torquato et al., 2021b), *PyMTDEvaluator* now also features a Multi-Criteria Decision Making module. The user can input the desired weight for each of the metrics of interest. After the evaluation, *PyMTDEvaluator* will rank the evaluated policies. The feature is helpful in aiding the selection of the MTD schedule based on the needs of each environment. Following the previous works in the field (Torquato et al., 2019; Araujo et al., 2018), we implement the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) approach for the Multi-Criteria decision-making. The tool output also contains the Weighted Sum Model (Singh and Malik, 2014) as a secondary Multi-Criteria Decision Making option.

The full details of the *PyMTDEvaluator* output are in Section 6, where we exercise the tool for a hypothetical scenario.

5.2. PyMTDEvaluator verification against model results

To verify the *PyMTDEvaluator* results and assure its correctness, we compared the TimeNET tool (Zimmermann, 2017) analysis results with the *PyMTDEvaluator* simulation results (see Fig. 16). This approach is similar to the verification proposed in Bai et al. (2023c,a,b).

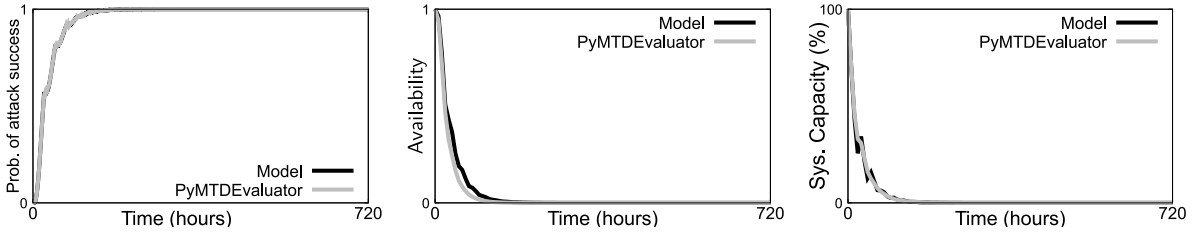
The comparison's main parameters are four seconds of downtime per movement, 24 h between MTD movements, and 24 h as the expected time for attack success. The results consider the first month of the system under attack (i.e., `Evaluation time` = 720 h). Note that these parameters are just for validation and may not represent a real-scenario situation. Furthermore, it is essential to highlight that we have validated the tool using many other parameter sets. These results are omitted due to space limitations. However, in all of them, the *PyMTDEvaluator* results are very close to the model results.

Fig. 16 shows that the results from *PyMTDEvaluator* are nearly the same as the model analysis results from TimeNET. The black line represents the model results, and the gray lines represent the *PyMTDEvaluator* results. The model results of the probability of attack success (Fig. 16(a)) and system capacity (Fig. 16(c)) are compatible with the *PyMTDEvaluator* results. The availability results are also very close but present a slight variance in the curve's points. The difference in the availability curves may be due to the simulation stoppage method when the system reaches the *availability attack* success (as mentioned at the end of Section 4). We had to implement this stoppage method in the *Transient evaluator* module to reduce *PyMTDEvaluator* computing time.

6. Case study

To answer our main research question (*What is the impact of the variation of MTD scheduling in the system availability and probability of attack success?*), this section exercises *PyMTDEvaluator* tool performing a sensitivity analysis (i.e., Experiment feature of *PyMTDEvaluator*) of the VM migration interval parameter (i.e., Movement trigger parameter of the *PyMTDEvaluator* interface). In the figures of the following sections, we marked the captions of all the plots obtained directly from *PyMTDEvaluator* with the prefix "*PyMTDEvaluator output*". The data presented in other plots were also obtained from *PyMTDEvaluator*. However, we use different tools to produce them.

Table 1 presents the default values used for our evaluations. We obtain these values from the recent papers (Torquato et al., 2020; Chen et al., 2020a). Note that these values are only illustrative and should be adapted for each scenario. For comparison purposes, we select two targets for the simulation: two days and one week. We emphasize that *PyMTDEvaluator* considers that the system is under an availability attack. Therefore, the system availability is affected by both the attack itself and the accumulation of MTD actions.



(a) Probability of attack success (b) Availability (c) System Capacity (%)

Fig. 16. PyMTDEvaluator results validation against SPN model results.

Table 1
Parameters used in the evaluation.

Parameter	Description	Value
`Movement Trigger`	Time interval for VM migration	From 0.5 h to 25.5 h with a 5-h step
`Time for attack success`	Expected time for the attacker to reach success if the system is without MTD	24 h
`Downtime per movement`	Downtime per VM migration	4 s
`Cost per movement`	Monetary cost per VM migration	0.3
`Evaluation time`	Target time for the simulation run	2 days and 1 week
Multi-Criteria Decision Making weights		
Criteria		Weight (%)
Availability		30
Probability of attack success		50
Capacity		5
Cost		15

We emphasize that the considered parameters are similar to the ones in Torquato et al. (2021b). It is worth highlighting that, instead of considering different parameters, this paper aims to better explain *PyMTDEvaluator* output by providing a more detailed analysis of the obtained data.

6.1. Steady state availability

The steady-state availability results neglect the attack effects. Recalling the content from Section 3.1, we emphasize that in the studied threat model, the VM migration intends to increase the system's resistance to attacks instead of ceasing the attack. Therefore, the attacker will eventually compromise the system. That means the system will reach unavailability in a steady-state (i.e., stationary) perspective. Then, when leaving out the attack effects from the steady state availability evaluation, it reveals how much downtime is due to VM migration accumulation. Finally, these results might be helpful in measuring the impact of VM migration scheduling that is already in place. These results are not a simple product of the migration frequency and expected downtime per MTD movement. The downtime of each migration may differ because of diverse reasons, from the network state to the dirty page rate (Salfner et al., 2011). Therefore, the results come from the simulation of the model searching for the stationary state.

The availability results (see Fig. 17) reveal a significant difference between the most and least frequent VM migration policies. Specifically, daily migrations (25.5 h) cause only about 22 min of annual downtime, while migrations on a 30-min basis produce more than 18 h of downtime in a year. Another conclusion is that, from the schedule of 15.5 h between migrations to the one with 25.5, the difference is less than 15 min. In some cases, a 15-min increase in annual downtime is acceptable because of the reduction in the monetary cost from MTD movements.

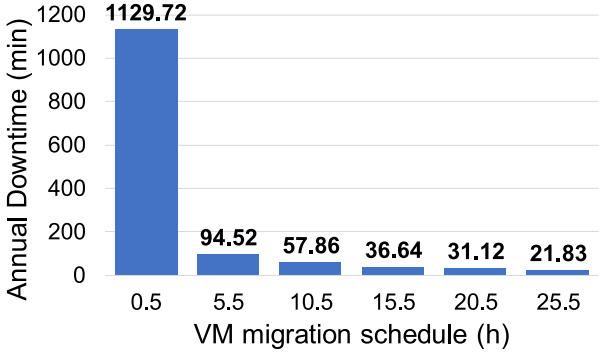


Fig. 17. Steady-state availability results.

6.2. Transient availability

Unlike the previous section, the transient availability results consider the attack effects. It reveals the expected system availability at a specific time once the attack is active in the environment. One may think of this particular metric as a *survivability* metric, as it considers both aspects: availability (i.e., downtime due to VM migration) and security (i.e., downtime due to the ongoing availability attack). However, we avoid following this path, as *survivability* may appear as a qualitative aspect in some scenarios (Liu and Trivedi, 2006; Al-Kuwaiti et al., 2009). We prefer transient availability or even *pointwise availability* (Govil, 1972), as the final goal is to analyze the availability at specific points in time (specifically, two days and one week).

Fig. 18 presents the transient availability results. *PyMTDEvaluator* output plot presents a curve for each studied MTD schedule. For the sake of readability, the tool omits the confidence intervals for each of the curves in the plot. Nevertheless, the data is available in the output files.

The policy with migrations after 30 min can keep the system available most of the first week of the attack. The simulation results for the policy 30 min do not show any significant drop in the availability curve during the evaluation. There are only a few slight drops (almost invisible) in specific curve points. They are related to the time when VM migration is triggered. A question that may arise from this evaluation is: *Why does the availability curve keep flat while we have continuous actions causing downtime (i.e., VM migration)?* The answer is that the results come from several simulation runs,[11] in each one of these, the VM migration downtime comes from a random variable. Therefore, grouping all the results, we finally notice that the VM migration accumulation does not produce noticeable changes in the availability curve while considering the first week of the attack. Although the curve does not reveal a substantial availability impact due to VM migration, the final result suggests a 99.73% availability (i.e., approximately a downtime of half an hour in a week).

[11] The exact number is 10 000.

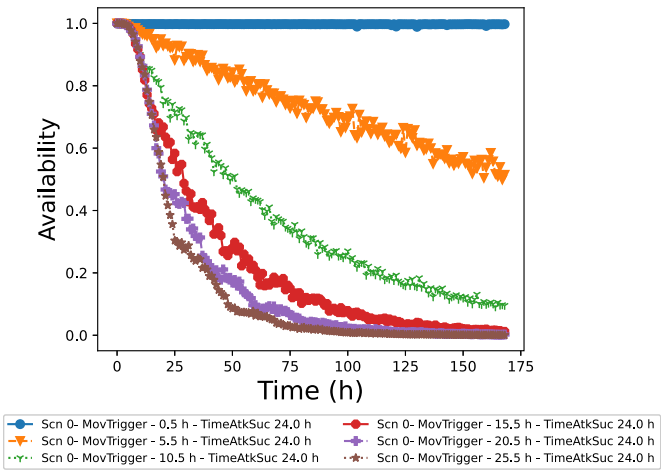


Fig. 18. PyMTDEvaluator output — Transient availability results.

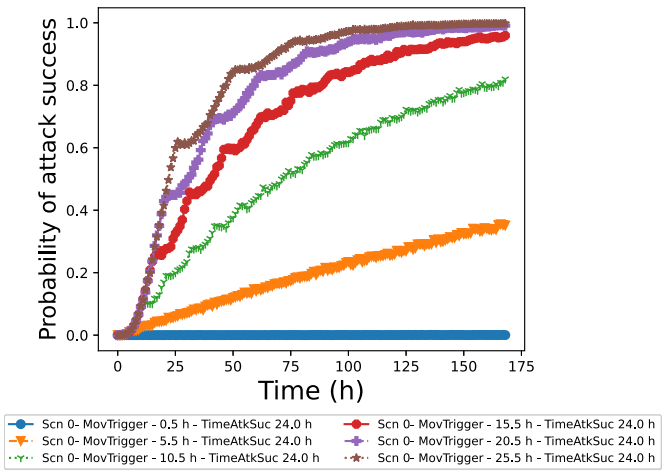
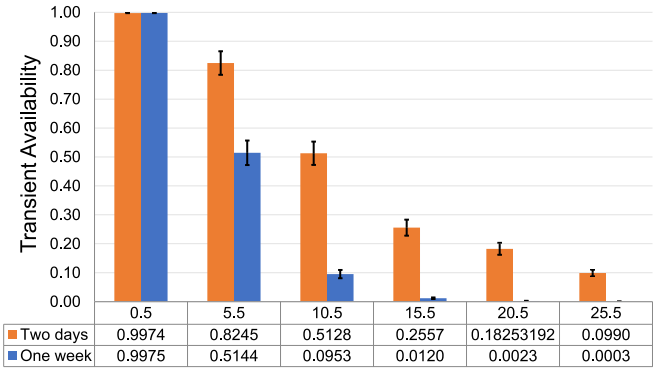
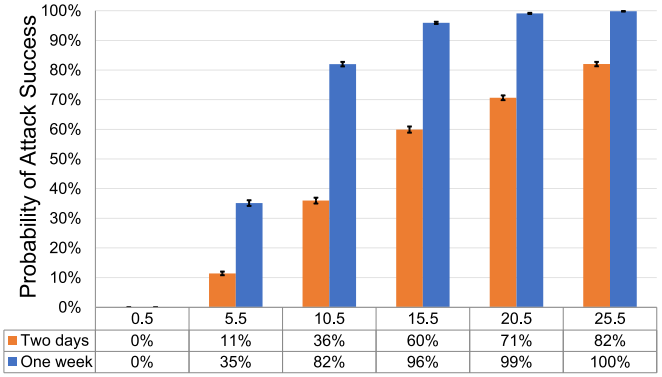


Fig. 20. PyMTDEvaluator output — Probability of attack success results.



	0.5	5.5	10.5	15.5	20.5	25.5
Two days	0.9974	0.8245	0.5128	0.2557	0.18253192	0.0990
One week	0.9975	0.5144	0.0953	0.0120	0.0023	0.0003

Fig. 19. Transient availability — comparison.



	0.5	5.5	10.5	15.5	20.5	25.5
Two days	0%	11%	36%	60%	71%	82%
One week	0%	35%	82%	96%	99%	100%

Fig. 21. Probability of attack success — comparison.

The plot also suggests that, at the specific point of one week of continuous attack, the policies with 15.5, 20.5, and 25.5 h between VM migrations produce similar availability results. In these cases, the system availability at one week (i.e., 168 h) is below 1%. As expected, these policies have the same availability results at the beginning of the curves, where the system is waiting for the first VM migration.

Fig. 19 compares the studied MTD policies considering two points in time: two days and one week. These results help detail how availability develops in the proposed scenario. The included error bars represent the 95% confidence interval.

The plot suggests the pace of availability decreases from two days to the completion of the first week. Interestingly, for the last three VM migration alternatives (15.5, 20.5, and 25.5), the system availability almost completely deteriorated from the first week of continuous attack. Besides that, considering the confidence interval, the availability results of these three policies at 168 h are similar. Therefore, considering this point in time, availability may be an irrelevant criterion while selecting these three alternatives.

6.3. Probability of attack success

For the MTD protection evaluation, we consider the probability of attack success. Specifically, the probability of attack success is the probability of the attacker successfully compromising the system in a given time. As expected, our analysis demonstrates that the more frequent migrations produce better protection results. However, careful attention must be exercised because the specific frequency is crucial to the results. The selected five-hour step shows the impact of varying the VM migration schedule on the probability of attack success results (see Fig. 20).

At first glance, the probability of attack success curves resemble upside-down variation transient availability curves (previous case study). This is because both curves deal with the availability aspect. Nevertheless, the probability of attack success results neglects the availability impact of continuous VM migration.

As in the previous case study, the last three proposed schedules (15.5, 20.5, and 25.5 h) have similar results at the beginning and the end of the simulation. The policy with the most frequent migrations (0.5 h) nullifies the probability of attack success during the first week. Finally, the comparison between 5.5 h and 10.5 h shows that the 5.5 h policy produces a result 50% better at 168 h.

Fig. 21 presents the results for two days and one week. At two days, the 25.5 h policy already reached a probability of attack success of 82%, showing that, in this case, the MTD effectiveness is poor compared to the other strategies. Besides that, the policies with migration schedule time greater than 10 h reached more than 80% of probability of attack success in one week. This means that they may not be suitable for preventing system attack success at this point in time. A possible approach to analyzing these results is from a tolerance-level viewpoint. For example, if the system (or the workload) is expected to run for two days, and it is possible to tolerate about 15% of attack success, the policy of 5.5 h could be preferred among the others.

6.4. Cost

Fig. 22 presents the expected monetary cost due to the accumulation of VM migrations in the environment. Fig. 23 compares such results

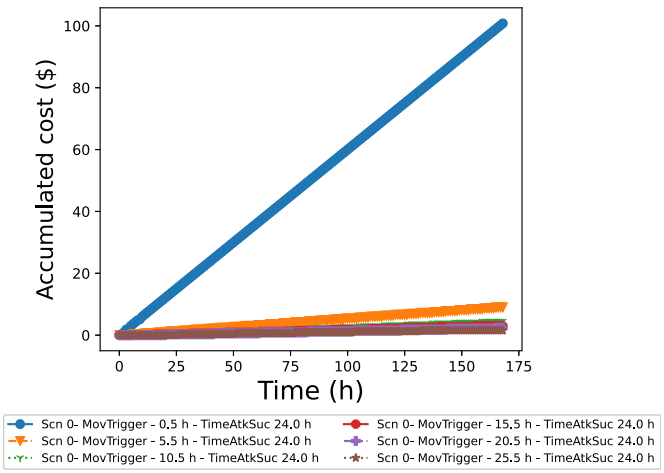


Fig. 22. PyMTDEvaluator output — Cost results.

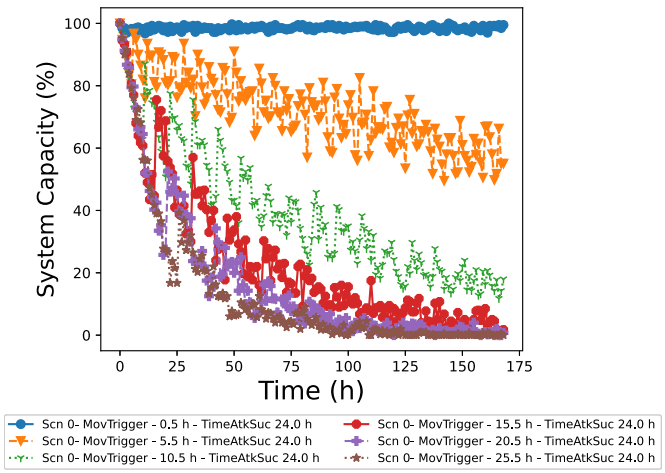
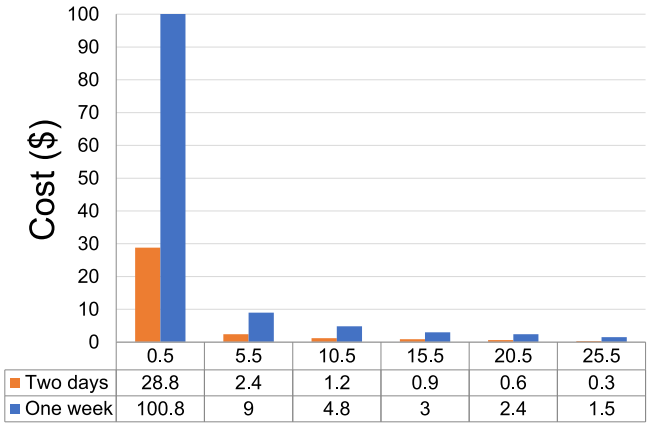
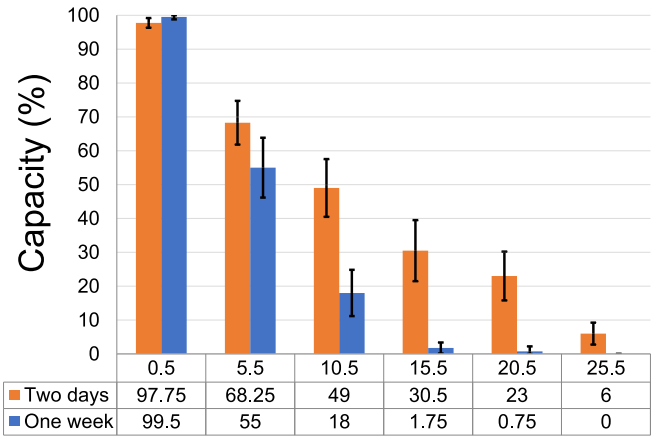


Fig. 24. PyMTDEvaluator output — Capacity results.



	0.5	5.5	10.5	15.5	20.5	25.5
Two days	28.8	2.4	1.2	0.9	0.6	0.3
One week	100.8	9	4.8	3	2.4	1.5

Fig. 23. Cost results — comparison.



	0.5	5.5	10.5	15.5	20.5	25.5
Two days	97.75	68.25	49	30.5	23	6
One week	99.5	55	18	1.75	0.75	0

Fig. 25. Capacity results — comparison.

at two days and one week. Here, the MTD action is supposed to be automated. Therefore, we notice a linear growth in the associated costs during the attack and MTD simulation. These results are helpful in finding the cost–benefit of each one of the proposed approaches. In this illustrative setup, we selected $ 0.3 as the cost for each VM migration.

The results show that the weekly cost of the 0.5-h policy is about 67 times higher than the 25.5-h policy. In only two days, the comparison indicates that the 0.5-h policy is 96 times higher than the 25.5-h policy. These results highlight the considerable difference and possible cost savings when selecting less frequent MTD actions.

Considering the proposed scenario where the MTD action is cheap to deploy, the weekly cost of the 5.5-h policy is less than 10$. Therefore, supposing a similar environment, selecting a 5.5-h policy might be appropriate as it reduces the probability of attack success without implying exaggerated downtime due to VM migrations.

6.5. Capacity

System capacity helps measure how much service the system delivers at a certain time (Heimann et al., 1991). The metric is based on the Capacity Oriented Availability (COA) metric (Maciel, 2023). Here, we extract the capacity metrics by observing the accumulation of tokens in the places related to the attack progress (see Section 4). The metric reveals the deterioration of the quality of the service during the attack. In the studied threat model, each VM migration completely restores the system capacity.

As expected, the curves present a decreasing behavior (see Fig. 24). They have a series of slow negative ramps and steep vertical rises. The vertical rises come from the continuous deployment of VM migration. Therefore, massive variability in the final data is expected. To dissect these results, Fig. 25 compares the two points in time: two days and one week.

The high variability in the Capacity results in broader confidence intervals. Therefore, compared to the previous results, it is harder to narrow the possible values for each one of the points in time. The results for the 0.5-h policy show that the system is expected to have higher capacity in one week than in two days. However, considering the confidence interval, they are nearly the same. As the VM migration maximizes the system capacity, the higher result for one week only suggests that, in the simulation runs, the point in time 168 h accumulated more migrations than the point 48 h.

The system capacity results help understand the expected impact of quality of service during an attack. As mentioned earlier, in practical situations, the parameters used need to be adjusted accordingly. However, the capacity results serve as an insight into setting up Service Level Agreements. Generally, the quality of service metrics (different from system metrics) is more visible from the client side. For example, a client may notice a service quality drop while the server monitoring still shows that the system is available. From this perspective, the capacity metric provides a quality-related measure for the server side. Therefore, the system manager can select proper MTD scheduling policies to maximize capacity or to prevent capacity drop to unacceptable levels.

Table 2
PyMTDEvaluator output – Mean capacity (%) results – One week.

VM migration schedule	95%$CI-$	Capacity	95%$CI+$
0.5 h	98.39	98.49	98.59
5.5 h	70.09	71.82	73.55
10.5 h	36.19	39.30	42.41
15.5 h	20.08	23.47	26.86
20.5 h	12.93	16.27	19.61
25.5 h	9.76	13.04	16.32

Table 3
Summary of the obtained results — One week timeframe.

VM migration schedule	Probability of attack success (%)	Mean expected downtime until one week	Mean expected capacity (%)	Accumulated cost ($)
0.5 h	0	27.31 min	98.49	100.8
5.5 h	35	1.82 days	71.82	9.0
10.5 h	82	4.28 days	39.30	4.8
15.5 h	96	5.32 days	23.47	3
20.5 h	99	5.72 days	16.27	2.4
25.5 h	100	5.90 days	13.04	1.5

In this context, Table 2 presents the mean capacity and 95% confidence interval (CI) for all the studied scenarios. The results refer to the one-week timeframe.

6.6. Multi-criteria decision making results

Table 3 provides a summary of the obtained results. This table shows the transient probability of attack success at time T, the expected accumulated downtime until time T, the expected capacity, and the accumulated monetary cost due to MTD actions.

Multiple factors may influence a decision to use a specific VM migration schedule. For example, in an environment where a sensitive application is running, a high level of security is required. In this situation, the system manager may select more frequent migrations to assure high MTD-related protection (e.g., 30 min between migrations). On the other hand, the system manager may select less frequent migrations to avoid a high availability impact. In any case, adopting Multi-Criteria Decision Making (MCDM) methods provides an objective way to compare the alternatives presented.

A important highlight here is the following. The results of capacity and availability below come from the mean value obtained during the simulation runs and not from the transient values. There is a slight difference between the transient value and the mean value. The transient value (as presented in Sections 6.2 and 6.5) refers to the pointwise value considering the selected time. For example, the transient availability of 0.9974 at one week reveals that the system has a 99.74% chance of being available at the point of time — one week. On the other hand, consider a mean availability of 98.65% during one week. It means that, during the first week of the attack, the system is expected to be available 98.65% of the time. We emphasize that both mean and transient results are present in the PyMTDEvaluator output.

The results in Table 3 show the summary of the obtained results. However, the question that may arise from this analysis is: "*Which alternative is the best?*". The selection of a specific policy depends on how important each one of the metrics is. To aid the selection, as mentioned earlier, *PyMTDEvaluator* embeds two Multi-Criteria Decision-Making methods: Weighted Sum and TOPSIS. For the sake of brevity, the complete definition of these methods is omitted from this paper. We suggest the following material for further reading (Triantaphyllou, 2000; Gal et al., 2013).

In this illustrative scenario, the Multi-Criteria Decision Making methods use the weights presented in Table 1 (i.e., Availability = 30%, Probability of attack success = 50%, Capacity = 5%, and Cost = 15%). The results (see Table 4) show that the best option is the policy of 30 min between MTD actions. The second best is the policy of 5.5 h between MTD actions. Interestingly, the third best is the policy of 25.5 h between migrations. Note that PyMTDEvaluator neglects the possible financial loss due to downtime, and this feature has yet to be implemented.

Table 4
PyMTDEvaluator output — Multi-Criteria Decision Making.

Ranking	Weighted Sum	TOPSIS
1st (best)	**0.5 h**	**0.5 h**
2nd	5.5 h	5.5 h
3rd	25.5 h	25.5 h
4th	10.5 h	20.5 h
5th	15.5 h	10.5 h
6th (worst)	20.5 h	15.5 h

The selection of weights for the MCDM parameters depends on the importance of each attribute. For example, for non-critical applications, we may prefer to reduce costs with MTD (i.e., assign a higher percentage to the cost parameter). We should assign a higher percentage to availability and probability of attack success for security-critical applications. For throughput-sensitive software (e.g., a client–server application), we may prefer to increase the importance of capacity to ensure higher quality of service. Undoubtedly, these parameters and the exact number will vary from user to user.

6.7. Results from a single simulation run

This last set of results for the proposed studied scenario contains the output of a single simulation run (example run). This result comes from a single simulation run of the scenario performed outside of the loop of the main results. Although the results from a single simulation run are derisory for significant analysis, they show a possible system behavior during the execution of the attack-defense setup. Hopefully, these results may provide insight that might go unnoticed from the previous results.

The availability results (Fig. 26) of 0.5 h produce significantly more rise and fall in availability than the other policies. Therefore, in the proposed scenario, it may have a high impact on the network services due to the overhead of VM migration. On the other hand, the same policy is constantly renewing the system capacity due to constant MTD actions. The results show that the capacity (see Fig. 27) is at least 75% during the example simulation run.

6.8. Search for optimal MTD scheduling

PyMTDEvaluator tool enables the search for optimal scheduling of MTD using a semi-automated process. Nevertheless, the results should be treated carefully as the simulation runs may present slightly different results due to the randomness of generated times. We present the step-by-step process below. The search process is based on the *binary search* approach (Nowak, 2008).

① **Defining what is "optimal".** In the search for an optimal solution, our first step is to define what are the targets of the search. In the case of *PyMTDEvaluator*, we can select the targets for the attributes: probability of attack success, capacity, availability, and cost. We also need to define the evaluation time target. For this illustrative example, let us consider the search for the scheduling that maintains the probability of attack success under 5% in the first 48 h of attacker presence with minimal costs. For the sake of simplicity, let us consider the parameters as presented in Table 1.

② **Run the first evaluation with a large range of `Movement trigger`.** Verify whether the obtained results already satisfy the search criteria (i.e., good solution). In case of the absence of good solutions, repeat the process, adjusting the range of `movement trigger`. Once finding a good solution, go to the next step. In the illustrative

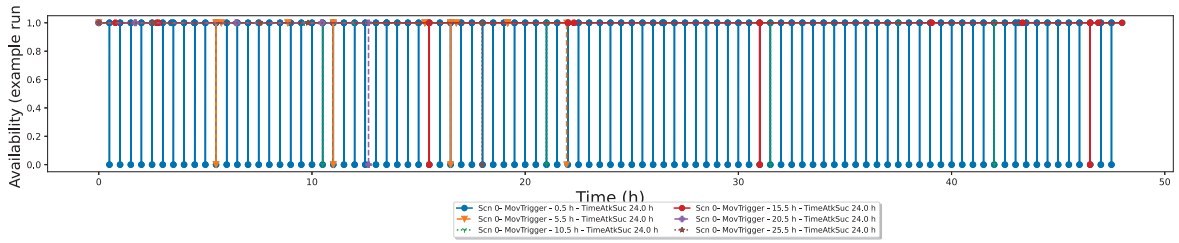


Fig. 26. PyMTDEvaluator output – Example run results – Availability.

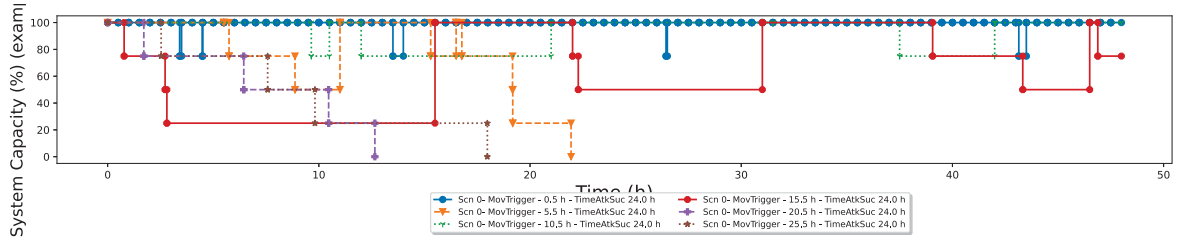


Fig. 27. PyMTDEvaluator output – Example run results – Capacity.

example, we noticed that the appropriate schedule is between 0.5 and 5.5 h (Fig. 21). After additional experiment runs, we noticed that the good solutions are between 3.5 and 4.0 h of interval between MTD actions.

③ **Adjust the level of precision decreasing the `Movement trigger step` parameter**. A good solution may be only sub-optimal. The goal of decreasing the `step` parameter is to increase the precision. It is worth highlighting that the high level of precision is unattainable because of the numerical approximations inherent to the simulation runs. In the illustrative example, we adjust the precision level to 0.1 (i.e., we performed the evaluation with `Movement trigger step` of 0.1 h from 3.5 to 4.0 h). Finally, we find out that the scheduling of **3.9 h** between migrations is the best solution. It achieves a probability of attack success of 5%, cost of $ 3.9, and mean capacity of 90%. We included the related XML file with these parameters in *PyMTDEvaluator* docker container.

7. Threats to validity and limitations

This section presents a non-exhaustive list of threats to the validity and limitations of this study. We also present possible mitigation for each one of the highlighted points.

Lack of proper parameter value validation

In the presented evaluation, we assume illustrative values for the parameters. This way, the presented results in Section 6 are for a hypothetical scenario and may not reflect a real-world scenario. Obtaining generic values for the model parameters is challenging as each system has its characteristics. To mitigate this issue, we propose the *PyMTDEvaluator* tool, which provides flexibility for studying, analyzing, and comparing different scenarios.

It is hard to obtain the parameters' values to feed the model

We acknowledge that obtaining *time to attack success* is hard. We emphasize some alternatives to mitigate this issue. (1) When available, resorting to previous data to estimate the time to attack success is possible. (2) Conduct a comprehensive sensitivity analysis using *PyMTDEvaluator* `Experiment` feature. The sensitivity analysis highlights how the protection fluctuates upon the time to attack success variation. Therefore, assuming a reasonable range for the time to attack success, it is possible to set up a plausible VM migration schedule for each scenario. (3) Replicating an attack and defense campaign in a real testbed. Our previous publication (Torquato and Vieira, 2021) provides the necessary details to reproduce the experiment in Section 3.2.

Lack of VM selection method in the MTD approach

The MTD method assures that the *Attacker VM* position will differ after each VM migration cycle. There are two main alternatives: (1) use attack/intrusion detection methods. In the case of *MemDoS*, it is possible to apply cache monitoring techniques to detect an ongoing attack (Li et al., 2020). (2) Apply *cyclic reconfiguration* in the VMs set after each migration. The *cyclic reconfiguration* strategy modifies the set of VMs in each physical host to guarantee new VM *neighborhood* after each migration cycle.

8. Related works

As mentioned, this paper extends the paper (Torquato et al., 2021b). Nevertheless, it also relates to our previous works (Torquato and Vieira, 2021), which presented the adopted experiment methodology. Besides that, the preliminary proposal of the model appears in the paper (Torquato et al., 2020). Another use of a similar model appears in Torquato et al. (2022).

It is worth highlighting inspiring works in MTD modeling through Petri Nets. Cai et al. (2016) is one of the first attempts at MTD modeling using Petri Nets. Their work focuses on specific aspects of Petri Nets, such as valid markings and marking probability. *PyMTDEvaluator* tries to simplify (and hopefully increase the understandability of) the results presentation by using end–user metrics as *probability of attack success* and *availability*.

Shi et al. (2021) and Chen et al. (2020b,a) propose model-based evaluation for MTD systems. They focus on performance-related metrics such as system throughput and the mean number of jobs waiting. The works from Mendonça et al. (2020) and Distefano et al. (2020) extend the scope by adding the availability metric. Unlike these works, we propose a graphical interface for the model through *PyMTDEvaluator*. Besides that, we present empirical evidence of the validity of the proposed MTD technique.

Enoch et al. (2022b) recently presented a framework for automated cyber defense deployment. The authors consider HARM (Hierarchical

Attack Representation Model) (Hong and Kim, 2015) models in the evaluation process, while we use Stochastic Petri Nets. The fundamental difference from our approach is that they adopt a runtime approach for MTD evaluation and deployment. Instead, we propose a method to support the MTD evaluation during design time.

Another inspiring work from Enoch et al. (2022a) presents a model-based security optimization. Their comprehensive work covers searching for the optimal solution in each studied scenario. Besides that, their proposed models also consider specific system vulnerabilities. Unlike their approach, we focus on the time-based VM migration method. We cover this technique's various aspects, from empirical evidence to modeling and simulation.

Recently, Yang et al. (2023) provided a Stochastic Reward Net model to evaluate VM migration as MTD. Similar to our approach, the authors also used the SimPy library to conduct simulation runs of the model. Different from our work, they add the metrics of task completion probability and data theft probability. Distinct from their work, we publicly deliver the simulation in the form of PyMTDEvaluator. Besides that, we also present empirical evidence of VM migration-based MTD.

9. Conclusion and future work

We proposed a Stochastic Petri Net model for time-based VM migration as MTD evaluation. We performed an attack-and-defense experimental campaign to support the model design. We also presented the *PyMTDEvaluator* tool to facilitate the model analysis. *PyMTDEvaluator* also features Multi-Criteria Decision-Making methods to aid the selection of a time-based VM migration policy.

Compared to the current literature, our work advances the state-of-the-art mainly in the following: (1) experimental campaign considers the *full cycle* (i.e., exploring the whole set of available physical machines) of VM migration as MTD. (2) extended *PyMTDEvaluator* functionalities to provide a more comprehensive MTD evaluation.

In future works, we aim to consolidate *PyMTDEvaluator* search method through multi-objective optimization (as presented in Enoch et al. (2022a)). Besides that, we intend to experiment with VM migration-based MTD against other threat models. Finally, one of the side research lines for the future is the refinement and improvement of the proposed SPN model.

CRediT authorship contribution statement

Matheus Torquato: Writing – original draft, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Paulo Maciel:** Writing – review & editing, Validation, Formal analysis. **Marco Vieira:** Writing – review & editing, Supervision, Software, Methodology, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Code available at Github.

Acknowledgments

Thanks to the anonymous reviewers for the comments on improving this research.

This work is funded by the FCT - Foundation for Science and Technology, I.P./MCTES through national funds (PIDDAC), within the scope of CISUC R&D Unit - UIDB/00326/2020 or project code UIDP/00326/2020.

References

Al-Kuwaiti, Mohamed, Kyriakopoulos, Nicholas, Hussein, Sayed, 2009. A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability. IEEE Commun. Surv. Tutor. 11 (2), 106–124.

Alhozaimy, Sarah, Menascé, Daniel A., 2022. A formal analysis of performance-security tradeoffs under frequent task reconfigurations. Future Gener. Comput. Syst. 127, 252–262.

Andersen, Bjorn, Fagerhaug, Tom, 2006. Root Cause Analysis. Quality Press.

Araujo, Julian, Maciel, Paulo, Andrade, Ermeson, Callou, Gustavo, Alves, Vandi, Cunha, Paulo, 2018. Decision making in cloud environments: an approach based on multiple-criteria decision analysis and stochastic models. J. Cloud Comput. 7 (1), 1–19.

Bace, Rebecca Gurley, 2000. Intrusion Detection. Sams Publishing.

Bai, Jing, Chang, Xiaolin, Machida, Fumio, Trivedi, Kishor S, Li, Yaru, 2023a. Model-driven dependability assessment of microservice chains in mec-enabled iot. IEEE Trans. Serv. Comput. 16 (4), 2769–2785.

Bai, Jing, Chang, Xiaolin, Rodríguez, Ricardo J, Trivedi, Kishor S, Li, Shupan, 2023b. Towards uav-based mec service chain resilience evaluation: A quantitative modeling approach. IEEE Trans. Veh. Technol. 72 (4), 5181–5194.

Bai, Jing, Li, Yaru, Chang, Xiaolin, Machida, Fumio, Trivedi, Kishor S, 2023c. Understanding NFV-enabled vehicle platooning application: A dependability view. IEEE Trans. Cloud Comput. 11 (4), 3367–3380.

Buyya, Rajkumar, Srirama, Satish Narayana, Casale, Giuliano, Calheiros, Rodrigo, Simmhan, Yogesh, Varghese, Blesson, Gelenbe, Erol, Javadi, Bahman, Vaquero, Luis Miguel, Netto, Marco AS, et al., 2018. A manifesto for future generation cloud computing: Research directions for the next decade. ACM Computing Surveys (CSUR) 51 (5), 1–38.

Cabral, Juan B., Luczywo, Nadia Ayelen, Zanazzi, José Luis, 2016. Scikit-criteria: Colección de métodos de análisis multi-criterio integrado al stack científico de python. In: XLV Jornadas Argentinas de Informática e Investigación Operativa (45JAIIO)- XIV Simposio Argentino de Investigación Operativa (SIO) (Buenos Aires, 2016). pp. 59–66, URL http://45jaiio.sadio.org.ar/sites/default/files/Sio-23.pdf.

Cai, Guilin, Wang, Baosheng, Luo, Yuebin, Hu, Wei, 2016. A model for evaluating and comparing moving target defense techniques based on generalized stochastic Petri net. In: Conference on Advanced Computer Architecture. Springer, pp. 184–197.

Chen, Zhi, Chang, Xiaolin, Han, Zhen, Yang, Yang, 2020a. Numerical evaluation of job finish time under MTD environment. IEEE Access 8, 11437–11446.

Chen, Zhi, Chang, Xiaolin, Mišić, Jelena, Mišić, Vojislav B, Yang, Yang, Han, Zhen, 2020b. Model-based performance evaluation of a moving target defense system. In: GLOBECOM 2020-2020 IEEE Global Communications Conference. IEEE, pp. 1–6.

Ciardo, Gianfranco, Muppala, Jogesh K, Trivedi, Kishor S, et al., 1989. SPNP: Stochastic Petri net package. In: PNPM. Vol. 89, Citeseer, pp. 142–151.

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

Distefano, Salvatore, Scarpa, Marco, Chang, Xiaolin, Bobbio, Andrea, 2020. Assessing dependability of web services under moving target defense techniques. In: Proceedings of the 30th European Safety and Reliability Conference (ESREL2020) and the 15th Probabilistic Safety Assessment and Management Conference (PSAM15). Research Publishing/Singapore. pp. 1988–1995.

Enoch, Simon Yusuf, Mendonça, Júlio, Hong, Jin B, Ge, Mengmeng, Kim, Dong Seong, 2022a. An integrated security hardening optimization for dynamic networks using security and availability modeling with multi-objective algorithm. Comput. Netw. 208, 108864.

Enoch, Simon Yusuf, Moon, Chun Yong, Lee, Donghwan, Ahn, Myung Kil, Kim, Dong Seong, 2022b. A practical framework for cyber defense generation, enforcement and evaluation. Comput. Netw. 208, 108878.

Gal, Tomas, Stewart, Theodor, Hanne, Thomas, 2013. Multicriteria Decision Making: Advances in MCDM Models, Algorithms, Theory, and Applications, vol. 21, Springer Science & Business Media.

Govil, A.K., 1972. Priority effect on pointwise availability of the system. Rev. Franç. d'Autom. Inform. Rec. Opér. Rec. Opér. 6 (V1), 47–56.

Heimann, David I., Mittal, Nitin, Trivedi, Kishol S., 1991. Dependability modeling for computer systems. In: Annual Reliability and Maintainability Symposium. 1991 Proceedings. IEEE, pp. 120–128.

Hong, Jin B., Kim, Dong Seong, 2015. Assessing the effectiveness of moving target defenses using security models. IEEE Trans. Dependable Secure Comput. 13 (2), 163–177.

Li, Zhuozhao, Sen, Tanmoy, Shen, Haiying, Chuah, Mooi Choo, 2020. Impact of memory dos attacks on cloud applications and real-time detection schemes. In: 49th International Conference on Parallel Processing-ICPP. pp. 1–11.

Liu, Yun, Trivedi, KISHOR S., 2006. Survivability quantification: The analytical modeling approach. Int. J. Perform. Eng. 2 (1), 29.

Macêdo, Autran, Ferreira, Taís B., Matias, Rivalino, 2010. The mechanics of memory-related software aging. In: 2010 IEEE Second International Workshop on Software Aging and Rejuvenation. IEEE, pp. 1–5.

Maciel, Paulo Romero Martins, 2023. Performance, Reliability, and Availability Evaluation of Computational Systems, Volume I: Performance and Background. CRC Press.

Marsan, M Ajmone, Balbo, Gianfranco, Conte, Gianni, Donatelli, Susanna, Franceschinis, Giuliana, 1998. Modelling with generalized stochastic Petri nets. ACM SIGMETRICS Perform. Eval. Rev. 26 (2), 2.

Matias, Rivalino, Costa, Bruno Evangelista, Macedo, Autran, 2012. Monitoring memory-related software aging: An exploratory study. In: 2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops. IEEE, pp. 247–252.

Mendonça, Júlio, Cho, Jin-Hee, Moore, Terrence J, Nelson, Frederica F, Lim, Hyuk, Zimmermann, Armin, Kim, Dong Seong, 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. pp. 1180–1189.

Mosberger, David, Jin, Tai, 1998. Httperf—a tool for measuring web server performance. ACM SIGMETRICS Perform. Eval. Rev. 26 (3), 31–37.

Nowak, Robert, 2008. Generalized binary search. In: 2008 46th Annual Allerton Conference on Communication, Control, and Computing. IEEE, pp. 568–574.

Patil, Rajendra, Modi, Chirag, 2019. An exhaustive survey on security concerns and solutions at different components of virtualization. ACM Comput. Surv. 52 (1), 1–38.

Salfner, Felix, Tröger, Peter, Polze, Andreas, 2011. Downtime analysis of virtual machine live migration. In: The Fourth International Conference on Dependability. DEPEND 2011, IARIA, pp. 100–105.

Scherfke, Stefan, Lünsdorf, Ontje, Grayson, Peter, LaFevers, Eric, Pinckney, Thomas, Klein, Cristian, Vaidya, Sundar, Reis, Larissa, Reed, Sean, Liu, Zhe, et al., 2020. Simpy. Discrete Event Simul. Python Date Access. 25 (04), 2023.

Sengupta, Sailik, Chowdhary, Ankur, Sabur, Abdulhakim, Alshamrani, Adel, Huang, Dijiang, Kambhampati, Subbarao, 2020. A survey of moving target defenses for network security. IEEE Commun. Surv. Tutor..

Shi, Leyi, Du, Shanshan, Miao, Yifan, Lan, Songbai, 2021. Modeling and performance analysis of satellite network moving target defense system with Petri nets. Remote Sens. 13 (7), 1262.

Shipman, John W., 2013. Tkinter 8.5 reference: a GUI for python. New Mexico Tech Comput. Cent. 54, 356–359.

Silva, Bruno, Matos, Rubens, Callou, Gustavo, Figueiredo, Jair, Oliveira, Danilo, Ferreira, Joao, Dantas, Jamilson, Lobo, Aleciano, Alves, Vandi, Maciel, Paulo, 2015. Mercury: An integrated environment for performance and dependability evaluation of general systems. In: Proceedings of Industrial Track at 45th Dependable Systems and Networks Conference. DSN, pp. 1–4.

Singh, Aarushi, Malik, Sanjay Kumar, 2014. Major MCDM techniques and their application-a review. IOSR J. Eng. 4 (5), 15–25.

Torquato, Matheus, Guedes, Erico, Maciel, Paulo, Vieira, Marco, 2019. A hierarchical model for virtualized data center availability evaluation. In: 2019 15th European Dependable Computing Conference. EDCC, IEEE, pp. 103–110.

Torquato, Matheus, Maciel, Paulo, Vieira, Marco, 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, pp. 50–59.

Torquato, Matheus, Maciel, Paulo, Vieira, Marco, 2021a. Analysis of VM migration scheduling as moving target defense against insider attacks. In: Proceedings of the 36th Annual ACM Symposium on Applied Computing. pp. 194–202.

Torquato, Matheus, Maciel, Paulo, Vieira, Marco, 2021b. PyMTDEvaluator: A tool for time-based moving target defense evaluation: Tool description paper. In: 2021 IEEE 32nd International Symposium on Software Reliability Engineering. ISSRE, IEEE, pp. 357–366.

Torquato, Matheus, Maciel, Paulo, Vieira, Marco, 2022. Software rejuvenation meets moving target defense: Modeling of time-based virtual machine migration approach. In: 2022 IEEE 33rd International Symposium on Software Reliability Engineering. ISSRE, IEEE, pp. 205–216.

Torquato, Matheus, Vieira, Marco, 2019. An experimental study of software aging and rejuvenation in dockerd. In: 2019 15th European Dependable Computing Conference. EDCC, IEEE, pp. 1–6.

Torquato, Matheus, Vieira, Marco, 2020. Moving target defense in cloud computing: A systematic mapping study. Comput. Secur. 92, 101742.

Torquato, Matheus, Vieira, Marco, 2021. VM migration scheduling as moving target defense against memory dos attacks: An empirical study. In: 2021 IEEE Symposium on Computers and Communications. ISCC, IEEE, pp. 1–6.

Tosi, Sandro, 2009. Matplotlib for Python Developers. Packt Publishing Ltd.

Triantaphyllou, Evangelos, 2000. Multi-criteria decision making methods. In: Multi-Criteria Decision Making Methods: A Comparative Study. Springer US, Boston, MA, ISBN: 978-1-4757-3157-6, pp. 5–21. http://dx.doi.org/10.1007/978-1-4757-3157-6_2.

Trivedi, Kishor S., Bobbio, Andrea, 2017. Reliability and Availability Engineering: Modeling, Analysis, and Applications. Cambridge University Press.

Von Kistowski, Joakim, Eismann, Simon, Schmitt, Norbert, Bauer, André, Grohmann, Johannes, Kounev, Samuel, 2018. Teastore: A micro-service reference application for benchmarking, modeling and resource management research. In: 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. MASCOTS, IEEE, pp. 223–236.

Yang, Xin, Smahi, Abla, Li, Hui, Zhang, Huayu, Li, Shuo-Yen Robert, 2023. An SRN-based model for assessing co-resident attack mitigation in cloud with VM migration and allocation policies. In: GLOBECOM 2023-2023 IEEE Global Communications Conference. IEEE, pp. 4995–5000.

Zhang, Tianwei, Lee, Ruby B., 2017. Host-based dos attacks and defense in the cloud. In: Proceedings of the Hardware and Architectural Support for Security and Privacy. HASP '17, Association for Computing Machinery, New York, NY, USA, ISBN: 9781450352666, http://dx.doi.org/10.1145/3092627.3092630.

Zhang, Tianwei, Zhang, Yinqian, Lee, Ruby B., 2017. Dos attacks on your memory in cloud. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 253–265.

Zimmermann, Armin, 2017. Modelling and performance evaluation with timenet 4.4. In: International Conference on Quantitative Evaluation of Systems. Springer, pp. 300–303.

Matheus Torquato is a Ph.D. candidate at the University of Coimbra. His research interests include Cloud Computing, Performance, Dependability, and Security Modeling. His current research focuses on designing analytical models to evaluate the performance, dependability, and security of moving target defense deployments in cloud computing. He received his Master's Degree in Computer Science from the Federal University of Pernambuco. He is Assistant Professor at the Federal Institute of Alagoas, Campus Arapiraca. His website is: https://www.matheustorquato.com/.

Paulo Maciel graduated in Electronic Engineering from the University of Pernambuco in 1987. He earned his master's and doctorate in Electronic Engineering and Computer Science from the Federal University of Pernambuco. During his doctorate, he completed a "sandwich internship" at EberhardKarls-Universität Tübingen, Germany, from 1996 to 1997. From 1989 to 2003, Paulo Maciel was a professor at the Department of Electrical Engineering at the University of Pernambuco. Since 2001, he has been at the Computer Center of the Federal University of Pernambuco, where he is a full professor. He also serves as a member of the National Council for Scientific and Technological Development (CNPq). In 2011, he took a sabbatical year at the Department of Electrical and Computer Engineering, Edmund T. Pratt School of Engineering, Duke University, USA. Paulo's research interests include performance, reliability, availability, capacity planning, and stochastic models, with applications in cloud computing, sustainable data centers, manufacturing, integration, and communication systems.

Marco Vieira is a Professor of Computer Science at the University of North Carolina at Charlotte. Before joining UNC Charlotte in 2023, he was a Professor at the University of Coimbra. His research interests include dependable computing, dependability and security assessment and benchmarking, software security, fault and vulnerability injection, failure prediction, static analysis and software testing, subjects in which he authored or co-authored more than 250 works in refereed conferences and journals. Marco is Chair of the IFIP WG 10.4 on Dependable Computing and Fault Tolerance, Associate Editor of the IEEE Transactions on Dependable and Secure Computing, Steering Committee Vice-Chair of the IEEE/IFIP International Conference on Dependable Systems and Networks, and member of the Steering Committee of the IEEE International Symposium on Software Reliability and Engineering and of the Latin-American Symposium on Dependable and Secure Computing. He served as Program Chair for the major conferences on the dependable computing area.