1 2 9 0

UNIVERSIDADE Đ
COIMBRA

Matheus D'Eça Torquato de Melo

Models for availability and security evaluation of time-based Virtual Machine migration as Moving Target Defense

PhD Thesis in Informatics Engineering, Architectures, Networks and Cybersecurity, supervised by Professor Marco Paulo Amorim Vieira, and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

December 2023

1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Models for availability and security evaluation of time-based Virtual Machine migration as Moving Target Defense

Matheus D'Eça Torquato de Melo
mdmelo@dei.uc.pt

Doctoral Program in Informatics Engineering, Architectures, Networks and Cybersecurity
PhD Thesis submitted to the University of Coimbra

Advised by Professor Marco Paulo Amorim Vieira

December, 2023

1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE Ð
COIMBRA

Models for availability and security evaluation of time-based Virtual Machine migration as Moving Target Defense

Matheus D'Eça Torquato de Melo
mdmelo@dei.uc.pt

Programa de Doutoramento em Engenharia Informática, Arquiteturas, Redes e Cibersegurança
Tese de Doutoramento apresentada à Universidade de Coimbra

Orientado pelo Professor Doutor Marco Paulo Amorim Vieira

Dezembro, 2023

This work was partially supported by the Portuguese Foundation for Science and Technology (FCT) under PhD grant SFRH/BD/146181/2019 and project grants nº 777154, POCI-01-0247-FEDER-039676, and POCI-01-0247-FEDER-045907.

“All models are wrong,
but some are useful.”

- George Box

Acknowledgments

"Thank God for giving me the strength to get here."

I am grateful to all the people who directly or indirectly helped me to reach the final step of this research. During the last years, I faced various problems that I could not overcome without your help. Thank you!

The first person I would like to thank is Prof. Marco Vieira. Marco, as you can tell, my Ph.D. journey was not the most linear one. Thanks for your patience and support during the difficult times and for your guidance and brilliant ideas during the most productive ones. Thanks for the verbal and non-verbal advice you gave me. I hope to continue learning a little more from your experience in the next years. I can tell you that coming to Coimbra to work with you was worth it.

I would like to thank our collaborator, Prof. Paulo Maciel, for being available to help in the modeling process and providing insights and directions.

I would like to thank all my colleagues at CISUC - Bruno de Jesus, Charles Gonçalves, David Lima, José Pereira, Magnus Cruz, and João Campos. A special thanks to the Software and Systems Engineering (SSE) group members, who warmly received me as a Ph.D. student. I would also like to thank Prof. Henrique Madeira for the valuable moments of conversation.

Thanks also to my colleagues at IFAL Campus Arapiraca. Thanks for understanding my *unavailable* state during the Ph.D. time.

I would also like to mention, on behalf of all my friends who cheered me up during the difficult times, Maria, Josemaria, Francisco, Edeilson, and Luís Júnior. Your support was crucial.

I want to thank my parents, Torquato and Isabel, my sister Isabela, and my brother (and best friend) Lucas.

Finally, I would also, and above all, like to thank my family. First, my wife, Carla. I can not express enough how grateful I am for your help and support. Thanks for maintaining continuous love and encouragement. My daughters Maria Clara, Maria Cecília, and Maria Carol whose love, hugs and smiles are the best rewards.

Abstract

Cybersecurity is a top concern in modern virtualization infrastructures and cloud computing. The growth of sophisticated cybersecurity threats imposes a heavy burden on the current defensive mechanisms. A skilled and motivated attacker gathers information from the system before launching an attack and, with the knowledge acquired, boosts the attack success probability. At the same time, defenders should protect the system from all possible threats. This unbalanced game (also known as *attacker asymmetric advantage*) is intrinsic to the attack-defense dynamics in cyberspace. Deploying a proactive and adaptive defense is of utmost importance to mitigate this issue.

Moving Target Defense (MTD) was proposed to bring more balance to the attack-defense game. The main goal is to apply a continuous shift in the attack surface to confuse attackers or even to react to attacks dynamically, thus reducing that advantage by moving or changing the configuration of the resources. For example, in an attack from a VM targeting its host, Virtual Machine (VM) migration may confuse the attacker as the target changes. Indeed, the current literature shows that VM migration is among the main cloud MTD techniques. However, although a roadblock for MTD adoption is its effectiveness evaluation, most works in the field focus on proposing new strategies and validating them instead of providing generic evaluation methods. Therefore, the proposal of evaluation methods to analyze and compare VM migration-based MTD is still a research challenge.

Measuring the cost and benefit of deploying a MTD is a starting point for its adoption. In cloud computing environments, a proper MTD deployment plan should comprise the evaluation of its impact on the other metrics of interest. As the cloud usually hosts multiple clients with their own goals, a comprehensive analysis of MTD impact is paramount. System availability, for example, is one of the primary metrics of interest for cloud managers and providers. It usually appears as a basic Service Level Agreement (SLA) in the majority of public Infrastructure as a Service (IaaS) clouds. Adopting a MTD alternative, which severely affects system availability, is not viable. This leads to the following research challenge: **devising sound evaluation approaches for VM migration-based MTD while considering aspects of its effectiveness and availability impact.**

This thesis contributes towards tackling such research challenge through the proposal of Stochastic Petri Net (SPN) models for evaluating availability and security of systems with VM migration as MTD. The proposal and analysis of the models follow a structured approach. Firstly, we empirically observe the system under the attack-defense scenario. Secondly, the models are proposed, starting from a baseline model for availability evaluation to a final model with MTD. Finally, a tool for automating the scenario analysis and comparison is provided.

The key contributions of this work can be divided into two groups. The first comprises *comprehensive evaluation of systems with VM migration as software rejuvenation*. Although our goal in this part of the research was to design a *baseline* model for cloud availability evaluation, we took one step further. For example, we extend the availability model into a performability model. Therefore, we compute not only availability but also reliability and performance-related metrics. Moreover, we propose an approach for evaluating the security in the availability-oriented models using a new metric named RiskScore. With the RiskScore, we shift the security evaluation from an attacker perspective to a system-state perspective, resulting in an attacker-disregard evaluation approach.

The second group of contributions comprises the *evaluation of a system using* VM *migration-based* MTD. The contributions here range from analytical models for a combined evaluation of the probability of attack success and availability to a simulation-based evaluation environment (*PyMTDEvaluator*) with a friendly user interface. We also propose a set of secondary metrics to reveal insights about the MTD effectiveness, which might be useful for similar research. For example, the *tolerance level* metric analyzes the probability of attack success results to find information on how long the system can keep the probability of attack success under a specific threshold.

In a cybersecurity landscape with ever-growing threats with inherent *attacker asymmetric advantage*, the development of evaluation methods for MTD is of utmost importance. The advent of *zero trust* and *attack tolerance* security approaches increases the importance of MTD deployments. The research presented in this thesis intends to fill some relevant research gaps, easing the way towards MTD adoption. Finally, considering a broadly used technique as VM migration improves the understanding of the potential security benefits we may achieve while using it in virtualized environments.

Keywords: Moving Target Defense, VM migration, Stochastic Petri Net, Cybersecurity, Software Aging and Rejuvenation

Resumo

A cibersegurança é um dos principais desafios para as infraestruturas de virtualização modernas e ambientes de computação em nuvem. O crescimento de ameaças sofisticadas é um desafio para os mecanismos de defesa atuais. Através de uma recolha prévia de informações sobre o sistema, os atacantes aumentam a probabilidade de ter realizar ataques com sucesso. Por outro lado, os defensores devem proteger o sistema de todas as ameaças. Este jogo não balanceado (também conhecido como *vantagem assimétrica do atacante*) é intrínseco à dinâmica de ataque-defesa no ciberespaço. Desenvolver defesas proativas e adaptativas é um aspecto chave para mitigar essas ameaças.

MTD foi proposto para trazer mais equilíbrio para a dinâmica de ataque-defesa. O principal objetivo é alterar a superfície de ataque continuamente para confundir os atacantes ou reagir dinamicamente a ataques em curso. Esta abordagem reduz a vantagem do atacante através da alteração da configuração dos recursos. Por exemplo, num ataque de uma Máquina Virtual (VM) à Máquina Física (PM) hospedeira, a utilização da migração de VMs pode confundir o atacante, pois o alvo está em constante mudança. De facto, a literatura atual mostra que a migração de VMs está entre as principais técnicas de MTD para a computação em nuvem. No entanto, apesar de um desafio à adoção de MTD ser a avaliação da sua eficácia, a maioria dos trabalhos concentra-se em propor e validar novas estratégias de MTD, em vez de propor mecanismos de avaliação mais genéricos.

A avaliação do custo-benefício da implementação de MTD é o primeiro passo para a sua adoção. Nos ambientes de computação em nuvem, um plano de implantação de MTD adequado deve contemplar a avaliação do seu impacto em outras métricas de interesse. Como os ambientes de nuvem geralmente hospedam diversos clientes que possuem objetivos particulares, uma avaliação abrangente é de extrema importância. A disponibilidade do sistema, por exemplo, é uma das principais métricas de interesse dos fornecedores de computação em nuvem, sendo frequentemente mencionada em acordos de nível de serviço (SLA). A adoção de uma técnica de MTD que afete severamente a disponibilidade não é viável. Este contexto levanta o seguinte desafio de investigação: **conceber abordagens de avaliação sólidas para MTD baseado em migração de VMs, considerando aspectos de eficácia e impacto na disponibilidade.**

Esta tese contribui para a diminuição do referido desafio através da proposta de modelos de Redes de Petri Estocásticas (SPN) para a avaliação de disponibilidade e segurança de sistemas com migração de VMs como MTD. Tal proposta e análise segue uma abordagem estruturada. Primeiramente, observa-se o sistema em contextos de ataque-defesa. Depois, propõem-se os modelos de avaliação, passando de um modelo de base para avaliação de disponibilidade até um modelo final considerando MTD. Finalmente, é apresentada uma ferramenta para automatizar a comparação e análise de cenários.

As contribuições desta tese podem ser divididas em dois grupos. O primeiro engloba *uma avaliação abrangente de sistemas com migração de VMs como rejuvenescimento de software.* Apesar do nosso objetivo inicial ser propor um modelo de base para a avaliação da disponibilidade de nuvem, a tese amplia esse objetivo. Por exemplo, estende o modelo de disponibilidade para um de *performability*, tornando possível obter, não apenas a disponibilidade, como também a confiabilidade e métricas relacionadas com o desempenho. Além disso, a tese apresenta uma abordagem para a avaliação de segurança no mesmo contexto, utilizando a métrica RiskScore. Esta muda o contexto da avaliação de segurança de uma perspetiva focada no atacante para uma perspetiva focada no sistema, resultando numa abordagem que ignora as características do atacante.

O segundo grupo de contribuições foca na *avaliação de sistemas com migração de VMs como MTD.* As contribuições aqui partem de modelos analíticos para a avaliação da probabilidade de sucesso do ataque e da disponibilidade até um ambiente de simulação (*PyMTDEvaluator*) com uma interface amigável para o utilizador. Além disso, a tese apresenta um conjunto de métricas secundárias para caraterizar a eficácia do MTD. A métrica de nível de tolerância, por exemplo, analisa as curvas de probabilidade de sucesso do ataque para encontrar o período em que o sistema é capaz de manter a probabilidade de sucesso do ataque abaixo de um certo limiar.

Num cenário de cibersegurança com ameaças crescentes e uma vantagem assimétrica inerente dos atacantes, o desenvolvimento de métodos de avaliação de MTD é de fundamental importância. O advento de *zero trust* e *attacker tolerance* aumenta a relevância de investigação na área de MTD. Esta tese surge para preencher algumas das lacunas no corpo de investigação atual. Finalmente, a consideração de uma técnica amplamente utilizada como a migração de VMs melhora o entendimento de potenciais benefícios de segurança advindos do seu uso.

Palavras-chave: *Moving Target Defense*, Migração de Máquinas Virtuais, Redes de Petri Estocásticas, Cibersegurança, Envelhecimento e Rejuvenescimento de Software

Foreword

The work detailed in this thesis was accomplished at the Software and Systems Engineering (SSE) group of the Centre for Informatics and Systems of the University of Coimbra (CISUC), within the context of the following projects and grants:

- **ATMOSPHERE** – Adaptive, Trustworthy, Manageable, Orchestrated, Secure Privacy-assuring Hybrid, Ecosystem for REsilient Cloud Computing; a 24-month project aiming at the design and development of an ecosystem of a framework, platform and application of next generation trustworthy cloud services on top of an intercontinental hybrid and federated resource pool. The framework considers a broad spectrum of properties and their measures. The platform supports the building, deployment, measuring and evolution of trustworthy cloud resources, data network and data services. ATMOSPHERE is funded by the European Union under the Cooperation Programme, Horizon 2020 grant agreement No 777154.

- **AIDA** – Adaptive, Intelligent and Distributed Assurance Platform; the project aims at improving a platform used by Mobileum for integral risk management in companies. This platform ensures revenue, corporate conditions and fraud control for companies. Thanks to the newest version of the platform, developed by AIDA, companies will be able to collect and monitor data in an extremely flexible way, with real-time guarantees, security and reliability. AIDA is co-financed by the European Regional Development Fund (ERDF) through the Operational Program for Competitiveness and Internationalization – COMPETE 2020 (POCI-01-0247-FEDER-045907) and by the Portuguese Foundation for Science and Technology under CMU Portugal Program.

- **TalkConnect** – Voice Architecture over Distributed Network (POCI-01-0247-FEDER-039676); the majority of cloud contact center providers have the ambition to have a great geographic coverage. In order to achieve this, these solutions are based on a model in which the communications are based on a solution provided by a third party (CPaaS) which guarantees the desired geographical coverage. Although this solution has its advantages, it also has relevant limitations in some markets were the local regulation is very strict. The alternative is the direct connection with local telcos. However, an analysis of these kind of solutions reveals limitations, not being an effective answer to the identified problem. In this scenario, TalkConnect intends to research and implement a new solution that combines multiple direct telco connections with third party CPaaS providers with global coverage. Consortium is compoesed by Talkdesk (main promoter), Vodafone, and University of Coimbra.

- **Ph.D. grant** – Foundation for Science and Technology (FCT) grant

SFRH/BD/146181/2019.

The contributions of this thesis resulted in several publications in international peer-reviewed journals and conferences:

1. Matheus Torquato, Erico Guedes, Paulo Maciel, and Marco Vieira. 'A hierarchical model for virtualized data center availability evaluation'. In 2019 15th European Dependable Computing Conference (EDCC), pp. 103-110. IEEE, 2019.
2. Matheus Torquato, and Marco Vieira. 'An experimental study of software aging and rejuvenation in dockerd'. In 2019 15th European Dependable Computing Conference (EDCC), pp. 1-6. IEEE, 2019. (Distinguished paper award)
3. Matheus Torquato, Paulo Maciel, and Marco Vieira. 'A model for availability and security risk evaluation for systems with VMM rejuvenation enabled by VM migration scheduling'. IEEE Access 7 (2019): 138315-138326.
4. Matheus Torquato, Lucas Torquato, Paulo Maciel, and Marco Vieira. 'IaaS cloud availability planning using models and genetic algorithms'. In 2019 9th Latin-American Symposium on Dependable Computing (LADC), pp. 1-10. IEEE, 2019.
5. Matheus Torquato, Paulo Maciel, and Marco Vieira. 'Availability and reliability modeling of VM migration as rejuvenation on a system under varying workload'. Software Quality Journal 28, no. 1 (2020): 59-83.
6. Matheus Torquato, and Marco Vieira. 'Moving target defense in cloud computing: A systematic mapping study'. Computers & Security 92 (2020): 101742.
7. Matheus Torquato, Charles F. Goncalves, and Marco Vieira. 'An Availability Model for DSS and OLTP Applications in Virtualized Environments'. In 2020 16th European Dependable Computing Conference (EDCC), pp. 85-92. IEEE, 2020.
8. Matheus Torquato, Paulo Maciel, and Marco Vieira. 'Security and availability modeling of vm migration as moving target defense'. In 2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 50-59. IEEE, 2020.
9. Matheus Torquato, Paulo Maciel, and Marco Vieira. '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. 2021.
10. Matheus Torquato, and Marco Vieira. 'VM Migration Scheduling as Moving Target Defense against Memory DoS Attacks: An Empirical Study'. In 2021 IEEE Symposium on Computers and Communications (ISCC), pp. 1-6. IEEE, 2021.
11. Matheus Torquato, Paulo Maciel, and Marco Vieira. 'PyMTDEvaluator: A

Tool for Time-Based Moving Target Defense Evaluation'. In 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE), pp. 357-366. IEEE, 2021.

12. Matheus Torquato, Paulo Maciel, and Marco Vieira. 'Model-Based Performability and Dependability Evaluation of a System with VM Migration as Rejuvenation in the Presence of Bursty Workloads'. Journal of Network and Systems Management 30, no. 1 (2022): 1-33.

13. Matheus Torquato, Paulo Maciel, and Marco Vieira. 'Software Rejuvenation Meets Moving Target Defense: Modeling of Time-Based Virtual Machine Migration Approach'. In 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE), pp. 357-366. IEEE, 2022.

The following papers were also published in workshops and secondary tracks:

1. Matheus Torquato, and Marco Vieira. 'Interacting SRN models for availability evaluation of VM migration as rejuvenation on a system under varying workload'. In 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 300-307. IEEE, 2018.

2. Matheus Torquato, and Marco Vieira. 'Towards models for availability and security evaluation of cloud computing with moving target defense'. In 2019 15th European Dependable Computing Conference (EDCC) - Student Forum - arXiv preprint arXiv:1909.01392 (2019).

Contents

List of Figures

List of Tables

Acronyms

AHP	Analytic Hierarchy Process.
AI	Artificial Intelligence.
APT	Advanced Persistent Threat.
AR	Aging Related.
AT	Attack Tree.
CPS	Cyber-Physical Systems.
CPU	Central Processing Unit.
CTMC	Continuous Time Markov Chain.
DoS	Denial of Service.
DTMC	Discrete Time Markov Chain.
FT	Fault Tree.
HARM	Hierarchical Attack Representation Model.
IaaS	Infrastructure as a Service.
IFR	Increasing Failure Rate.
IoT	Internet of Things.
IP	Internet Protocol.
IT	Information Technology.
KVM	Kernel Virtual Machine.
MC	Markov Chain.
MCDM	Multi Criteria Decision Making.
MITM	Man-in-the-middle.
ML	Machine Learning.
MTD	Moving Target Defense.
MTTF	Mean Time to Failure.
MTTR	Mean Time to Repair.
NIST	United States National Institute of Standards and Technology.
OS	Operating System.
PM	Physical Machine.
PN	Petri Net.

POST	Power On Self Test.
QoS	Quality of Service.
RBD	Reliability Block Diagram.
SaaS	Software-as-a-Service.
SBE	Software Behavior Encryption.
SDN	Software Defined Network.
SLA	Service Level Agreement.
SMP	Semi Markov Process.
SPN	Stochastic Petri Net.
SRN	Stochastic Reward Net.
SWARE	Stress Wait Rejuvenate.
TOPSIS	Technique for Order Preference by Similarity to Ideal Solution.
TTARF	Time To Aging Related Failure.
VIM	Virtual Infrastructue Manager.
VM	Virtual Machine.
VMM	Virtual Machine Monitor.

Chapter 1

Introduction

Cloud computing is nowadays the Information Technology (IT) foundation for several businesses worldwide. Companies and organizations rely on cloud systems to deploy, scale, and run their applications [Sadiku et al., 2014]. The paradigm is so successful that several new concepts are continuously arising as a result of its popularity (e.g., mobile cloud computing, *cloud, edge and fog computing*, Software-as-a-Service (SaaS))[Francis, 2018; Huang and Wu, 2017]. In one way or another, cloud computing is now in our daily lives. For example, we consume cloud computing services when we upload our archives to a storage cloud or use spreadsheets online.

In the traditional *on premises* IT deployment, the users have a considerable up-front cost as they need to allocate the physical resources and all the required apparatus (e.g., cooling, network devices). Through cloud computing, the users can promptly run their applications by outsourcing the IT capabilities to a cloud provider [Buyya et al., 2009]. Even companies that own and manage their data centers are adapting the resources to run a cloud environment. This approach is known as *private cloud* deployment [Goyal, 2014].

Cloud computing relies on *virtualization* technology in its core architecture. The goal of *virtualization* is to increase the efficient resource usage [Portnoy, 2012]. To reach that goal, it enables running multiple systems inside the same Physical Machine (PM). These multiple systems are known as Virtual Machines (VMs). Each VM can run its own Operating System (OS), set of applications and services. The ownership of the set of VMs in a cloud environment is, usually, not strict for a single user. In fact, we frequently have several clients running different applications in the same cloud environment, even sharing the same PM (i.e., *multi-tenancy* feature).

Due to *multi-tenancy*, in a *public cloud* deployment (where the client outsources the computing capabilities to a provider instead of using on-premise virtualized resources), the environment is open for clients to rent and use the available resources. Unfortunately, this feature opens the door for malicious users to attack the shared resources aiming at affecting the co-resident VMs. This way, the attacker is no longer outside the environment but can launch attacks inside the cloud computing environment.

Dependability and security are among the major concerns of cloud computing users [Franklin, 2023]. Many organizations seek to improve their service dependability when selecting the cloud to host their applications [Pluralsight, 2023]. Additionally, cloud computing security is at the top of challenges for cloud computing [Flexera, 2023].

Although it is usual to consider that only *public clouds* are liable to suffer attacks from inside users (as the *private cloud* deployment usually has a controlled set of users), advances on cybersecurity[1] threats lead to adopting a *zero trust* approach in all contexts. Accordingly to United States National Institute of Standards and Technology (NIST), *"Zero trust assumes there is no implicit trust granted to assets or user accounts based solely on their physical or network location (i.e., local area networks versus the internet) or based on asset ownership (enterprise or personally owned)"* [Stafford, 2020].

One of the motivations for shifting to *zero trust* is the existence of unknown (*zero day*) and *multi-stage* attacks and even malicious authorized users inside the cloud. *Zero day* attacks exploit unknown (and probably weak defended) system flows [Parrend et al., 2018]. This way, the system is vulnerable to attacks in areas without known vulnerabilities and proper defensive mechanisms. *Multi-stage* attacks are complex and have multiple phases. Advanced Persistent Threat (APT)-based attacks adopt a *multi-stage* approach to boost attack success probability [Tatam et al., 2021]. An example of an APT attack following a *multi-stage* approach is given by the *cyber kill chain* [Bahrami et al., 2019].

The *Cyber Kill Chain* has seven phases [Yadav and Rao, 2015]: *i)* **reconnaissance** - information gathering, identification of targets; *ii)* **weaponize** - production of a deliverable cyber weapon; *iii)* **delivery** - transmission of the cyber weapon to the target environment; *iv)* **exploitation** - activation of the attack on the environment; *v)* **installation** - install and maintain a backdoor; *vi)* **command and control** - allow remote server to communicate with the target; and *vii)* **act on objective** - data exfiltration, system disruption.

The proposal of effective defensive mechanisms is challenging in scenarios with sophisticated and advanced threats. Furthermore, the default static security configuration works in favor of the attackers as, in such contexts, they can gain knowledge (i.e., reconnaissance phase) about the environment to design more powerful attacks. Cloud computing environments should leverage internal technologies to increase protection against advanced threats. For example, *Virtualization* technology brings one interesting manageability feature, VM migration, that consists of moving the VMs across the available PMs. We can deploy VM migration in a variety of scenarios: for example, *for improving sustainability* - by packing the VMs in fewer PM hosts, *for improving dependability* - by moving VMs away from a faulty PM host, and *for improving security* - moving VMs away from a compromised host, as a Moving Target Defense (MTD) action.

MTD raises as a flexible technique for system security improvement. The United States Department of Homeland Security defines MTD as *"the concept of controlling change across multiple system dimensions to increase uncertainty and ap-*

[1] In the context of this thesis, we use the terms *cybersecurity* and *security* interchangeably. In the scope of this thesis, both have the same meaning.

parent complexity for attackers, reduce their window of opportunity and increase the costs of their probing and attack efforts." [U.S. Department Homeland of Security , 2020].

1.1 Problem Statement

MTD consists on continuously shifting the available *attack surface* to confuse attackers or to dynamically react to an ongoing attack [Jajodia et al., 2011]. Despite the variety of definitions of *attack surface* [Theisen et al., 2018], here we follow the one from Manadhata and Wing [2010]: "*Intuitively, a system's attack surface is the set of ways in which an adversary can enter the system and potentially cause damage*". Let us consider a practical example. An attacker controls a set of VMs in the environment. From the VMs, the attacker runs a malicious action against the shared resources of the PM host (i.e., a *host-based* attack - an insider attack targeting the PM host). In this scenario, the triggering of VM migration may prevent the attack success as the VMs of the attacker is remapped to another PM host. This is an example of VM migration-based MTD deployment. Section 2.4 presents more specific details of MTD deployments.

Recent surveys and technical reports point out MTD as a necessity for effective cybersecurity [Ross et al., 2022][Pingree, 2023]. PICUS® 2023 Red Report [PICUS, 2023] show that about 32% of 500000 malware analyzed apply *defense evasion* tactic. MITRE® ATT&CK matrix defines *defense evasion* tactic as the ability to evade the known detection mechanisms [MITRE, 2023]. An effective MTD deployment may help fight those threats as it can change the system configuration, thus dynamically modifying the nature of the current defensive measures. As the attacks and malicious events are (usually) unexpected, it is beneficial to deploy such a proactive defense.

MTD is considered a proactive defensive mechanism [Cho et al., 2020][Soussi et al., 2021]. The current cloud computing features favor the MTD development and deployment as it is possible to continuously change the system configuration through specific algorithms as a security measure. For example, VM migration is a potential solution to support cloud computing MTD. Although VM migration-based MTD is not able to defend against all threats, it appears as a defense against *host-based* attacks. It is one of the preferred MTD alternatives, as various cloud computing platforms already offer it *off-the-shelf*.

There are several types of MTD deployments for cloud computing environments. For example, in the application layer, MTD uses Software Behavior Encryption (SBE) that consists of changing the application properties (e.g., data format or code) at runtime to confuse attackers. In the platform layer, MTD can apply VM migration for dynamic environment reconfiguration. Finally, in the network layer, MTD can leverage the Software Defined Network (SDN) paradigm to change network properties as network addresses and routes dynamically.

Previous works also showed the effectiveness of MTD deployment in environments like the Internet of Things (IoT) [Kahla et al., 2018], Virtualized Containers [Azab et al., 2016], SDN [Chowdhary et al., 2018], and cloud computing [Villarreal-

Vasquez et al., 2017]. In the cloud computing context, MTD techniques can be used to thwart or reduce the impact of security attacks such as co-residency attacks [Kashkoush et al., 2018] and Distributed Denial of Service attacks [Jia et al., 2014].

An MTD deployment plan for cloud computing should comprise the evaluation of possible impacts on different system attributes such as availability. Besides the security concern, some cloud-hosted applications need high availability levels. Several strategies to achieve high availability include failover techniques, redundancy, and software rejuvenation. The problem is evaluating the possible availability and security impacts of applying MTD in such scenarios.

When applying a Moving Target Defense, there is usually a trade-off between security and availability in cloud computing systems. For example, in MTD based on VM migration, VMs are moved from one physical machine to another. This remapping can, for example, avoid co-residency attacks, but each VM migration (even in a Live Migration mode) has an associated downtime [Clark et al., 2005]. If we decide to perform too frequent VM migrations, we may achieve higher levels of system security but lower availability levels. Otherwise, if the system manager deploys less frequent migrations, the system may reach higher availability levels, but that decreases the MTD-related protection.

The main challenge of our research work is to study "*what are the trade-offs between cloud computing availability and security when applying time-based VM migration as MTD?*". In this context, we have three main evaluation approaches [Jain, 1990]: measurement, simulation, and modeling. Measurement-based methods require a system implementation to be exposed to tests and benchmarking routines to produce the evaluation output. Measurement-based methods have precise results but are often specific to a platform or software version. Availability evaluation through measurement-based methods is usually prohibitive due to the long mean time to failure of the hardware and software components. Simulation-based evaluation methods require a platform to perform simulations. Due to its flexibility, simulation-based evaluation seems interesting for both availability and security aspects, but simulation environments for MTD evaluation are still an open problem. Finally, modeling is particularly interesting in scenarios with combined evaluation metrics [Jain, 1990]: it does not require implementing complex simulation environments nor deploying monitors for the events of interest. In fact, previous works show that modeling-based methods are suitable for cloud computing availability and security evaluation [Kim et al., 2009; Dantas et al., 2012; Fitch and Xu, 2012].

1.2 Contributions

This work delivers a **set of models for availability and security evaluation of cloud computing when applying MTD based on VM migration**. Starting from a baseline model without MTD, we build up models to evaluate VM migration-based MTD against *persistent* and *non-persistent* threats, derive a tool from those models to facilitate the scenarios analysis and comparison, and conduct an empirical evaluation to support the model design choices. Based on the results of the models, it is possible to select specific policies to reach (or approximate) the desired levels of security and availability. Our models are based on Stochastic Petri Net (SPN) that are extensively used for cloud computing availability evaluation [Maciel et al., 2021; Machida et al., 2010]. Petri Net (PN) based models (as SPNs) are also suitable for security evaluation [Wang et al., 2013].

This thesis advances state-of-the-art on MTD by reducing the gap in the current literature concerning the *absence of combined evaluation methods for availability and security of systems under VM migration-based MTD*. We propose a series of SPN models to overcome the existing challenges and limitations in this field. The thesis is thus developed considering three building blocks: *availability model →availability and security model →model for MTD evaluation*. The contributions include empirical results, models and analysis, and a simulation tool.

In short, the most important contributions of this thesis are:

- **Empirical observations of the system under attack-defense using MTD based on VM migration**. Unlike most model-based evaluation works, we started with a practical evaluation to learn about the system's behavior. Specifically, we conducted a series of experiments with time-based VM migration for MTD purposes to obtain knowledge to aid the model design. This way, we empirically observed the MTD effectiveness against real attacks. Results show that the continuous modification of attacker position through VM migration reduces the probability of attack success. The frequency of VM migration is the decisive factor for the survival of the system to the attack, meaning that delayed migrations may allow the attacker to reach attack success. In the studied scenarios, once the attack succeeds, the system stays down even after the departure of the attacker's VMs to another host.
- **Performability model for virtualized systems with VM migration**. We contribute to the state-of-the-art in this field by providing a comprehensive performability evaluation of a virtualized system with VM migration. The evaluation covers system availability, reliability, and service throughput. The proposed models also take the occurrence of burst workloads into account. Our results show that the software rejuvenation-induced performability improvement depends on the incidence of such workloads. In fact, the VM migration-based software rejuvenation ends up producing minor improvement in scenarios with frequent bursty workloads.
- **Metric for system security evaluation**. We propose the RiskScore metric, which characterizes security not from the perspective of the attack characteristics (i.e., attack intensity or attacker abilities) but from a system state perspective. In other words, the RiskScore metric measures the risk of attack success based on the system's current state by observing the time the system spent on risky or vulnerable states. The proposed metric is beneficial for extracting a security evaluation perspective from unaltered availability models.

- **Models for availability and security evaluation of cloud with MTD based on VM migration**. On top of the baseline model design, we included the intended MTD behavior by proposing models considering different attack tactics. Specifically, we consider threats where the VM migration totally nullifies the attack progress (i.e., *non-persistent* tactic) and threats where the attacker is able to continue the attack as soon as the attacker VM arrives in a previously visited host (i.e., *persistent* tactic). To enhance the evaluations, we propose a set of secondary metrics; *e.g.*, *tolerance levels* - to measure how long the system resists until reaching a specific probability of attack success, and *transient probability of attack success* - to measure the probability of attack success in a specific point in time. Results from the model reveal which VM migration policy is able to achieve the desired levels of system protection. We noticed that the protective effects reduce in scenarios with longer intervals between VM migrations. Depending on how long is the migration interval, the MTD may reach a negligible protection effect.
- **Tool to analyze and compare scenarios of VM migration-based MTD against *non-persistent* threats**. We developed a simulation tool to automate the execution of the proposed MTD models. Through *PyMTDEvaluator*, the user can analyze and compare various scenarios using their own set of parameters. The tool provides a detailed output, compiling the results in PDF reports.
- **Evaluation of the crossover of software rejuvenation and MTD based on VM migration**. We present a comprehensive evaluation of a cloud computing environment with a multipurpose time-based VM migration deployment (i.e., software rejuvenation and MTD). We consider the trade-offs between availability and security in the evaluations while selecting specific VM migration schedules.

1.3 How to Read this Thesis

One of our research design principles is **modularity**, meaning that each particular advancement towards the goal should produce concrete artifacts. During our work, we succeeded in publishing in different venues related to dependability and security. This thesis is a compilation of those papers. Therefore, naturally, the content of the following chapters is based on our publications.

For clarity, it is essential to highlight how the following chapters relate and cooperate to reach the thesis goal. Therefore, we present below an overview of the thesis structure, which consists of ten chapters. Each core chapter (2 to 9) presents a concrete *individual contribution* that is the main result or implication from the research that originated the chapter. Figure 1.1 summarizes those contributions (chapters 6, 7, and 8 are starred because they represent the pinnacle of our work, specifically related to our main goal of evaluating VM migration as MTD).

Chapter 2 presents background concepts and a comparison with related works. It absorbs inputs from a systematic mapping of the literature [Torquato and Vie-

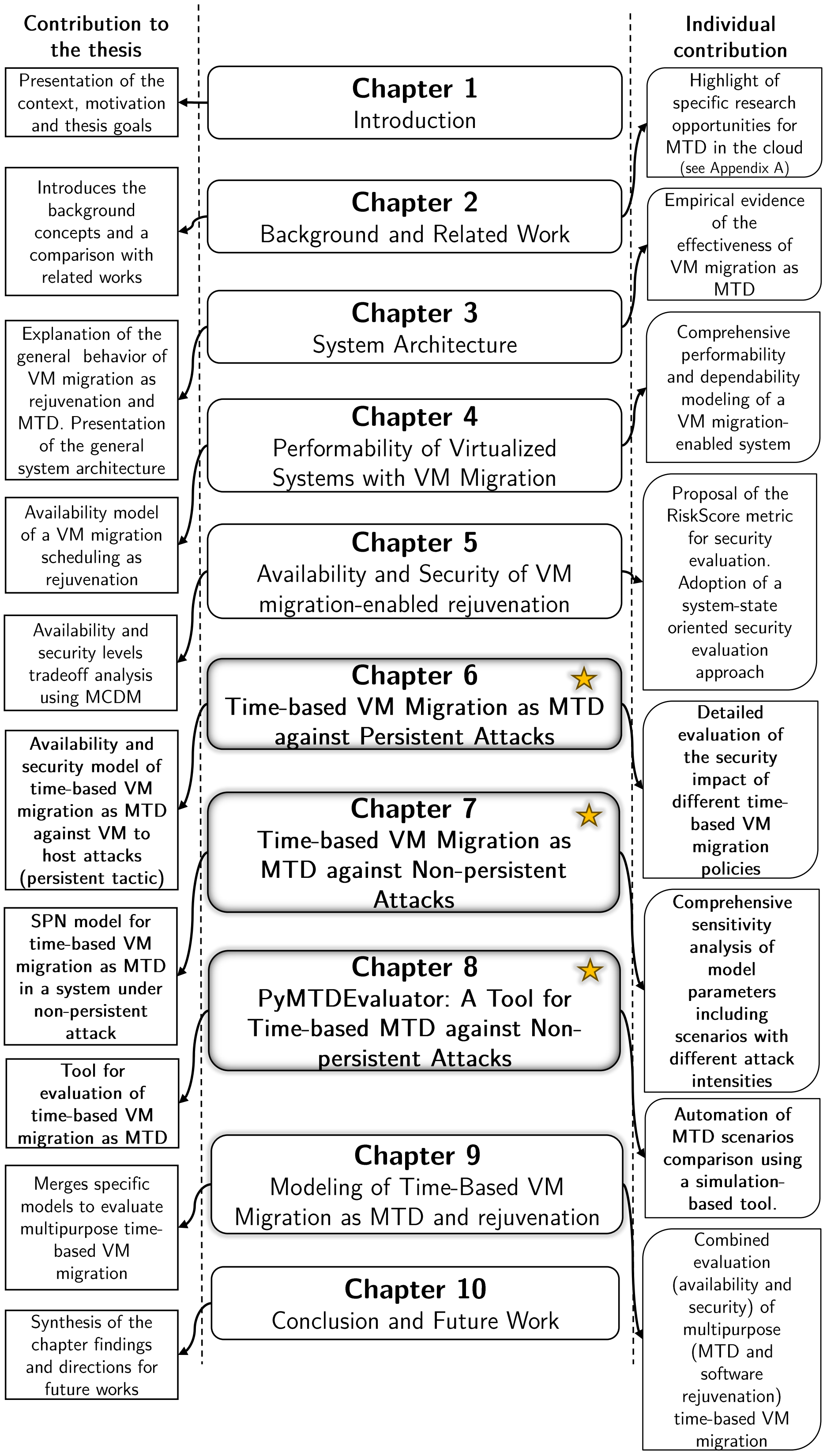

Contribution to the thesis
Individual contribution
Chapter 1
Introduction
Presentation of the context, motivation and thesis goals
Highlight of specific research opportunities for MTD in the cloud (see Appendix A)
Chapter 2
Background and Related Work
Introduces the background concepts and a comparison with related works
Empirical evidence of the effectiveness of VM migration as MTD
Chapter 3
System Architecture
Explanation of the general behavior of VM migration as rejuvenation and MTD. Presentation of the general system architecture
Comprehensive performability and dependability modeling of a VM migration-enabled system
Chapter 4
Performability of Virtualized Systems with VM Migration
Availability model of a VM migration scheduling as rejuvenation
Chapter 5
Availability and Security of VM migration-enabled rejuvenation
Proposal of the RiskScore metric for security evaluation. Adoption of a system-state oriented security evaluation approach
Availability and security levels tradeoff analysis using MCDM
Chapter 6
Time-based VM Migration as MTD against Persistent Attacks
Availability and security model of time-based VM migration as MTD against VM to host attacks (persistent tactic)
Detailed evaluation of the security impact of different time-based VM migration policies
Chapter 7
Time-based VM Migration as MTD against Non-persistent Attacks
SPN model for time-based VM migration as MTD in a system under non-persistent attack
Comprehensive sensitivity analysis of model parameters including scenarios with different attack intensities
Chapter 8
PyMTDEvaluator: A Tool for Time-based MTD against Non-persistent Attacks
Tool for evaluation of time-based VM migration as MTD
Automation of MTD scenarios comparison using a simulation-based tool.
Chapter 9
Modeling of Time-Based VM Migration as MTD and rejuvenation
Merges specific models to evaluate multipurpose time-based VM migration
Combined evaluation (availability and security) of multipurpose (MTD and software rejuvenation) time-based VM migration
Chapter 10
Conclusion and Future Work
Synthesis of the chapter findings and directions for future works

Figure 1.1: Chapter contributions

ira, 2020] and presents open cloud computing research opportunities related to

MTD.

Chapter 3 presents the system architecture. We detail the approaches of using VM migration as support for software rejuvenation and as MTD. Besides that, the chapter also provides empirical observations on the use of VM migration for both scenarios. Specifically, we observe the VM migration-based software rejuvenation effectiveness against *hypervisor* software aging, and the time-based VM migration approach effectiveness as MTD against a specific *host-based* attack (memory DoS [Zhang et al., 2016b]).

Chapter 4 presents our baseline availability model from which the other models are derived. The chapter showcases a thorough performability evaluation of a VM migration-based rejuvenation, including bursty workload occurrence and system throughput evaluation.

Chapter 5 proposes a model for availability and security evaluation of a cloud system with software rejuvenation based on VM migration. It discusses our approach to multi-criteria decision-making for availability and security analysis. One of the most relevant outputs of the chapter is the RiskScore metric, a system state-oriented security evaluation metric based on the current conditions of the system that enable or facilitate an attack.

Chapter 6 presents our first model for VM migration-based MTD that considers the *persistent* attack tactic, where the attacker can continue the attack as soon as the VM returns to a previously visited host. The chapter proposes secondary metrics for security evaluation, such as *tolerance levels* - time needed to reach a specific attack success probability, and comprehensively evaluates security levels under different scenarios.

Chapter 7 presents the evaluation of VM migration-based MTD against *non-persistent* attacks. In *non-persistent* attacks, the attacker must restart the attack after each migration. Specifically, the chapter makes use of the secondary metrics from Chapter 6 and evaluates them in this different attack scenario. The chapter also provides a comprehensive sensitivity analysis, even considering different attacker capabilities (i.e., more or less skilled attackers).

Chapter 8 presents an easy-to-use interface for the model proposed in Chapter 7. *PyMTDEvaluator* is a simulation tool that executes the model using the desired user parameters. The main feature is the possibility of automating the comparison of MTD scenarios by compiling several scenarios in a single assessment.

Chapter 9 merges the previous models to evaluate scenarios combining the use of VM migration as software rejuvenation and MTD. As a highlight, the chapter investigates the relationship between software rejuvenation and MTD when used simultaneously. To our knowledge, this is the first work to provide security and availability evaluation of such scenarios.

Chapter 10 concludes the thesis, presenting the synthesis of the contributions and findings and putting forward a set of ideas for future works.

Appendix A reproduces (in full) our systematic mapping paper [Torquato and Vieira, 2020], which is partially presented in the chapter of the background and related work (Chapter 2).

Appendix B reproduces the paper [Torquato and Vieira, 2021], which presents empirical evidence of the effectiveness VM migration as MTD in the context of a client-server application and a Machine Learning (ML) application.

Appendix C reproduces the paper [Torquato and Vieira, 2019], which presents empirical evidence of software aging effects in Docker container platform.

Appendix D presents an example of one of the outputs of *PyMTDEvaluator* tool. The referred output is a report compiling the results from the simulation runs.

To attain the **modularity** principle mentioned above, chapters 4 to 9 are intended to be self-contained. Except for the system architecture presented in Chapter 3, they present enough details by themselves without requiring reading the previous chapters. Nevertheless, these chapters are not isolated pieces; they connect by providing inputs, models, or insights that will be used in developing other chapters. Figure 1.2 highlights the relationship between chapters and the papers that originated them.

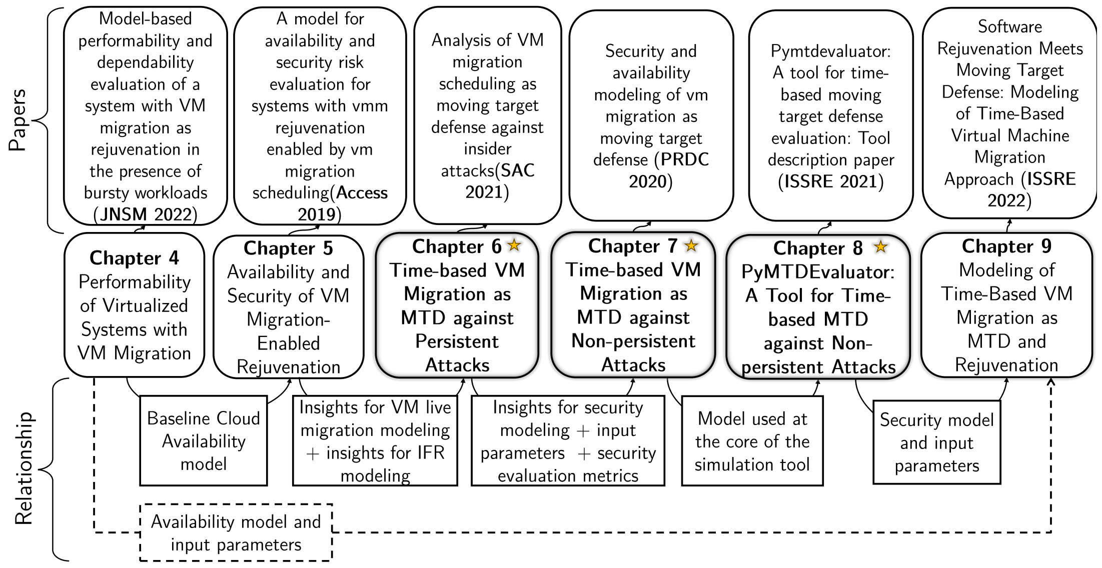

Papers
Model-based performability and dependability evaluation of a system with VM migration as rejuvenation in the presence of bursty workloads (JNSM 2022)
A model for availability and security risk evaluation for systems with vmm rejuvenation enabled by vm migration scheduling(Access 2019)
Analysis of VM migration scheduling as moving target defense against insider attacks(SAC 2021)
Security and availability modeling of vm migration as moving target defense (PRDC 2020)
Pymtdevaluator: A tool for time-based moving target defense evaluation: Tool description paper (ISSRE 2021)
Software Rejuvenation Meets Moving Target Defense: Modeling of Time-Based Virtual Machine Migration Approach (ISSRE 2022)
Chapter 4 Performability of Virtualized Systems with VM Migration
Chapter 5 Availability and Security of VM Migration-Enabled Rejuvenation
Chapter 6 ☆ Time-based VM Migration as MTD against Persistent Attacks
Chapter 7 ☆ Time-based VM Migration as MTD against Non-persistent Attacks
Chapter 8 ☆ PyMTDEvaluator: A Tool for Time-based MTD against Non-persistent Attacks
Chapter 9 Modeling of Time-Based VM Migration as MTD and Rejuvenation
Relationship
Baseline Cloud Availability model
Insights for VM live migration modeling + insights for IFR modeling
Insights for security modeling + input parameters + security evaluation metrics
Model used at the core of the simulation tool
Security model and input parameters
Availability model and input parameters

Figure 1.2: Relation between chapters 4 to 9 and the produced papers

This thesis investigates time-based VM migration while applied as **Software Rejuvenation** and **MTD**. At the time of this thesis writing, the intersection of these two fields was relatively new. As some readers may be interested in only one of the fields, we propose the following reading paths. Nevertheless, we recommend all readers to consider Chapter 9, as it introduces the evaluation of a multipurpose deployment of time-based VM migration considering both fields.

- **Software rejuvenation**
  - Chapter 1 → Chapter 2 → Chapter 3 → Chapter 4 → Chapter 5 → Chapter 9 → Chapter 10;
- **Moving Target Defense**

  - Chapter 1 → Chapter 2 → Chapter 3 → Chapter 6 → Chapter 7 → Chapter 8 → Chapter 9 → Chapter 10;

- **Software Rejuvenation + Moving Target Defense**
  - All the chapters.

1.4 Disambiguation

This thesis document is a result of combining our previous works into a single document. As mentioned earlier, the chapters are intended to be *self-contained*. Here, *self-contained* means that except for the basic architecture presented in Chapter 3, the reader might be able to follow each chapter without requiring the reading of other chapters. However, as each chapter presents its context, some terms may appear ambiguous or non-consistent, considering the thesis as a single document. Below, we present a non-exhaustive list of these terms. We hope that the explanation suffices to mitigate possible misunderstandings:

- **Tactic, attack and threat** - the security evaluation chapters presents these terms associated with *persistent* and *non-persistent* concepts. In the scope of this document, for variety and to avoid excessive repetition, we used these terms interchangeably. However, they have slightly different meanings: *tactic* - refers to the strategy used in the attack deployment; *attack* - actual action against the system; and *threat* - potential attack.
- ***Insider*, host-based, VM to host, VM to hypervisor, VM escape, resource starvation, Memory Denial of Service (memory DoS)**- these terms are used in the chapters when referring to attacks. They have their particular meanings, but overall, in the scope of this thesis, they serve to highlight an attack coming from an attacker VM targeting a resource in the same PM host. Their slightly different meanings are: *insider* - attack coming from an authorized user; *host-based* - attack occurring inside the PM; *VM to host* - similar to *host-based*; *VM to hypervisor* - attack targeting the hypervisor component; *VM escape* - attacker tries to break VM isolation to affect or control the underlying host; *resource starvation* - abusive use of the resources to induce lack of resources for other users or VMs; and *Memory* Denial of Service (DoS) - specific *resource starvation* attack targeting the main memory of the underlying host.
- **SPN and Stochastic Reward Net (SRN)** - the original versions of our papers use the term SRN. Nevertheless, after further consideration, we decided to use SPN instead. The reason is twofold: i) to emphasize the **Petri Nets** context of our models, and ii) to avoid misleading information about the tool-set we used. For all the PN-based models, we used the TimeNET tool [Zimmermann, 2017], which is a tool for SPN evaluation. More details about SPN characteristics are presented in Chapter 2.

Chapter 2

Background and Related Work

This thesis falls into the intersection of dependability and security evaluation. Through analytical models, our research considers several scenarios for evaluating a virtualized system with time-based VM migration as MTD policy. Besides that, we dedicate a considerable amount of work to cover baseline aspects, including software aging and rejuvenation.

In the following sections, we present background concepts related to *Availability and security evaluation* (Section 2.1), concepts on *State Space Models* (Section 2.2), and *Software Aging and Rejuvenation* (Section 2.3). Then, we add two sections related to MTD: the first presents the general concepts of the technique with a summary of results of a systematic mapping of the literature (Section 2.4), while the second presents the specific context of MTD based on VM migration (Section 2.5). We also include a brief overview of Multi Criteria Decision Making (MCDM) methods, as these are used in some of our contributions (Section 2.6). Finally, we analyze and compare related works (Section 2.7).

2.1 Availability and Security Evaluation

Events affecting system availability and security (e.g., outages, crashes, and attacks) are usually unexpected. Therefore, setting up a reliable and feasible approach to system availability and security evaluation through measurements is a challenging task. Accelerated life testing [Nelson, 1980] and fault injection [Hsueh et al., 1997] are alternatives to solve the evaluation of the dependability-related metrics. These alternatives allow the insertion of specific actions to induce a failure-prone behavior. There are a lot of benefits in *accelerated* experiments; for example, observing the system reaction after a fault activation and observing which faults lead the system to failures. With adequately designed experiments, these techniques also help compute the system availability. However, using such techniques usually alters the internal state of the system (e.g., forcing it into a faulty behavior): performing a security evaluation in a system already in an altered state could produce misleading results as these conditions may not repeat in a real-world scenario. Another problem is security evaluation itself: it may require vulnerability, attack, or intrusion injection [Fonseca et al., 2013][Gonçalves et al., 2023], for which we do not have sufficiently mature and generic techniques and tools nowadays.

Model-based evaluation is an alternative solution for the dependability and security combined assessment [Nicol et al., 2004]. Models provide flexibility to exercise different scenarios and do not require direct intervention in a running system [Jain, 1990]. However, their results must be considered carefully, as they produce only approximated results that depend on the input parameters. The fine-tuning of the parameters and a proper sensitivity analysis are of utmost importance in a comprehensive model-based evaluation. From a more generic perspective, model-based evaluation helps understand the expected behavior of the metrics while the input parameters change without interacting with a real system.

In the realm of models for dependability and security evaluation, it is possible to highlight the following three types: i) combinatorial models (Reliability Block Diagram (RBD), Attack Tree (AT), Fault Tree (FT)), ii) state-space models (SPN, Continuous Time Markov Chain (CTMC)), and iii) hierarchical compositions (e.g., RBD in the upper level and CTMC in the lower level) [Trivedi et al., 2009]. During the development of this thesis, we adopted state-space models. This way, combinatorial models are not included in this section on background concepts. We recommend the books [Maciel, 2023] and [Trivedi and Bobbio, 2017] for a more thorough review of such models.

Combinatorial models assume independence of the system components. Thus, in the evaluation process, a component state change does not affect the state of the other components, meaning that, in a scenario with complex dependencies between the modeled components, the combinatorial models are not suitable for the evaluation. Alternatively, state-space models can represent the interaction of the components. In the modeling of cloud and virtualized environments, we can leverage both model types in a hierarchical composition as presented in our previous work [Torquato et al., 2019a].

Due to the complexity of our target systems, we decided to adopt state-space models, as the combinatorial models cannot capture component dependency. Our approach uses models capable of evaluating both metrics (availability and security), resulting in a unified evaluation. Also, such models take account of component dependency (e.g., a VM depends on its physical machine host to run) and interactions (e.g., a VM migration from one physical machine to another). Due to its features, we selected a space model based on SPNs. SPNs are a subtype of PNs, which provides a subset of needed features for our evaluation [Marsan, 1988]. Section 2.2 provides additional details of SPN models. Previous research highlighted the application of PN models for availability [Melo et al., 2013a] and security [Wang et al., 2012].

For the scope of this thesis work, we focus mainly on evaluating two metrics: availability and security. Availability is the system property related to the readiness for correct service delivery [Avizienis et al., 2004]. We usually measure availability (A) as a proportion of the time that the system is ready to deliver the correct service (*uptime*) in its complete life-cycle (i.e., *uptime* + *downtime*). Thus, we can obtain availability using the equation 2.1.

$$A = \frac{uptime}{uptime + downtime} \tag{2.1}$$

From the security perspective, we used several metrics in the evaluation process. Firstly, we propose a metric related to the security risk. Security risk evaluation is usually part of a technical management process known as *risk management* [Ross et al., 2016]. Usually, security risk evaluation takes account of two system aspects: the system conditions that enable (or improve the chance of) an attack success (e.g., in general, a system with a disabled firewall is exposed to a higher security risk than a system with an enabled firewall), and the importance of the asset hosted by the system (e.g., generally, a virtual machine that runs applications to support credit card operations is exposed to a higher security risk than a virtual machine that runs a simple calculator). In Chapter 5, we propose the RiskScore metric, which focuses on the first aspect of security risk evaluation - assess how much the internal state of the system may favor (or enable) an attack success.

For the MTD evaluation, we mainly use the probability of attack success, representing the chance of the ongoing attack reaching its objective. Additionally, we propose a set of secondary metrics, which are detailed at Chapters 6, 7, and 9. For example, the *Tolerance Level* - metric which measures how long the system takes to reach a specific probability of attack success, the *Effectiveness Limit* - point in time where the MTD does not produce defensive effects, and the *Increased Resistance* - protection levels comparison of the systems with and without MTD.

2.2 Concepts on State Space Models

The models presented in this thesis are based on PN. A necessary introduction for PN must start from the Markov Chain (MC) formalism. The MCs are an essential element of the SPN-based evaluation methods. Roughly, the solution of SPN models involves mapping them into MC models [Bobbio, 1990]. For that reason, the first subsection below presents the fundamental concepts of MCs. The rest of the sections are as follows. Section 2.2.2 presents the concepts of PN, Section 2.2.3 discusses the particularities of SPN, and Section 2.2.4 introduces the hierarchical model composition. Finally, Section 2.2.5 closes, presenting a concise view of how to apply sensitivity analysis in the model-based evaluations.

2.2.1 Markov Chains

Proposed in 1907 by Andrei Andreevich Markov, the MC have been widely adopted for dependability evaluation since the fifties [Maciel et al., 2010]. The first step in their knowledge is to understand the stochastic processes.

A stochastic process is a set of random variables $X(t)$ in a sample space. The variables $X(t)$ values are the *states*. The set of all possible *states* is the *state space*. The *state space* can be discrete or continuous. In the former case, the stochastic process is a *chain*.

Following the Markovian definitions [Ching and Ng, 2006], if the state of a stochastic process is only dependent on its immediate predecessor, then the stochastic process is a *Markov process*. From a perspective, the current state stores all the data needed to determine the next state. Markov processes with discrete (i.e., countable) *state space* are known as MCs. Here, we have two additional classifications, Discrete Time Markov Chain (DTMC), where the parameter space of the MC is discrete, and CTMC, with a continuous parameter space.

Graphically, the MC diagrams consist of directed graphs in which the nodes represent the system states, and the arcs represent the transitions between states. Each arc has a label to assign the probability or rate related to the events leading to a state transition.

Under the assumption of a CTMC for availability evaluation, the transitions occur observing a rate instead of a probability [Guedes, 2019]. The usual way to derive steady-state and transient CTMC solutions is through its transition matrix Q (i.e., infinitesimal generator matrix). The elements in the Q matrix are related to the transitions between the states. And the elements in the main diagonal are equal to the negative sum of the rest of the elements in the line (i.e., $q_{ii} = -\sum_{j:j\neq i} q_{ij}$) [Trivedi and Bobbio, 2017].

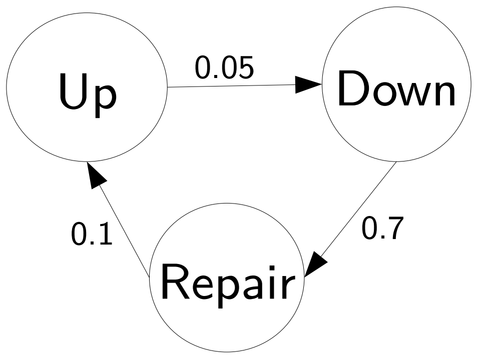


Figure 2.1: CTMC availability model, retrieved from [Guedes, 2019]

Consider, for example, the CTMC model in Figure 2.1. Where the rates represent failures (transition from *Up* to *Down*), detection (transition from *Down* to *Repair*), and repairs (transition from *Repair* to *Up*) per second. In this case, its generator matrix Q, considering the *state space* $S = Up, Down, Repair = 0, 1, 2$ is as follows.

$$Q = \begin{pmatrix} q_{00} & q_{01} & q_{02} \\ q_{10} & q_{11} & q_{12} \\ q_{20} & q_{21} & q_{22} \end{pmatrix} = \begin{pmatrix} -0.05 & 0.05 & 0 \\ 0 & -0.7 & 0.7 \\ 0.1 & 0.1 & -0.1 \end{pmatrix}$$

Through the CTMC solution methods, which are omitted here for the sake of simplicity, we find the following equations for the transient (Equation 2.2) and steady-state solution (Equation 2.3), respectively. More details of CTMC solution methods are in reference [Bolch et al., 2006].

$$\pi'(t) = \pi(t)Q, \; given\, \pi(0) \tag{2.2}$$

$$\pi Q = 0, \sum_{i \in S} \pi_i = 1 \tag{2.3}$$

In the CTMCs, all the transitions follow the exponential distribution. The exponential-only behavior may be a limitation for some systems evaluation, as the events may occur in different distributions. To mitigate this problem, we can apply phase approximation, using specific structures to represent different probability distributions [Trivedi, 2008].

2.2.2 Petri Nets

Since their original proposal by Carl Adam Petri in 1962 [Petri, 1962], the PNs have been used in various scenarios. Ranging from electrical engineering [Castellanos Contreras and Rodríguez Urrego, 2023], telecommunication systems [Boubour et al., 1997] to computer science and manufacturing [Pawlewski, 2012], the mathematical formalism serves as support for system analysis and evaluation.

The PNs models provide a powerful abstraction of the studied system. In general, the graphical representation of PN makes it possible to understand the system behavior and the interactions between the system's components. Due to its popularity, there are nowadays several tools to draw, analyze, and execute PN-based models [Zimmermann, 2017][Maciel et al., 2017][Paolieri et al., 2019]. The graphical representation of PN has four main components, as presented in Figure 2.2. In summary, the elements refer to:

- **Token** - stored inside the places, they move accordingly to the transition firing;
- **Places** - are the passive element of the model. They offer and collect tokens from the transitions;
- **Transitions** - represents the event occurrence in the system. Their firing depends on some pre-conditions determined by the input arcs;
- **Arcs** - directed edges indicating the token flow in the net. Depending on the scenario, they may have a weight to indicate how many tokens are flowing upon the transitions firing.

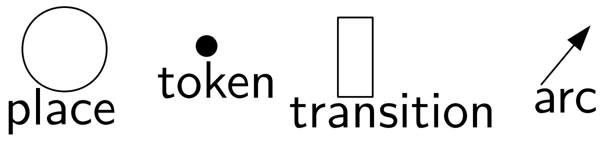


Figure 2.2: Petri Net components

Formally, a PN is a 5-tuple $PN = (P, T, Pre, Post, M_0)$, where: $P = p_1, p_2, ..., p_n$ is a finite set of places, $T = t_1, t_2, ..., t_n$ is a finite set of transitions, $Pre : P \times T \rightarrow N$ is the input incidence function, $Post : P \times P \rightarrow N$ is the output incidence function, and $M_0 : P \rightarrow N$ is the initial marking. Marking is the distribution of tokens in the net places [Küngas, 2005].

The PN dynamics depend on the occurrence of transition firing. A transition firing changes the net state. A transition is enabled if the corresponding marking M provides enough tokens for the respective input arcs (i.e., $\forall p \in P, M(p) \geq Pre(p,t)$, where $Pre(p,t)$ is the weight of the $arc(p,t)$).

A specific marking M_x is reachable from a given marking M' if a sequence of transitions s is capable of bringing the net from M' to M_x. In the PN realm, the problem of finding a specific s from the initial marking M_0 to a hypothetical marking M_x is known as *reachability problem*.

2.2.3 Stochastic Petri Nets

The original concept of PN lacks the notion of timed transitions. The introduction of time events and parameters in the PN results in a timed Petri Net. Precisely, where all the times and delays considered in the modeling obey the exponential distribution, we have a SPN. SPN evaluation process comprises the transformation of the SPN into a CTMC model. Specifically, after exploring all the reachable markings, we produce a *reachability graph*. The *reachability graph* structure is indeed a CTMC where the nodes are the net markings, and the transitions are related to the SPN transitions triggering. In the past years, the SPN models have been extensively used in the dependability evaluation [Trivedi and Bobbio, 2017][Maciel, 2023].

Throughout the years, the development and adaptation of SPN models resulted in several new concepts as Generalized SPN, Extended Deterministic SPN and SRN [Heiner et al., 2009][German, 2000][Muppala and Trivedi, 1991]. Therefore, for the sake of simplicity, we decided to stick to the term SPN. In this thesis, we consider that the models: i) can have immediate transitions (i.e., firing with zero time); ii) have inhibitor arcs - arcs that prevent (instead of enabling) transition firing; iii) can have deterministic transitions (i.e., transitions which follow the deterministic distribution instead of exponential distribution); iv) are able to represent transition firing priority, and v) provide guard functions features (i.e., additional conditions, besides the input arcs, to enable transition firing). In a more advanced modeling technique presented in Chapter 4, we also consider *marking-dependent firing rates*, i.e., dynamic adjustments in the transition parameters depending on the state of the net.

Let us consider the flow of a simple SPN availability model in Figure 2.3. In the initial state, the system is running, presented by the token in the `UP` place. The transition `MTTF` represents the system Mean Time to Failure (MTTF). `MTTF` firing represents a system failure occurrence. The same transition moves the token from `UP` place to the `DW` place. The system repair is represented by the `MTTR` transition (Mean Time to Repair (MTTR)). `MTTR` transition firing returns the model to its initial state.

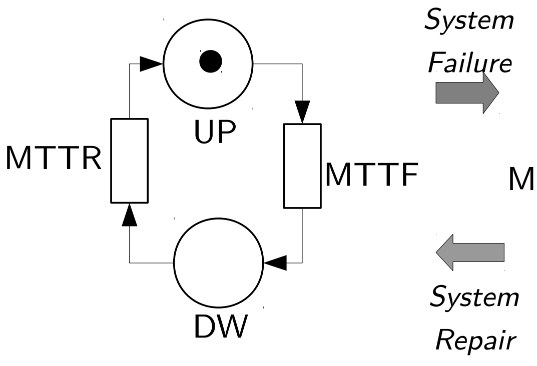


Figure 2.3: Flow of a SPN simple availability model

We can compute the system availability using the following reward measure *Availability* $= P\{UP > 0\}$, which captures the probability of tokens presence in the `UP` place.

2.2.4 Hierarchical Compositions

It is possible to adopt multi-layered models to describe more complex systems. These multi-layered models obey a hierarchy. For that reason, they are usually known as *hierarchical* models. Throughout the years, the *hierarchical* models have been extensively used in availability and reliability evaluation [Ammar et al., 1987][Clemente et al., 2022][Nguyen et al., 2019][Trivedi and Bobbio, 2017].

In the hierarchical models, the bottom-layer model solutions or extracted metrics are used as input for the top-layer models. It is possible to have multiple layers with heterogeneous model composition. For example, in our previous work [Melo et al., 2013b], we used a composition of RBD at the bottom layer and SPN at the top layer to model the VM migration as support for software rejuvenation problem. We presented another approach for the same problem in the paper [Torquato and Vieira, 2018]. In that one, we used SRN models in two layers. The same idea is applied in the Chapter 4 of this thesis. Finally, it is worth highlighting one of the previous studies that focused on the availability evaluation of large cloud computing environments [Torquato et al., 2019a]. In that model, we mixed RBD and SRN models.

Hierarchical modeling allows the separation of the sub-component or subsystem in a dedicated model. Therefore, applying modifications without changing the entire model structure is possible, leading to easier model scalability. In general, hierarchical modeling decreases the solution obtainment time, as presented in our previous paper [Torquato and Vieira, 2018].

Even with increasing computing power, solving large models may be impracticable. As stated previously, the solution method for SPN involves the obtainment of the *reachability graph*. Large SPN models tend to suffer from the *state-space explosion* problem. Any complex system, for example, cloud computing, has numerous states. Usually, the number of states grows exponentially with the number of considered variables, components, and processes [Valmari, 1996][Muppala and Lin, 1996][Kot, 2003]. In such cases, obtaining the *reachability graph* may be prohibitive or impracticable.

Longo et al. [2011] highlighted an interesting problem solution through hierarchical composition. The composition is a bit more advanced in their case, as the layers have cyclic dependencies. As they are applying SRN models, this specific model composition is named *interacting models*. In their paper, they show that they cannot compute large scenarios without the use of the composition. Their results support the use of hierarchical compositions not only to simplify some scenarios but also to enable the solution obtainment of highly complex models.

2.2.5 Sensitivity Analysis

Sensitivity analysis methods aim to understand how much a parameter variation or a set of parameters affects the model output [Mainkar et al., 1993]. This type of analysis, widely adopted in dependability models, helps find possible bottlenecks and barriers in the system behavior [Matos et al., 2012c].

There are a variety of methods for sensitivity analysis, such as differential analysis, correlation analysis, regression, or perturbation analysis [Iooss and Lemaître, 2015]. Nevertheless, in the scope of this thesis, we used the most straightforward method, the *graphical sensitivity analysis*. We use *graphical sensitivity analysis* methods to obtain visual indicators of the parameter variation effect in the observed metric [Christopher Frey and Patil, 2002]. The visual indication comes from the model's output plots.

2.3 Software Aging and Rejuvenation

A software may pass an extensive test phase before its release version [Pan, 1999]. The test phase is an essential step in the software development process and usually encompasses the verification of the software behavior under different workloads and conditions [Royce, 1970]. However, some bugs and errors may go undetected even after extensive testing [Grottke and Trivedi, 2007]. The phenomenon of bug activation may occur after a long time of execution of the software. Therefore, defining the roots of the faults and failures to solve the problem becomes difficult. These types of bugs are known as *Mandelbugs* [Grottke and Trivedi, 2005] [Gray and Reuter, 1993].

The *aging-related* bugs have a similar nature to that *Mandelbugs*. Nevertheless, the *aging-related* bugs activation depends on specific system conditions (e.g., lack of computational resources) which are difficult to reproduce [Vaidyanathan and Trivedi, 2001]. Following the classical definition of the *chain of threats* from Avizienis et al. [2004], Grottke et al. [2008] proposed a "*chain of threats*" for Aging Related (AR) failures (see Figure 2.4).

The AR failure occurs from the propagation of AR errors accumulation in conjunction with specific system-internal environment conditions. The AR failure occurrence has an increasing probability as long as the system runs. Specifically,

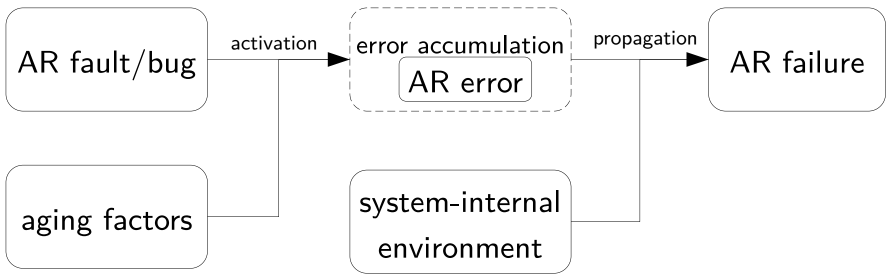


Figure 2.4: *Chain of threats* for AR failure - retrieved from [Grottke et al., 2008]

the AR errors that do not yet cause a AR failure accumulate in the internal state of the system in the presence of successive AR faults. The accumulation of AR errors leads the system to a state where they propagate into a AR failure. The aging factors are the activation patterns that trigger an AR bug.

One of the most critical metrics in software aging studies is the Time To Aging Related Failure (TTARF). The TTARF is the expected time between system startup and an AR failure occurrence. The TTARF probability distribution is highly influenced by the workload submitted to the system [Bovenzi et al., 2011][Cotroneo et al., 2014]. Specifically, the intensity of the workload may accelerate or enable the occurrence of aging factors.

AR bugs activation leads the system to a degraded state. The degraded state is the result of the accumulation of software aging effects. The software aging effects bring the system from a reliable condition to a failure-probable state. The software aging effects may generally be perceived even before a AR failure. For example, software aging effects may be related to resource leakage, numerical error accrual, and data corruption. Finally, if no countermeasure is taken, the system reaches the AR failure. Figure 2.5 depicts the general behavior of a system under software aging.

Huang et al. [1995] presented the first definitions of software rejuvenation. Their paper defines software rejuvenation as a proactive technique to prevent aging effects from reaching critical levels. The rejuvenation actions rely on gracefully terminating and restarting an application to conduct it to a clean state without accumulation of effects of aging. Standard techniques usually consist of an operating system reboot or application restart. Therefore, in the usual scenario, software rejuvenation imposes system downtime.

Software rejuvenation may be triggered using periodic intervals obtained from analytical models. The main goal is to improve the overall system availability by controlling the accumulated downtime due to software rejuvenation action in relation to the possible system downtime due AR failures [Grottke et al., 2008].

Previous research shows that **cloud software is liable to suffer from AR bugs**. The works from Araujo et. al. [Araujo et al., 2011] [Araujo et al., 2014] [Matos et al., 2012b] highlighted AR effects in the Virtual Infrastructue Manager (VIM) software. The VIM software is responsible for managing the cloud platform in a macro perspective, including the PMs, VMs, and other virtualized resources

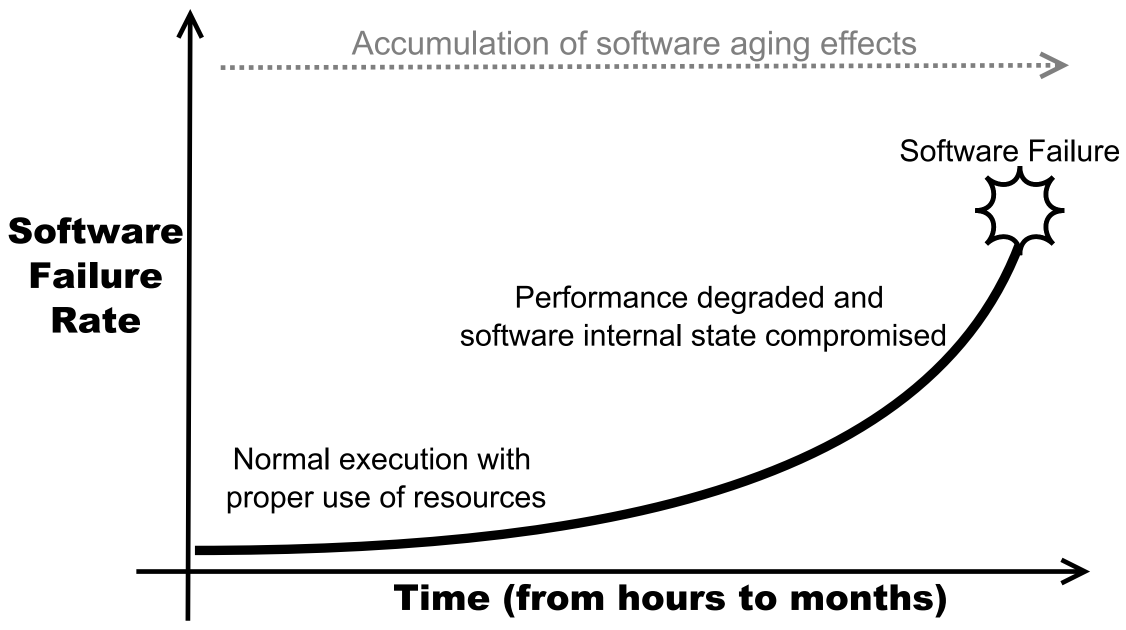

Accumulation of software aging effects
Software Failure
Software Failure Rate
Performance degraded and software internal state compromised
Normal execution with proper use of resources
Time (from hours to months)

Figure 2.5: Software aging general behavior

(e.g., OpenStack, OpenNebula). Specifically, they detected that the software for managing the VM storage, as well as the software in charge of managing the VM life cycle, suffer from software aging.

Matos et al. [2012a] investigated software aging evidence in the Virtual Machine Monitor (VMM)[1] software. The VMM software is the middleware between the VM and the underlying PM. In their work, they found AR effects in Kernel Virtual Machine (KVM) *hypervisor*. Machida et al. [2012] also investigated AR effects, but in another VMM, in this case, the Xen *hypervisor*.

More recently, software aging has also been detected in virtualized containers and in applications running in the cloud. Due to the relevance of this topic, one of the research efforts was devoted to conducting an investigation in this field. The investigation resulted in a previous work [Torquato and Vieira, 2019], which suggests AR effects in Docker container software. The evidence was further confirmed by other authors in the field [Vinícius et al., 2022][Oliveira et al., 2020]. Costa et al. [2023] show AR effects evidence in Kubernetes software. Besides that, Andrade et al. [2021] show AR evidence in image classifiers running in the cloud.

As a final remark, all the cloud computing software may suffer from AR effects coming from the underlying OS platform. Previous studies highlighted such effects in the Linux OS [Cotroneo et al., 2010][Matias et al., 2010]. The effects are also present in Windows OS software [Umesh et al., 2017].

The papers highlighted above are only a partial observation of AR bugs evidence in cloud computing software. We notice that, across the cloud layers, the literature suggests AR bugs existence. Therefore, the proposal of proper software

[1] In this thesis, we use the terms VMM and *hypervisor* interchangeably.

rejuvenation actions for such platforms is of utmost importance.

Due to the complexity of cloud computing software, it is usual to propose software rejuvenation actions according to the cloud layers [Alonso and Trivedi, 2015]. Here, we highlight four layers to consider in cloud computing rejuvenation actions (see Figure 2.6). This layer organization suggests that the software rejuvenation of the bottom layers also affects the upper layers. For example, OS software rejuvenation actions usually comprise the rejuvenation of the *hypervisor*, VM, and the application running inside the VM. Usually, the software aging effects of a bottom-layer component end up affecting upper-layer components. For example, *hypervisor* AR effects affect the performance of applications running in a VM.

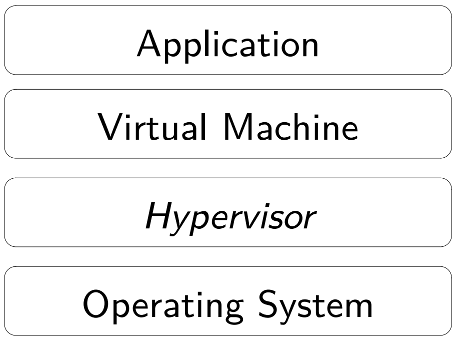


Figure 2.6: Cloud computing layers for software rejuvenation

We can highlight the following rejuvenation actions for the proposed layers:

- **Application** - restart the application and its related modules.
- **Virtual Machine** - complete re-initialization of VM, including its applications and OS.
- ***Hypervisor*** - restart the *hypervisor* software. Note that the *hypervisor* restart implies the hosted VMs downtime.
- **OS** - reboot the OS. This action usually requires firmware restart and operating system, a re-run of Power On Self Test (POST), and a reboot of the kernel and its services. Usually, this type of rejuvenation is able to completely clean up the accumulated AR effects in the platform.

The usual rejuvenation actions presented above may impose unacceptable downtime for more critical applications. Therefore, some alternatives are available to reduce rejuvenation-related downtime. One of these alternative techniques is the *microreboot* or *microrejuvenation* [Candea et al., 2004] [Cotroneo et al., 2022]. The *microreboot* consists of selecting specific service elements to restart instead of the entire application or OS.

For the bottom layers (i.e., *hypervisor* and OS), we can leverage Virtual Machine migration to reduce rejuvenation-related downtime. With its roots in the *warm-VM* reboot approach [Kourai and Chiba, 2010], which consists of suspending and resuming VMs, the VM migration-based rejuvenation aims at moving VMs away from the host under aging before rebooting or restarting the system. Previous works show the applicability of VM migration-based rejuvenation from theoretical [Bai et al., 2020] and practical [Torquato et al., 2018a] perspectives.

Previous studies highlight the analytical modeling approach to evaluate software aging and rejuvenation impact[Trivedi et al., 2000] [Garg et al., 1995] [Vaidyanathan and Trivedi, 2005]. Some papers propose scheduling of software rejuvenation actions [Machida et al., 2013] [Melo et al., 2013a] to determine when to perform rejuvenation actions to maximize overall system availability.

2.4 Moving Target Defense: A Systematic Mapping

Moving Target Defense (MTD) consists of a system reconfiguration to dynamically change the attack surface available for malicious users (e.g., VM migration, Internet Protocol (IP) address shuffling). MTD relies on environment reconfiguration to confuse attackers and nullify the knowledge of the attackers about the system state or to react to a detected attack. Basically, its foundation is to assume that the perfect system security is unattainable [U.S. Department Homeland of Security , 2020]. MTD aims to produce an attack surface that appears chaotic [Chong et al., 2009]. Therefore, MTD substantially increases the effort of the attacker to conduct the attack as the target environment is changing continuously [Jajodia et al., 2011]. MTD actions triggering usually follow scheduling with fixed or variable intervals [Sengupta et al., 2020].

In order to better understand the challenges and current limitations on the MTD in the cloud research area, we conducted a structured analysis of the state-of-the-art. This analysis follows the systematic mapping approach [Petersen et al., 2008][Petersen et al., 2015]. The systematic mapping aims to bring an overview of such state-of-the-art, usually focusing on specific aspects of the literature. Different from systematic reviews, which discuss the related papers in detail, the systematic mappings focus on particular elements and perform a more concise analysis of the papers. The main advantage of systematic mappings is to usually provide faster results, as they focus on the key aspects of the papers analyzed. Besides that, the visual data (maps) facilitates the understanding of the state of the literature. Below, we bring the highlights of our systematic mapping findings. The entire content was published in [Torquato and Vieira, 2020] and is also provided in Appendix A.

The main goal of the systematic mapping study is to provide an overview of recent research on Moving Target Defense mechanisms in cloud computing environments. The main research question we aim to answer is: *What are the most researched techniques for moving target defense on cloud computing?* In the process of answering this question, we find relevant information about the current status of the MTD research.

We considered five major computer science-related databases (i.e., ACM Digital Library, IEEE Xplore Digital Library, ScienceDirect, SpringerLink, and Online Wiley Library) in which we used the following search string *"moving target defense" AND "cloud"*. The search was made in July 2019 and considered the papers published in the past ten years. After the screening and selection of the papers, we ended up with 95 papers to classify.

2.4.1 Classification of Works

The first task after the papers screening is the proposal of classifications for the MTD research. Following the classification presented in [Cai et al., 2016b] and [Okhravi et al., 2013], we propose a classification with three categories: i) MTD research area; ii) MTD strategy; iii) evaluation metrics. These categories aim to cover the meaningful aspects of each paper, considering our research questions. Our classification approach is transverse in all the proposed categories, meaning that a paper classification can comprise more than one group of each proposed category. The details of each category are discussed in the following paragraphs.

2.4.1.1 Moving Target Defense (MTD) Research Area

In this category, we aim to classify the type of research published by observing the classification proposed by [Cai et al., 2016b]. The category has three groups: Theory, Strategy, and Evaluation, as follows:

- **Theory** - Find answers to fundamental questions regarding MTD techniques.
  - How to create effective MTD system?
  - What capabilities and features are essential to MTD systems?
- **Strategy** - Propose a technique for MTD.
- **Evaluation** - Measure the effectiveness of existing (or proposed) strategies.

As mentioned before, we consider that a paper can be classified into more than one category. For example, some papers propose and evaluate a mechanism for MTD on the cloud. Therefore, these papers are classified as *Strategy + Evaluation (S+E).*

2.4.1.2 Moving Target Defense (MTD) Strategies

While the research area category is quite generic, we assume that MTD strategies are more focused on cloud computing environments. Here, we highlight two proposed classifications present in the current literature.

Hong and Kim [2015] proposes a classification for MTD actions. They suggest three different categories. *Shuffle* - reshuffle of a system configuration (e.g., IP address or port reassignment through software-defined networking). *Diversity* - replace a system component with a variant able to perform the same task (e.g., VM migration between heterogeneous hypervisors). *Redundancy* - enlargement of attack surface to deal with incoming attacks (e.g., VM replication).

Okhravi et al. [2013] also propose the following taxonomy for MTD techniques:

- **Dynamic Runtime Environment** - Techniques that focus on changing the environment used by the applications in a dynamic manner.
  - **Address Space Randomization** - Changing the memory layout. The randomization may include the location of the program code, libraries, and other elements stored in the memory.
  - **Instruction Set Randomization** - Changing the OS interface presented to the application.
- **Dynamic Software** - It consists of changing the software code dynamically. Specifically, it is possible to change the instructions, their order, grouping, and format.
- **Dynamic Data** - It aims at changing the application data dynamically (e.g., format, syntax, encoding, representation).
- **Dynamic Networks** - It alters the network configuration and properties dynamically (i.e., network addresses and ports).
- **Dynamic Platform** - Changing the platform properties as OS, Central Processing Unit (CPU) dynamically. In a virtualized setup, it is possible to achieve *dynamic platform* MTD through VM migration across different hypervisors.

Observing the current literature, we decided to simplify the classification presented in [Okhravi et al., 2013]: ***Dynamic Application*** - which comprises dynamic *data*, dynamic *software*, and dynamic *runtime environment*; ***Dynamic Platform*** and ***Dynamic Network***.

2.4.1.3 Evaluation Metrics

The deployment of an MTD mechanism implies costs for system performance while improving the system security. Therefore, we intend to understand which evaluation metrics are currently used in MTD in cloud research. We propose only two groups for this category:

- ***Performance***, evaluation comprises performance metrics, such as response time, system overhead, *etc*;
- ***Security***, evaluation covers security aspects, such as attack success rate, survivability, *etc.*

2.4.2 Mapping Results

This section provides an overview of Moving Target Defense in cloud computing research. We present charts and diagrams with the distribution of publications regarding the classification mentioned earlier.

2.4.2.1 Research Area - Papers Distribution

The first map is a Venn Diagram of the distribution of papers in terms of theory, strategy, and evaluation (see Figure 2.7). We noticed that most papers propose an MTD strategy and present its assessment. As we have books and seminal papers to support MTD theory [Jajodia et al., 2011, 2012; Zhuang et al., 2014], the scientific community seeks to offer more approaches to enhance the available set of MTD mechanisms. However, theoretical papers usually provide more generic contributions on the use of MTD in other scenarios (not only cloud computing). For example, Leslie et al. [2015] proposes a model based on game theory to support resource configuration to reduce the likelihood of a security attack. Lei et al. [2018] also suggests an approach based on game theory. Their proposal consists of an incomplete information Markov game theory comprising a moving attack surface and optimal strategy selection. The papers [Song et al., 2019], [Peng et al., 2014], and [Wang et al., 2016] are at the intersection between theory, strategy, and evaluation. Besides proposing an MTD strategy and its evaluation, they present a robust theoretical framework with models and algorithms.

Papers that propose generic evaluation methods are helpful to support the comparison of MTD techniques. Alavizadeh et al. [2018b, 2017, 2018a] provide a modeling framework for the evaluation of MTD in cloud environments. The authors cover relevant security aspects such as return on the attack, attack cost, and the probability of attack success. Their results also comprise a comparison between a system with and without MTD deployments.

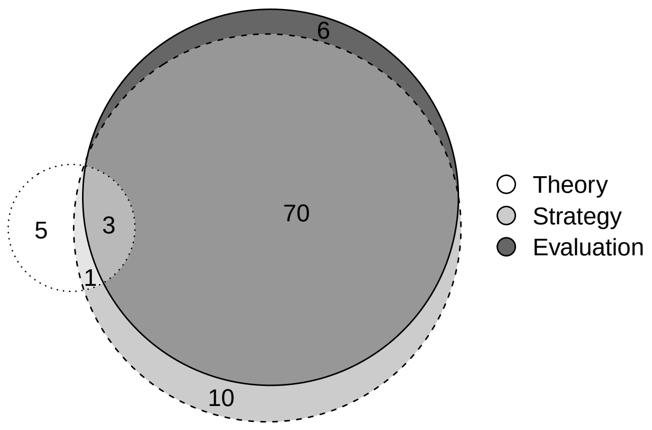


Figure 2.7: Research area classification

2.4.2.2 Evaluation Metrics - Papers Distribution

Figure 2.8 presents a Venn diagram with the distribution of the metrics found in the selected papers. We noticed a balanced distribution of papers in the proposed classification. Papers that include a performance evaluation usually focus on the overhead caused by the proposed (or evaluated) MTD strategy. For example, Yang and Cheng [2018] present an MTD based on Software Defined Network (SDN). Among their results, the authors compared the response time of the application when using their proposal with the traditional MTD strategies. Wang et al. [2016] propose a cost-effective MTD against Distributed Denial of Service (DDoS) and Covert Channel Attacks. Their performance evaluation covers the cost per minute of using different variations of MTD algorithms.

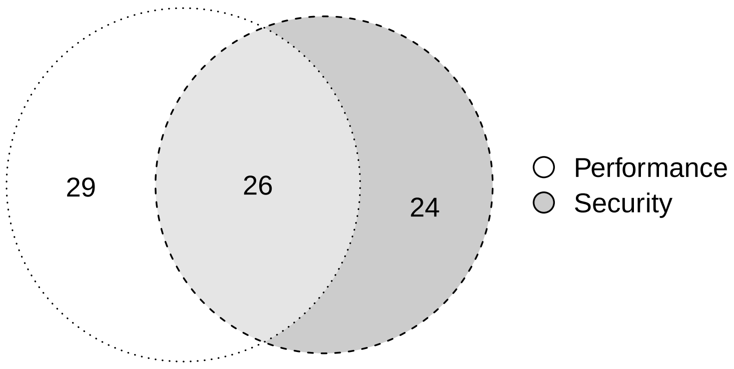


Figure 2.8: Evaluation approaches - Papers distribution

The specific assumptions of each research impose barriers to the comparison of different MTD strategies. The security evaluation metrics considered in the works analyzed tend to be related to specific aspects to characterize the proposed MTD effectiveness. The metrics are usually defined by the authors and applied in their particular context. For example, the study from Sianipar et al. [2018] is focused on the Meltdown and Spectre vulnerabilities. Therefore, their results are based on Spectre and Meltdown's effectiveness while applying their approach. Wang et al. [2014] propose an MTD solution as a DDoS defense. The evaluation comprises the percentage of clients saved based on the number of shuffles. Wahab et al. [2019] propose a comprehensive framework for MTD deployment in the cloud. Their framework comprises several techniques and methods, including a risk assessment methodology and a machine learning approach to collect information from malicious activities. In the evaluation, the authors use two primary metrics: percentage of attack detection and survived services.

The most recurrent security metric is related to the MTD impact in the attack success rate [Nguyen et al., 2018; Ma et al., 2016; Zhang et al., 2016a; Debroy et al., 2016]. However, it is still challenging to set up a direct comparison between the papers due to their particular assumptions. The development of a unified approach for MTD evaluation is an open problem.

2.4.2.3 Strategy - Papers Distribution

This section presents an overview of the type of MTD strategies applied in cloud computing. The results presented here provide answers to the proposed research question: "*What are the most researched techniques for moving target defense on cloud computing?*" Figure 2.9 presents a Venn diagram with the distribution of the proposed MTD strategies. Each set in the Venn diagram corresponds to MTD strategies related to the dynamic application, dynamic network, and dynamic platform.

We noticed that most of the MTD techniques leverage cloud computing's inherent features. For example, MTD based on the dynamic platform usually relies on VM migration for the environment reconfiguration. In this context, VM migration is typically used to defend against side-channel attacks [Moon et al., 2015; Zhang et al., 2012; Azab et al., 2017; Kashkoush et al., 2018; Adili et al., 2017; Yang et al., 2019]. Liu et al. [2018] present an MTD approach against side-channel attacks based on dynamically scheduling VM computing resources. Agarwal and Duong [2019] propose a different MTD solution to defend against side-channel attacks using a VM placement technique. They propose an algorithm to reduce the probability of malicious VM co-location. Also, using VM placement techniques, Ahmed and Bhargava [2016] propose a MTD framework based on the creation and deletion (*reincarnation*) of VMs. To improve security, the authors dynamically change the OS instance on the VM in each *reincarnation* round. Jia et al. [2014] present an MTD mechanism to isolate attacked servers from benign clients during DDoS attacks. Their approach consists of turning victim servers into moving targets. Penner and Guirguis [2017] leverage on both VM migration and VM placement techniques to provide a comprehensive MTD mechanism for cloud computing.

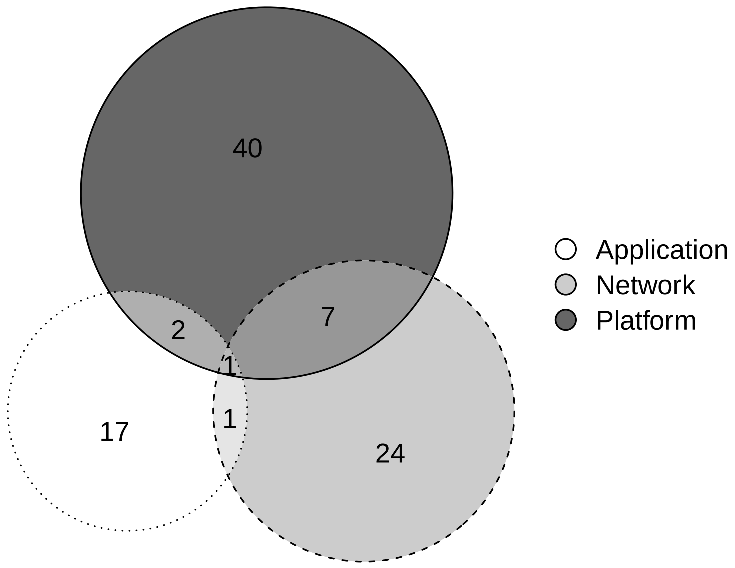


Figure 2.9: Strategy classification

Dynamic network approaches are usually based on network address hopping techniques [Groat et al., 2013; Luo et al., 2016; El Mir et al., 2017]. Kurra et al. [2013] present an MTD mechanism based on data partitioning and key hopping. Using key hopping mechanisms, the authors can reduce the length of keys to improve system performance while maintaining system security levels. Fleck et al. [2018] propose dynamic changes on the IP addresses of proxies to thwart the reconnaissance phase of attacks. Lysenko et al. [2018] also propose dynamic network configurations to protect a Corporate Area Network.

Regarding MTD techniques related to the dynamic application approach, we highlight that the most used technique is SBE. SBE is usually applied using a dynamic selection of functionally equivalent software variants at runtime [Dsouza et al., 2013; Le Goues et al., 2013; Hosseinzadeh et al., 2015]. The oldest paper in our classification [Azab and Eltoweissy, 2011] also applies SBE in the context of Cyber-Physical Systems (CPS). The authors used the *ChameleonSoft*, a biological-inspired MTD framework that provides software diversity at runtime.

Finally, we highlight that just one paper [Chung et al., 2015] proposes a framework (SeReNe) comprising all three layers (application, network, and platform). However, SeReNe is still in the conceptual phase and lacks practical implementation and evaluation.

2.4.2.4 Research Area and Strategy - Classification Relationship

The bubble chart in Figure 2.10 presents the relationship between the research area and strategy categories of the selected papers. Bubble charts simplify the identification of research gaps and the areas that received the most attention from the research community.

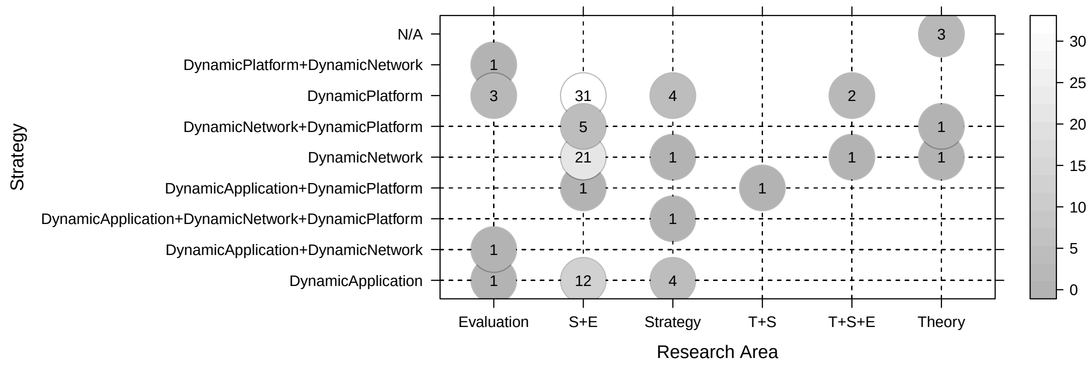


Figure 2.10: Relationship between Research Area and Strategy categories

As mentioned earlier, most papers propose an MTD strategy and its evaluation. Moreover, among these papers, the majority use dynamic platform strategies. There are three theoretical papers [Leslie et al., 2015][Lei et al., 2018][Bazm et al., 2017] that study generic MTD theory without focusing on specific strategies. Some papers propose MTD strategies but lack the evaluation of their effectiveness. MTD Theory receives less attention than the other areas from the research community. The paper from Casola et al. [2018] was classified as a theory and strategy because, besides presenting a security SLA-driven MTD framework, it presents a strong theory about cloud applications and security SLAs.

2.4.3 Key Findings

While applying or proposing MTD mechanisms for cloud computing, the researchers leverage cloud and virtualization features as VM placement or migration techniques. The problem with this approach is that these MTD techniques rely only on platform modification. Therefore, more well-prepared attackers may develop security attacks for higher layers, such as application confidentiality or user privacy. However, the use of cloud and virtualization capabilities reduces the cost for MTD implementation, as such techniques use cloud-embedded features.

There is a significant research effort in expanding the MTD in cloud computing to other platforms. The relationship between the cloud and these platforms is mutual. In some cases, the works use the cloud to enable an MTD in another specific platform (like CPS and IoT). In other cases, the papers use different platforms to improve security levels in an MTD in the cloud (like some papers using SDN).

Research opportunity 1 - ***Developing theoretical research about MTD in Cloud***. The majority of theoretical papers are generic. The development of MTD theories, which comprise the characteristics of the cloud and its virtualized environment, is a research opportunity. For example, a relevant MTD problem is the *MTD timing problem* where we try to define optimal schedules to perform MTD actions taking into account the desired system attributes (e.g., security, performance or sustainability).

Research opportunity 2 - ***Developing a unified framework for MTD evaluation***. The development of security benchmark tools is a challenging task because of the inherent unpredictability of the actions of the attacker. However, previous research [Vieira and Madeira, 2005][Dumitraş and Shou, 2011] provides directions for the design of such benchmarks. The current MTD in cloud research focuses on proposing new techniques to avoid (or reduce the likelihood of) specific security threats. The problem is that, without unified evaluation metrics, it is hard to compare and decide among the available MTD methods. Proposing a unified MTD evaluation approach may be an insurmountable challenge. However, starting to prove evaluation approaches for specific scenarios (e.g., MTD in the cloud, which applies VM migration to avoid attacks) seems to be an interesting research opportunity.

Research opportunity 3 - ***Developing multi-layer MTD***. As mentioned earlier, the MTD in cloud researchers explored mainly the cloud features as enabling mechanisms for MTD deployments. There is still a gap in the development of multi-layer MTD frameworks for cloud computing. Just applying dynamic platform and network techniques is not enough to mitigate sophisticated attacks that aim at the system's confidentiality or users' privacy. The development of a self-adaptive framework capable of dynamic multi-layer MTD selection is an interesting research challenge.

Research opportunity 4 - ***Study the impact of MTD in context-oriented clouds***. As presented in [Buyya et al., 2018], there is a need for holistic evaluations in cloud computing environments. Applying MTD in context-oriented clouds may impose severe system overhead. The development of strategies to evaluate the tradeoffs when using MTD in the cloud is a relevant future research line.

As a final remark, we highlight a simplified answer for the proposed research question (*what are the most investigated techniques in moving target defense on cloud computing research?*). In the dynamic platform papers, the most used technique is VM migration. In dynamic network papers, the most used technique is network address randomization. A significant number of dynamic network papers rely on SDN flexibility to perform dynamic network changes. Finally, the most used technique in dynamic application papers is SBE.

2.5 VM Migration-based MTD

We highlight that our thesis contributions fall into two research opportunities from our systematic map study: first, about MTD theoretical field. Our modeling framework provides information for *MTD timing problem* solving in specific cases. Second, about the development of a unified framework for MTD evaluation. Our thesis work proposes a model capable of evaluating the availability and security of a cloud computing environment. Through the proposed models and *PyMTDEvaluator* tool, it is possible to analyze and compare different MTD policies. Furthermore, we selected the VM migration-based MTD as it appears as a highly used technique in *dynamic platform* approach. Besides that, VM migration tends to be easier to apply, as it is a usual management task and is usually available in a variety of cloud computing platforms.

To analyze the VM migration-based MTD effectiveness, we need to select a suitable security threat. Note that VM migration-based MTD has a limited range of protection, as the VM and its applications states must remain the same after the process. This way, the VM migration is useful to defend against specific *host-based* threats [Zhang and Lee, 2017]. Basically, in the *host-based* attacks, the attacker controls a set of VMs and conducts attacks against its own PM host. In the scope of this thesis, we use the time-based VM migration with fixed interval as MTD. The approach is to swap the attacker position periodically to increase the security of the system.

Figure 2.11 presents the dynamics of a VM migration-based MTD against a *host-based* attack. In this example, the attacker runs a single VM in a two-PM virtualized setup. We have three stages. The switch between the stages is triggered by the VM migration. In the **Stage 1**, the attacker is in the *Host A*, and the *Host B* is free from attack. Then, in the **Stage 2**, after the triggering of VM migration, the attacker arrives in the *Host B*. During the attacker VM stay on the *Host B*, the attack effects on *Host A* cease. Finally, in the **Stage 3**, the attacker returns to *Host A*. Depending on the threat, the attacker may be able to continue the attack, or the attack needs to be restarted.

Inspired by the definitions of MITRE Corporation [2023], we named the two types

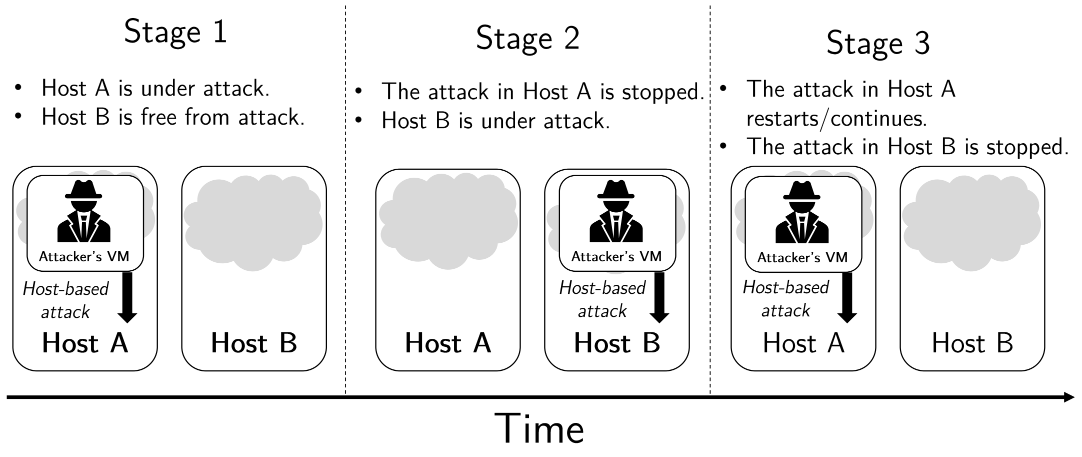

Stage 1
• Host A is under attack.
• Host B is free from attack.
Stage 2
• The attack in Host A is stopped.
• Host B is under attack.
Stage 3
• The attack in Host A restarts/continues.
• The attack in Host B is stopped.
Attacker's VM
Host-based attack
Host A
Host B
Time

Figure 2.11: VM migration based MTD dynamics - Two PM setup

of threats as *persistent* tactic and *non-persistent* tactic. In the *persistent* tactic, the attacker accumulates knowledge and is able to continue the attack as soon as the VMs arrives in a previously visited PM. On the other hand, in the *non-persistent tactic*, the attacker needs to restart the attack after each movement of the VMs.

Returning to the example depicted in Figure 2.11, let us consider the behavior of *Host A*. For the sake of simplicity, we consider that the attack progress is linear during the time in this illustration. Note that this example is an abstraction and that in real-world scenarios, the attack progress may vary according to the considered threats. Figure 2.12 presents the expected attack progress line in *Host A* in three different situations: a) systems without MTD, b) systems with MTD and under a *persistent* threat, and c) system with MTD and under a *non-persistent* threat.

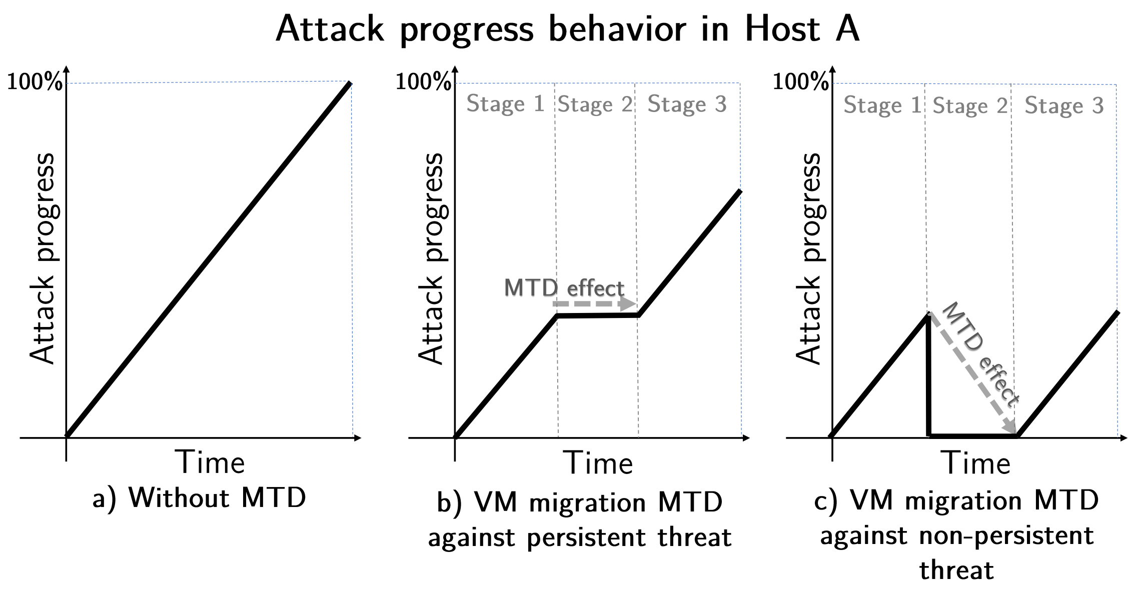


Figure 2.12: VM migration based MTD dynamics against *persistent* and *non-persistent* tactics

In the plot in Figure 2.12, the black line represents the attack progress. The vertical dashed lines correspond to the stage limits. The gray dashed arrow illustrates the expected MTD effect in each of the considered scenarios. The scenario with the absence of MTD (Figure 2.12 a)) shows that the attack progresses normally until it reaches success (i.e., attack progress of 100%).

In the scenario under a *persistent* threat, the behavior is as follows (Figure 2.12 b)). The attack progress is interrupted while the attacker is away from the *Host A*. Then, when the attacker returns to *Host A*, the attack continues its progress. Finally, in the scenario under a *non-persistent* threat, the behavior is the following (Figure 2.12 c)). The attack progress is nullified after the VM migration. The attack needs to be restarted when the attacker returns to *Host A*.

On the threats with potential defense through VM migration, we highlight the memory Denial of Service (memory DoS). Leveraging on issues in the hardware memory isolation [Li et al., 2020], memory Dos targets the memory of the VM launching the attack, trying to affect the co-resident VM availability by overloading shared memory.

The inspiring work Zhang et al. [2017] provided memory DoS background (including the attack source code). Li et al. [2020] provided insights on the memory DoS attack detection. Although both papers mentioned VM migration as a potential defense for memory DoS, the authors followed different approaches from ours. Zhang et al. [2017] dealt with the problem using *execution throttling.* [Li et al., 2020] focused on the memory DoS detection instead of the MTD proposal.

Chapter 3 presents our empirical observation results of using VM migration-based MTD against memory DoS. Appendix B provides additional empirical results showing the impact of memory DoS in different applications running inside the VM. In summary, existing studies confirmed the MTD effectiveness and highlighted that the MTD timing function (i.e., when to apply MTD) is a decisive factor in preventing attack success.

2.6 Multi Criteria Decision Making (MCDM)

In the development of this thesis, we used some MCDM methods to support the selection of time-based VM migration policies according to the user priorities. These methods are specifically in the Chapters 5 and 9. The use of MCDM in this context helps in answering questions such as: "*Which VM migration scheduling policy do I need to select when to prioritize availability over security?*" or "*How often do I need to migrate VMs to find a policy with 30% for availability importance and 70% for security importance?*".

There are numerous MCDM methods available [Ishizaka and Nemery, 2013]. From more rudimentary methods based only on the Euclidean distance from an optimal solution to more sophisticated methods considering the criteria to criteria comparison (e.g., Analytic Hierarchy Process (AHP) method [Asadabadi et al., 2019]). During our research, we used such methods at two levels of maturity. In Chapter 5, we used simpler methods based on the Euclidean distance, and in Chapter 9, we decided to use a more elaborated method based on the Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) method [Pavić and Novoselac, 2013].

2.6.1 MCDM based on the Euclidean Distance from an Ideal Solution

For the understanding of this MCDM method, let us consider an illustrative example below. After the computation of a hypothetical example, the evaluation output suggests three VM migration policies $P1, P2$, and $P3$. $P1$ policy achieves 0.001 of Unavailability (UA) and 70% of Probability of Attack Success (PAS). $P2$ has the following results $UA = 0.0004$ and $PAS = 80\%$. And $P3$, $UA = 0.0002$ and $PAS = 90\%$. The question now is how to decide which one of these policies is the best one when considering both metrics equally (i.e., both with the same weight for the decision-making).

The first step is to normalize the data. Here, we apply the method of dividing each element by the maximum obtained value (i.e., $x_{norm} = x/max(x)$). After the obtainment of the normalized values, it is necessary to propose an ideal solution. In this scenario, the ideal solution is the one that produces no unavailability and keeps the probability of attack success at 0% (i.e., origin point (0,0) of a xy plot with x=UA and y=PAS). Finally, we compute the Euclidean Distance[2] from each one of the normalized points to the ideal solution. In this scenario, we search for the shortest distance to the ideal solution. In this hypothetical example, the policy $P2$ is the best solution. Figure 2.13 summarizes the approach.

2.6.2 Technique for Order Preference by Similarity to Ideal Solution (TOPSIS)

Following the footsteps of [Araujo et al., 2018], we applied the TOPSIS [Chen and Hwang, 1992] method for selecting the policies in the software rejuvenation and MTD scenario. Different from the previous method, TOPSIS allows the inclusion of weights in the criteria used in the decision-making.

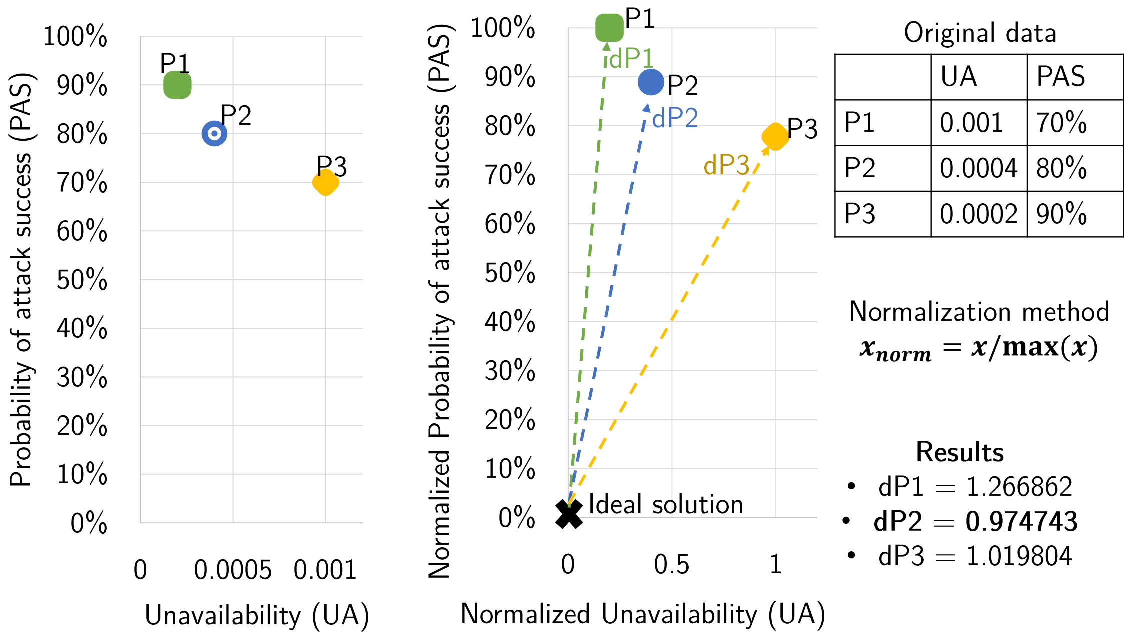
TOPSIS method has six steps as follows [Jahanshahloo et al., 2006]:

1. Compute the normalized decision matrix. The normalized value n_{ij} is obtained through the following.

$$n_{ij} = x_{ij}/\sqrt{\sum_{i=1}^{m} x_{ij}^2},\ i = 1, ..., m,\ j = 1, ..., n.$$

2. Obtain the weighted normalized decision matrix. Considering w_j as the weight of the ith criteria, and that $\sum_{j=1}^{n} w_j = 1$, the weighted value v_{ij} is

[2] $d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$, where p, q are two points in the Euclidean n-space, q_i, p_i are the Euclidean vectors and n is the n-space.



	UA	PAS
P1	0.001	70%
P2	0.0004	80%
P3	0.0002	90%

- dP1 = 1.266862
- **dP2 = 0.974743**
- dP3 = 1.019804

Figure 2.13: Hypothetical example using the MCDM method based on Euclidean distance from an ideal solution

obtained using

$$v_{ij} = w_j n_{ij},\ i = 1, ..., m,\ j = 1, ..., n,$$

3. Find the positive and negative ideal solutions. Consider I associated with benefit criteria and J associated with loss (cost) criteria.

$$A^+ = \{v_1^+, ..., v_n^+\} = \{(max_j v_{ij} | i \in I), (min_j v_{ij} | i \in J)\}$$

$$A^- = \{v_1^-, ..., v_n^-\} = \{(min_j v_{ij} | i \in I), (max_j v_{ij} | i \in J)\}$$

4. Compute the Euclidean distance from each normalized solution to the positive (d_i^+) and negative (d_i^-) ideal solutions.

5. Obtain the relative closeness to the ideal solution. The relative closeness from alternative A_i to A^+ is the following.

$$R_i = d_i^- / (d_i^+ + d_i^-)\ i = 1, ..., m.$$

6. Ranking using decreasing order the evaluated alternatives.

2.7 Related Works

The contributions of this thesis range from performability models of cloud systems with software rejuvenation to security models of MTD-enabled systems. In the current literature, we were unable to find related works with a similar range of contributions. Therefore, below, we categorize the related works in two sections, the first one for software rejuvenation-related works and the second one for the

MTD related works.

2.7.1 Software Rejuvenation

The works Machida et al. [2010, 2013] provide helpful insights into our availability modeling strategy. For example, we used their modeling strategy for our CLOCK and AVAILABILITY models. Specifically, we adopt deterministic transitions for the CLOCK model and a hypo-exponential distribution to represent the software aging accumulation in the AVAILABILITY model. Unlike their work, we also include the possibility of failure in the VM migration target host. Besides that, our model also covers the influence of software aging status in the VM migration pre-copy phase duration.

Thein and Park [2009] (one of the first works to propose evaluation through Markov Chains), Wang et al. [2007] (inclusion of performability metrics), also provide insights into the modeling process. However, none of these, or the others mentioned above, include the relationship between Moving Target Defense and Software Rejuvenation.

Bai et al. [2020] present a Semi Markov Process (SMP) for VM migration-based software rejuvenation. They analyze system availability and job completion time metrics. The main strength of their work is the possibility of using different probability distributions for the model's transitions. Different from their work, we consider several additional aspects, such as bursty workload occurrence and the security evaluation of a system using time-based VM migration. Furthermore, unlike their paper, we present empirical background to highlight VM migration effectiveness as software rejuvenation.

Liu et al. [2015] theoretically show a VM migration rejuvenation with a proactive error detection approach. Their proposed rejuvenation action comprises additional system entities to support the fault tolerance process, namely, the *aging failure detector*, which analyzes the physical resource status, and the *aging degree evaluator* used to store the context of aging failures. In the paper, the authors neglect to present experimental validation of their proposed technique. Unlike their work, we highlight previously obtained practical results of VM migration technique against VMM software aging. Besides that, we exercise different scenarios to find proper rejuvenation scheduling.

In an approach similar to [Bai et al., 2020], Okamura et al. [2014] theoretically analyzes software rejuvenation policies using a phase-type expansion approach. One of the highlights of their paper is the use of Markov Regenerative Stochastic Petri Nets (MRSPNs) to allow transient evaluation. In this field, the majority of the works focus on steady-state evaluation metrics. Different from their approach, our modeling ends up comprising more system details such as VM migration pre-copy details, bursty workload occurrence, and security evaluation.

Another interesting work of time-based software rejuvenation was proposed in Paing and Thein [2012]. In their work, they also considered the VM migration technique. Through SRN models, they compute rejuvenation policies that maximize the desired metrics. Interestingly, they carefully present a set of closed forms for metrics computation. As in some of the previously mentioned related works, our thesis comprises additional aspects of the system as well as the security evaluation.

Following a different approach from the majority of the works in this field, Fakhrolmobasheri et al. [2018] took a step further in analyzing not only availability but also performance and power consumption. They compute the overall power consumption and probability of VMM failure under different circumstances. Besides the differences in the research scope of our work and theirs, we highlight that, unlike their work, our models take the VM migration overhead impact into consideration.

It is important to highlight the works from Levitin et al. [2018] and Dohi et al. [2001], which provide significant contributions to the software rejuvenation scheduling field. Their theoretical contributions improve the overall understanding of how to optimize the policies and how to estimate the scheduling for rejuvenation.

As a final remark on software rejuvenation modeling papers, we highlight the additional works Bai et al. [2023a,b], which exercise the rejuvenation-enabled system with specific applications (namely, multi-access edge computing and vehicle platooning). From a perspective, our approach of merging MTD and software rejuvenation. However, unlike their work, we present the application-related metrics (e.g., probability of attack success, tolerance levels)

2.7.2 Moving Target Defense

Alavizadeh et al. in the papers [Alavizadeh et al., 2018a, 2019b, 2020, 2021] provide a comprehensive security assessment of MTD on cloud computing. The evaluation is based on modeling and analysis of MTD techniques. The authors evaluate four security metrics: system risk, attack cost, return on attack, and availability. The main contribution of their work is an approach to evaluate the effectiveness of combined MTD. They provided models for evaluating the combination of Shuffle, Diversity, and Redundancy MTD techniques in cloud computing environments. The assessment uses Hierarchical Attack Representation Model (HARM) models Hong and Kim [2015] for combined MTD techniques. HARM effectively evaluate MTD strategies as they combine Attack trees in the bottom layer and Attack graphs in the upper layer. However, they are often specific for certain scenarios as they require network topology description. Unlike their approach, we use SPN in the security evaluation, as these provide a different perspective on security evaluation, focusing on how stochastic events affect the system status. Additionally, through the SPN models, we can cover aspects such as the influence of software aging in the MTD system. Besides that, we extend our contributions proposing *PyMTDEvaluator*, to enhance the model-based evaluation usability by providing a user interface.

We highlight another work Alavizadeh et al. [2019a], which proposes an automation framework for MTD deployment on cloud computing. Like *PyMTDEvaluator*, their work also provides a user interface to interact with the engine of the security evaluation framework. Their framework is capable of communicating and deploying MTD on OpenStack-based clouds. While their work goal is to deliver a well-rounded solution for automated deployment of MTD on the cloud, *PyMTDEvaluator* is more focused on providing means for evaluating and comparing time-based MTD alternatives. The inspiring work from Alavizadeh et al. is nicely compiled in his Ph.D. Thesis [Alavizadeh, 2020].

Thebeau II et al. [2014] provides a theoretical point-of-view of how to measure the resiliency of a cloud that applies SBE MTD. SBE uses software diversity to improve system security, survivability, and resilience. The paper describes some of the essential concepts of security evaluation, such as integrity, availability, survivability, and confidentiality. Finally, the paper proposes a model for resiliency quantification in scenarios with SBE-based MTD. Unlike their work, we deliver models covering different MTD deployments on Cloud Computing. Such deployments are based on VM migration.

Ahmed and Bhargava [2016] propose *Mayflies* MTD framework for distributed systems. *Mayflies* use a specific policy of VM placement as MTD. The idea is to perform VM substitution through creation and deletion cycles, obeying certain time intervals. Every cycle of substitution changes a VM characteristic. In *Mayflies*, VMs are created to use a different operating system from the previously deleted VM. The strategy avoids attack progress or the spread of an undetected attack. The authors evaluate their proposed framework through experiments in a real testbed. However, different from our approach, the paper does not present a security analysis of the proposed technique.

Chung et al. [2015] propose SeReNe, a platform to deliver Network-Security-as-a-Service (NSaaS) for multi-tenant data center environments. SeReNe plans to apply MTD using diversity techniques to mitigate software vulnerabilities such as Bohrbugs, Mandelbugs, and aging-related bugs. However, SeReNe is still in a conceptual phase, and the paper lacks its practical implementation and evaluation.

Mendonça et al. [2020] presented a model for performability evaluation of a system with a time-based Moving Target Defense. The proposed MTD leverages Software-Defined Networking (SDN) to perform the environment modifications. The authors neglect the security improvement assessment due to MTD deployment. Unlike their work, we focus on the security evaluation regarding the probability of attack success.

Chen et al. [2020] presented a SRN model for job finish time evaluation of a system with MTD based on VM Migration. The paper offered relevant insights into our model design and parameterization. The presented results focus on the job finish time under different conditions. Unlike their work, we focused on evaluating the MTD *effectiveness* under different architectures and VM migration scheduling policies. By *effectiveness*, we mean a reduction in the probability of attack success.

The papers [Cai et al., 2016a; Moody et al., 2014] were among the initial works about MTD evaluation using PN. Their research was one of the first efforts to use

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

Connell papers [Connell et al., 2018][Connell et al., 2017] presented comprehensive models for availability, security, and performance evaluation of an environment with MTD. In these works, the authors also covered the probability of attack success and availability in the evaluations. They presented optimal reconfiguration rate results. Different from their work, we cover scenarios with different levels of probability of VM migration failure. Moreover, our modeling approach is different as we decide to represent the attack progress Increasing Failure Rate (IFR) using an Erlang sub-net. Instead of investigating specific scenarios, *PyMTDEvaluator* aims to provide mechanisms for the users to analyze and compare their own MTD deployment alternatives.

Chang et al. [2020] provided an SRN model for job completion time evaluation in an MTD system that considers a virtualized platform with Software Defined Networking capabilities. They also computed the availability and probability of attack success. The main goal of their paper is to investigate the MTD impact on job protection and performance. The significant difference between our approach and theirs is the threat model. In their work, they consider the attacker to select the attack targets without accumulating knowledge. In their threat model, the attacker is trying to compromise a specific job. Besides, different from their work, we also present the *tolerance levels* results, aiming to support managers' decision-making.

It is important to mention the work [Sianipar et al., 2018], which investigated the MTD technique to protect the system against live memory dumping, specter, and meltdown attacks. They proposed an MTD based on moving data in physical memory. Their paper gave us valuable insights into the threat model considered in Chapter 7.

Anderson et al. [2016] proposed a Stochastic Petri Net to evaluate MTD effectiveness. The authors considered a metric named *probability of Information Operation (IO) success.* This metric has a similar meaning to our probability of attack success. Like our work, theirs also presents an analysis of the impact due to the variation of the parameters of the models. Differently from their work, we also covered the system availability in our work. Besides that, we also deliver a set of different metrics as *tolerance levels* and *effectiveness limit.*

Distefano et al. [2020] present a Petri Net model for Moving Target Defense evaluation. Their work also includes security and availability metrics in the evaluation and considers a multi-stage attack. Different from their work, we propose specific metrics to evaluate MTD as $ETTL$, $PAS(t)$ and IR.

Enoch et al. [2022] present an approach for MTD evaluation using models and multi-objective algorithms. Their approach also covers availability and security metrics. They also use Petri Net models to evaluate system availability. Their defensive mechanism consists of node isolation and application disabling. Unlike them, we use VM migration to support diversity MTD.

Alhozaimy and Menascé [2022] provide a formal analysis of task reconfiguration (i.e., MTD) using regular and irregular intervals. Their evaluation presents the tradeoffs between performance and security metrics, and the work focuses on studying how regular and irregular intervals of MTD affect the capability of the attacker to successfully finish the *reconnaissance* phase of the attack. Unlike their work, the MTD technique we consider intends to counteract the *attack* phase instead of the *reconnaissance* phase. Besides, their work neglects the influence of software aging and rejuvenation on MTD deployment.

2.8 Summary

Table 2.1 presents a concise comparison between this thesis and the most relevant related works. The table and all the discussion in this chapter reveal a research gap on comprehensive MTD evaluation mechanisms comprising aspects of availability and security in a VM migration-enabled environment. Specifically, the findings of the systematic mapping study motivate the research in the MTD evaluation topic. The proposal of a MTD **unified evaluation framework** is still a relevant research challenge. Our starting point to tackle this problem is to use a well-known modeling framework (i.e., SPN) for the evaluation of one of the main cloud MTD techniques (i.e., VM migration). From this point on, the question is how to fill the existing gap.

The first effort, presented in the following chapter, is to clarify what are the intricacies of VM migration as MTD. One effective method to reach this goal is to analyze the system behavior under such circumstances. The knowledge obtained from this observation may support the model design. For that reason, the following chapter brings the system behavior details while using VM migration as software rejuvenation and MTD. The chapter also includes empirical results, which highlight the effectiveness of the proposed technique for both scenarios. It is one of the core chapters of this thesis, as it presents the system architecture adopted in the next chapters.

Table 2.1: Most relevant related works comparison - a summary

Work	**Availability evaluation**	**Software aging and rejuvenation**	**Security evaluation**	**Moving Target Defense**	**Tool proposal**	**Empirical Results**
[Alavizadeh et al., 2018a][Alavizadeh et al., 2019b][Alavizadeh et al., 2020][Alavizadeh et al., 2021] [Enoch et al., 2022] [Alhozaimy and Menascé, 2022]	Yes	No	Yes	Yes	No	No
[Alavizadeh et al., 2019a]	No	No	Yes	Yes	Yes	Yes
[Machida et al., 2010][Machida et al., 2013]	Yes, some of them also include performance metrics	Yes	No	No	No	No
This work	Yes, including also performability metrics	Yes	Yes	Yes	Yes	Yes

Chapter 3

System Architecture

The size of virtualized environments may vary from a single PM to a large datacenter with hundreds of servers. Therefore, it is important to clarify what is the specific virtualized environment deployment that we consider in our work. In practice, the baseline system architecture (see Figure 3.1) consists of a virtualized environment with two physical machines: *Main Node* and *Standby Node*. *Main Node* runs the VMs. It is possible to perform VM migration from *Main Node* to *Standby Node* and back. HW+OS refers to the hardware components and the Operating System, which are treated as a single component in the availability evaluations. VMM (hypervisor) refers to the VMM component.

The VM migration triggering observes a schedule with regular intervals between migrations (i.e., time-based VM migration policy). The adopted VM migration follows the live migration *pre-copy* approach [Clark et al., 2005] that has specific stages (namely, *stop-and-copy* and *commitment*) in which the VM goes out of service. Therefore, each VM migration has an associated downtime. In the model-based evaluation (next chapters), we take into account the impact of the VM migration-related downtime on the availability.

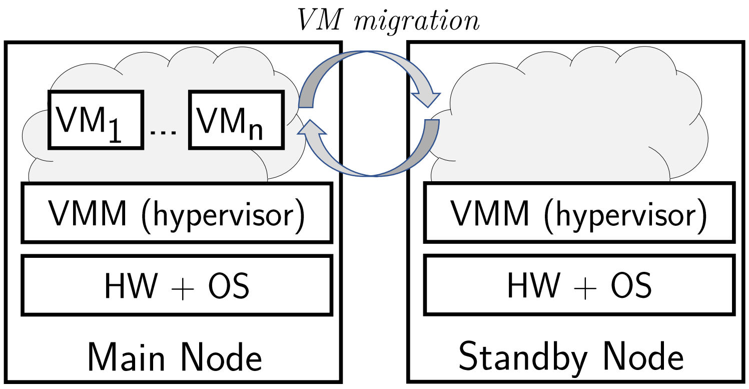


Figure 3.1: System architecture

The system becomes non-operational after a VM failure, and it returns to operation after a VM repair. As the VM depends on the *Main Node* to perform its operations, failures on the *Main Node* affect the VM availability. After a *Main Node* failure, the system turns operational again by performing a two-step recovery: repair the *Main Node* and restart the VM. Failures on the *Standby Node* do not represent a system failure. However, *Standby Node* failures prevent the

VM migration. The VMM component is prone to suffer software aging. The proposed evaluations also consider failures that are not related to software aging (e.g., hardware or OS failures).

We selected this architecture for our experimentation and modeling for the following reasons:

1. This is the **baseline architecture for virtualized environments**. The proposed architecture is the fundamental building block for a VM migration-enabled environment. From small to large datacenters, we need at least two available PMs to enable VM migration. Nevertheless, nowadays, a cloud deployment may be as simple as a single PM, as is the case of the MicroStack deployment that can be used in different scenarios [Sarwar et al., 2022; Kielland et al., 2022; Jyotinagar and Meshram, 2023].

2. **Control the model complexity**. State space-based models may suffer from *largeness* problems. Even with the use of PN-based models (which are usually adopted as a *largeness* avoidance technique), it is necessary to carefully select what components to include in the availability and security evaluation.

3. **Easier instantiating in a real testbed**. The selection of a reduced architecture simplifies its instantiating in a real testbed. This is particularly appropriate for our experiments as we need a dedicated setup to run the workloads.

4. **Increased research reproducibility**. The adoption of a simplified architecture facilitates the process of experimental results verification.

This chapter also presents the results from empirical observations of VM migration as support for software rejuvenation and MTD. We applied VM migration to counteract *hypervisor* software aging. The main goal is to verify whether the migration can remove the accumulated AR effects. The observation results show, in a practical manner, the validity of using the VM migration as support for software rejuvenation. In the MTD observation, we put the VM migration-based MTD technique to test against a *host-based* attack. The obtained results also proved the validity of the technique for cybersecurity defense purposes.

These empirical observations are necessary to properly understand the behavior of the system, providing insightful information to guide the model design and to justify the modeling of VM migration and MTD in the studied scenarios. Theoretically, migration is able to support *hypervisor* software rejuvenation and to protect the system against *host-based attacks*. However, our goal here is to verify these proprieties in a real-world scenario. Furthermore, the empirical observation also sheds light on details that might be missing from prior knowledge. For example, the software rejuvenation experiment results highlighted the AR effects in the applications running inside the VM. Besides that, an interesting insight from the MTD experiment is that the target application does not recover even after the migration of the attacker VM to another host.

The rest of this chapter is organized as follows. Section 3.1 presents the details of time-based VM migration as support for software rejuvenation. Section 3.2 explains the general strategy of using time-based VM migration as MTD. Section 3.3 presents two case studies on the use of VM migration for both purposes (i.e., software rejuvenation and MTD). Section 3.4 presents the limitations and threats to validity. Finally, Section 3.5 concludes this chapter.

3.1 Virtual Machine Migration for Software Rejuvenation

Also known as *Migrate-VM rejuvenation* [Machida et al., 2010], the rejuvenation approach consists of moving the VMs away from a host with software aging accumulation to a host without aging accumulation (the technique targets the aging accumulation of the PM hosting the VMs). Specifically, we aim to counteract software aging accumulation in the VMM component, as previous works [Machida et al., 2012; Matos et al., 2012a; Torquato et al., 2018a] found empirical evidence of aging bugs in major VMM software as Xen[Xen, nd] and KVM[KVM, nd].

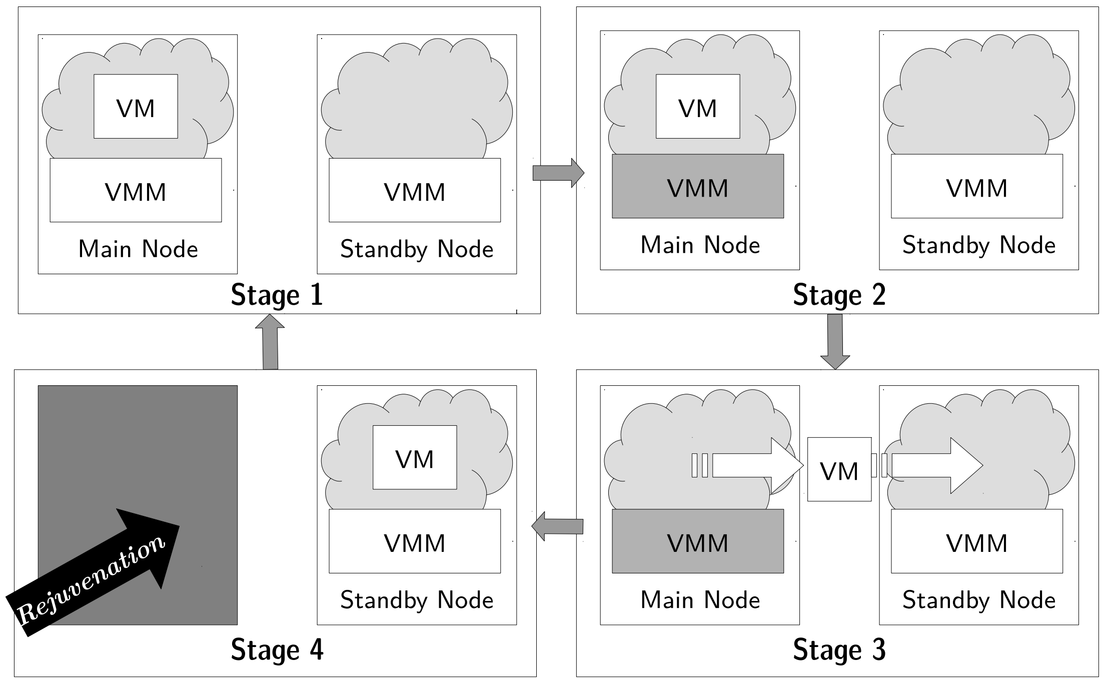


Figure 3.2: VM migration as support for software rejuvenation - workflow

Figure 3.2 summarizes the adopted *Migrate-VM rejuvenation* approach. It works in a cycle and has four stages:

- **Stage 1** - In the first stage, the VMs are running in *Main Node* and *Standby Node* is idle, waiting to receive migration.
- **Stage 2** - as time passes, *Main Node* VMM component starts to accumulate software aging effects. As the VMs depends on the VMM to run, the aging effects also affect the VMs execution. Then, the system enters a degraded state of execution, which may reach failure in the case of the lack of timely software rejuvenation.

- **Stage 3** - At this stage, the system triggers the scheduled VM migration. The main goal is to restore the VM execution state. To assure the intended restoration, migration moves the VMs from the *under-aging* VMM to the *fresh-state* VMM. Bovenzi et al. [2011] highlights the influence of the workload in software aging, showing that heavy workloads accelerate aging accumulation compared to light workloads. Therefore, although the VMM of *Standby Node* has been running, its aging accumulation is negligible.
- **Stage 4** - After VM migration completion, *Standby Node* assumes the role of *Main Node* (i.e., in charge of VM running). In this stage, *Main Node* passes through software rejuvenation (i.e., software restart or OS reboot). After rejuvenation completion, *Main Node* turns into *Standby Node* (i.e., waits as the target for the next VMs migration). Then, the cycle restarts to **Stage 1**.

3.2 Virtual Machine Migration as MTD

Following the classical definitions of Okhravi et al. [2013], VM migration-based MTD is a **dynamic platform** technique. In the scope of our work, MTD serves as a defense against specific *host-based* attacks (e.g., VM to host (or hypervisor) attacks as specific *resource starvation* attacks, and *VM escape* attacks [Patil and Modi, 2019]). The goal is to continuously shift the attack target by moving the VMs across the available physical hosts. Note that, for specific VM to VM threats, the VM migration movement may suffice to defend the system. For example, in the case of *RowHammer* threat, a VM tries to compromise the co-resident VMs through the shared memory. In such cases, the complete remapping of VM placement may prevent the success of an attack.

In Chapters 6 and 7, we analyze the time-based VM migration as MTD against threats using *persistent* and *non-persistent* tactics, respectively. As stated before, the general principle is that, in both scenarios, the goal of the attacker is to compromise the physical host and consequently the set of co-residents VMs. In the *persistent* tactic, the attacker can continue the attack as soon as the VM arrives in a previously visited physical host. Alternatively, in the *non-persistent* tactic, the attack must restart the attack after each VM migration. We provide more details about the attack and defense models in each chapter, as needed.

Migration can be used as MTD against *persistent* and *non-persistent* threats. The context discussed previously is focused on VM migration as a defense against *host-based* attacks. This type of threat is further investigated in Section 3.3.2, through an empirical observation of the validity of the technique. Nevertheless, it is also interesting to highlight that the VM migration-based MTD may act as a competent defense against *VM escape* attacks. In the MITRE ATT&CK matrix [MITRE, 2023], we find the *defense evasion* tactic. In this tactic, the attacker tries to avoid the usual detection mechanisms. A usual way to perform such evasion is through the abuse of trusted processes (e.g., their own authorized VMs). In the *VM escape* attack, the malware bypasses the *hypervisor* layer to access the host and consequently the co-resident VMs. To reach this specific goal, the attacker may capitalize upon specific hypervisor vulnerabilities. This type of threat is highlighted as highly relevant in the PICUS Red Report [PICUS, 2023]. In that report, the authors analyzed half a million malware to find the specific tactics applied. They found the *virtualization/sandbox* evasion, which is similar to the *VM escape*, in more than 10% (i.e., about 50000) malware. As VM migration is a dynamic platform technique, it might be useful to mitigate those threats moving VMs across multiple *hypervisors*.

Figure 3.3 shows the dynamics of using VM migration-based MTD against *VM escape* attacks. In this hypothetical scenario, the attacker targets a victim VM in the environment. The attack necessarily passes through the underlying hypervisor to reach the target. In an environment with heterogeneous hypervisor VM migration [Kargatzis et al., 2017], it is possible to move the VMs across PM hosts with different hypervisors. The technique leverages the multi-*hypervisor* environment to continuously shift the attacker position. In the **Stage 1**, the VMs are in a host with KVM *hypervisor*. In **Stage 2**, we move the VMs to the Xen *hypervisor*. This migration may stop the progress of the attack under the KVM *hypervisor*, as these attacks are dependent on the presence of VM in the targeted host. For example, they may use MAC addresses or CPU identification to find out which *hypervisor* hosts the VM. Finally, whenever possible, the next migration should arrive in a different *hypervisor* (for example, in the VMWare ESXi in **Stage 3**). When all the *hypervisor* variants are visited, the cycle may restart (i.e., moving from **Stage 3** back to **Stage 1**).

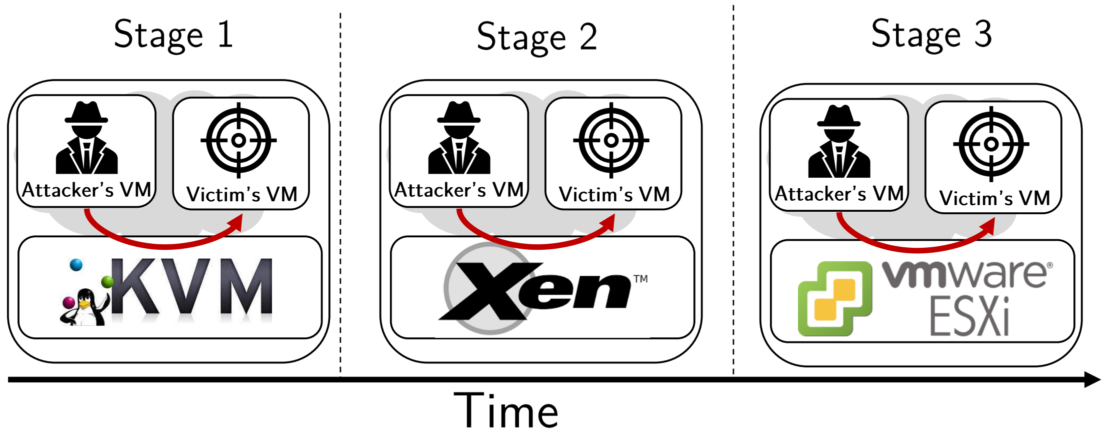


Figure 3.3: Dynamics of using VM migration as MTD against *VM escape* attacks

3.3 Case Studies

The results presented below come from empirical observations of the VM migration technique applied in both considered scenarios: software rejuvenation and MTD. These observations are mainly to verify the validity of the proposed approach in a real-world setup. Therefore, they should not be considered as validation for the proposed models, as such validation experiments would require a more extensive statistical analysis of the obtained results. The conclusions from these observations serve as inputs for the modeling process in the next chapters.

3.3.1 CS #1 - Software Rejuvenation

In previous works [Torquato et al., 2017, 2018a], we proposed the *Stress Wait Rejuvenate (SWARE)* approach for software aging and rejuvenation experiments. Since then, the approach has been applied in other contexts, such as for virtualized containers software aging research [Torquato and Vieira, 2019; Vinícius et al., 2022] (for convenience, we reproduced the paper [Torquato and Vieira, 2019] in Appendix C). In the *SWARE* paper, we used a VM migration as a rejuvenation experiment to validate the proposed approach. The results below are a summary from that original paper [Torquato et al., 2018a].

In the experiment, we adopt an infrastructure similar to the one presented before (Figure 3.1). Based on the environmental limitations and previous knowledge, we select a specific workload to accelerate possible aging-related bug activation. The main idea is not to submit a realistic workload to the system. Instead, the goal is to overload the system, forcing it to an increasing allocation of resources. Although we need a heavy workload, we must also be careful to avoid premature failures. This way, we consider a workload based on mounting and unmounting 15 disks in a VM running a web server (henceforth, *aging workload*). Besides the *aging workload*, the web server responds to a constant workload of 2000 requests per second[1]. Figure 3.4 summarizes the workflow.

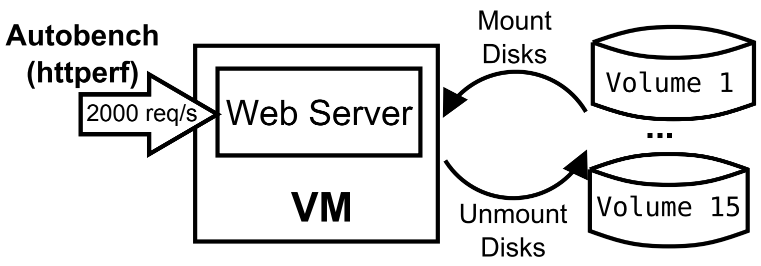


Figure 3.4: VM migration as support for software rejuvenation experiment - Selected workload - extracted from [Torquato et al., 2018a]

SWARE has tree phases:

- **Stress** - This phase aims to stress the system with *aging workload.* The stress workload leads to system internal state degradation.
- **Wait** - Starts after the Stress phase. The idea is to observe the system behavior after *aging workload* exposure. There are two possibilities: *i)* the system recovers without intervention - which provides no evidence of software aging; and *ii)* the system stays in a degraded state - providing evidence of software aging accumulation.
- **Rejuvenation** - Starts with the submission of software rejuvenation action. The objective is to observe the impact of such rejuvenation action on the internal system state. Ultimately, this phase reveals whether the proposed software rejuvenation was effective or not.

[1]Obtained through a capacity test. More details in the original paper [Torquato et al., 2018a]

Figure 3.5 presents the main results. As expected, the Web Server suffers effects from the high workload exposure in the Stress Phase. Response Time and Amount of Errors increased during that phase. In the Wait Phase, results show that Errors and Response Time remain at high levels. The rejuvenation phase brings the system to a stable state.

In a normal setup, without *aging workload* exposure, the web server can respond to 2000 requests per second with a negligible amount of errors and a minimum response time. However, in our experiments, we noticed a degraded behavior accumulated after the *aging workload* exposure. Finally, VM migration triggering brings the system back to normal levels. The peak of errors and response time right after VM migration is due to the downtime phase of the pre-copy migration algorithm. This clearly shows the effectiveness of using VM migration against KVM software aging, highlighting the prompt recovery of the application-related metrics after the migration. In the original paper [Torquato et al., 2018a], we also present the results of CPU and memory usage of the PMs), showing VM migration also resulting in a complete removal of aging-related effects.

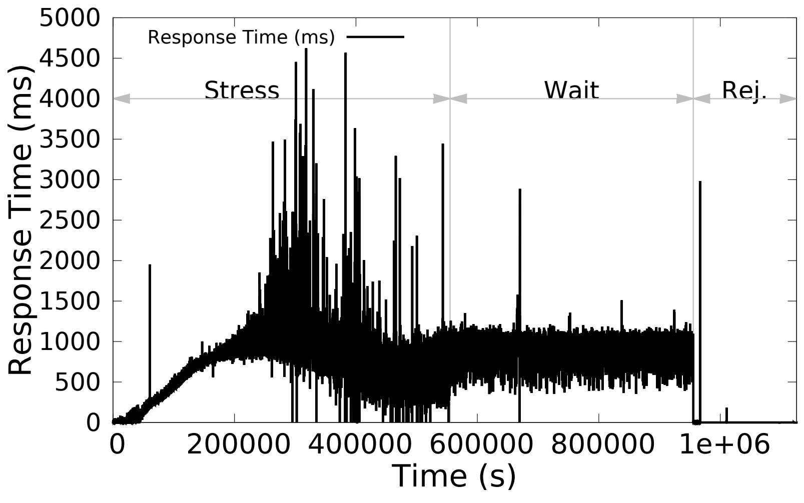
3.3.2 CS #2 - Moving Target Defense

As mentioned before, the use of VM migration as MTD lies in the *dynamic platform* technique [Cai et al., 2016c]. Specifically, we consider *host-based* attacks, which aim to affect the underlying physical host and, consequently, the co-resident VMs. MTD seeks to interrupt the attack progress by moving the VMs across the available PMs.

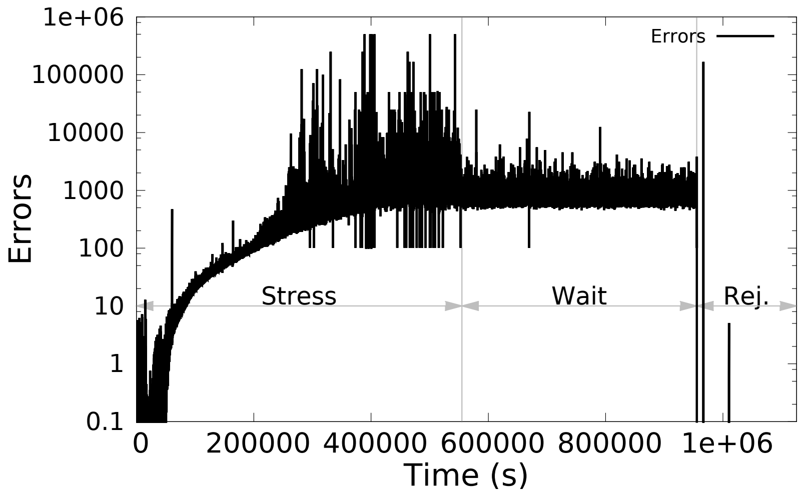
We conducted an experimental evaluation for a better understanding of VM migration as a MTD technique. In other words, we aim to verify the effectiveness of the time-based VM migration policy as MTD against *host-based* attacks. To reach this goal, we conducted an *host-based attack* campaign in a real testbed. This *host-based attack* campaign includes two experiments:

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

The details and results are described below. Although the results refer to a single run of each experiment, we performed additional runs to confirm the observations. At the time of this thesis writing, we were able to perform ten runs for the *attack severity experiment* and five runs of attacks in an MTD-enabled environment. The results presented below come from Torquato et al. [2021b]. We present additional evidence of VM migration as MTD in another publication [Torquato and Vieira, 2021], which is, for convenience, reproduced in Appendix B.



(a) Response Time (ms)



(b) Errors

Figure 3.5: Software aging and rejuvenation experiment results - VM migration starts the Rejuvenation phase - extracted from [Torquato et al., 2018a]

Figure 3.6 presents the experimental testbed, which includes three physical machines: Client (Intel i5 8250U + 16GB of RAM) - responsible for stressing the Victim VM; *Main Node* (Intel Xeon E5-2620 2.00GHz + 16 GB of RAM with Error Correction Code enabled) - main host of the Victim VM and Attacker VM; *Standby Node* (Intel Core i7-9700 3.00GHz + 16GB of RAM) - host for receiving VM migration. Attacker VM and Victim VM are KVM VMs with a homogeneous configuration: single-core processor + 3 GB of RAM. The Main Node and Standby Node run Ubuntu Server 20.04.2 with kernel 5.4.0-72 and KVM 4.2.1. The Victim VM runs the TeaStore microservice architecture [von Kistowski et al., 2018], which emulates a basic web store.

To investigate the system behavior under a host-based attack, we exercise our testbed with a DoS attack from an insider VM. We aim to observe the attack impact in a co-resident VM running a standard client-server application. This

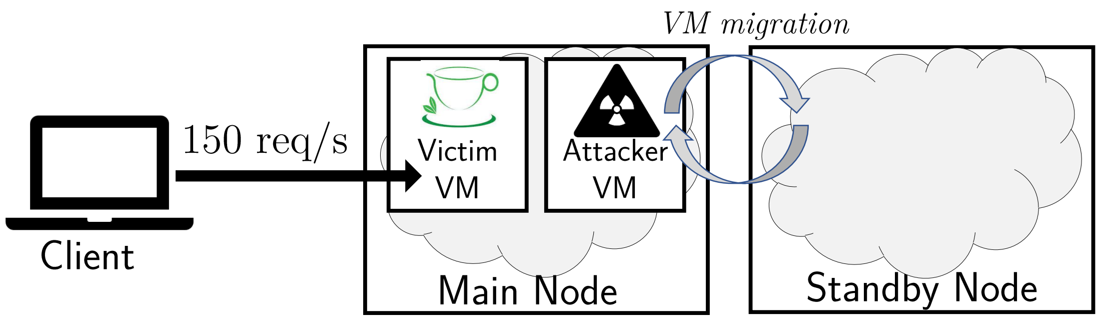


Figure 3.6: Experiment testbed

way, the first step in the *attack severity experiment* is to define a proper workload for the Victim VM. Using the httperf tool [Mosberger and Jin, 1998], we conducted a capacity assessment of that VM. Figure 3.7 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 decided to select a lighter workload to provide room for expected performance oscillations due to the host-based attack. Thus, we arbitrarily configured the Client to send a steady workload of 150 req/s to Victim VM.

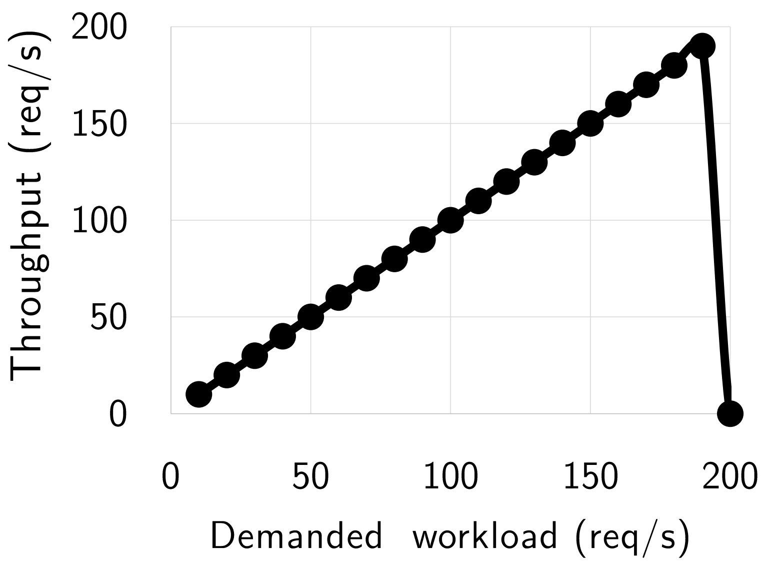


Figure 3.7: Victim VM TeaStore capacity assessment results

With knowledge about the Victim VM capacity, we can set up the attack load of the *attack severity experiment*. The specific host-based attack is a memory DoS [Zhang et al., 2017] attack from the Attacker VM running inside *Main Node*. In practice, the Attacker VM runs an infinite loop of unalign atomic accesses [Zhang et al., 2017] (*unalignAttk*) against its own memory. We then analyze the possible impact of such an attack in the Victim VM TeaStore service.

We performed the *attack severity experiment* for 30 minutes. The *golden run* results are included for comparison. Figure 3.8 presents the TeaStore throughput results, and Figure 3.9 presents the number of observed errors. The errors are related to connection timeout (i.e., the server is offline).

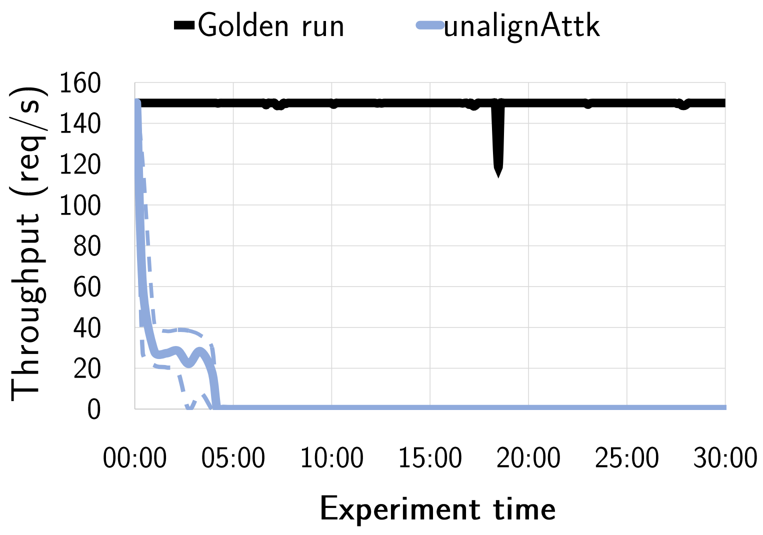


Figure 3.8: TeaStore - Throughput (req/s) - attack severity experiment

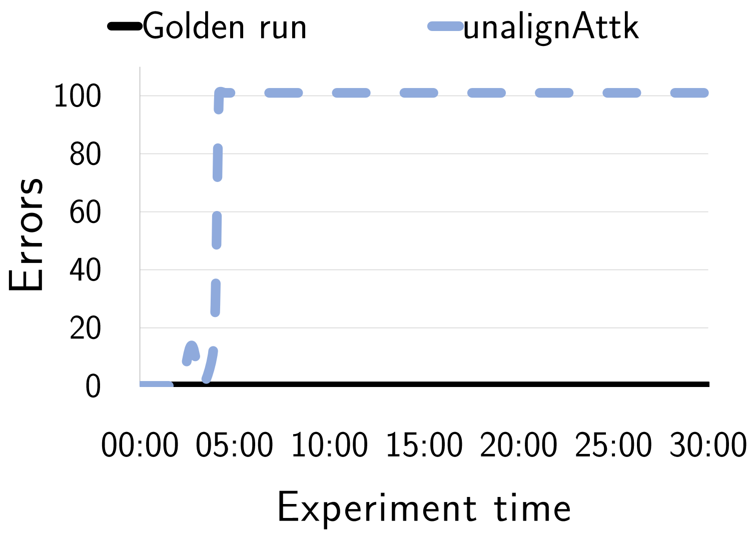


Figure 3.9: TeaStore - Errors - attack severity experiment

We can see an immediate drop in the throughput when the Attacker VM is running the *unalignAttk*. After five minutes, we noticed a substantial increase in the number of observed errors, suggesting server unavailability. Therefore, in our specific scenario, the *unalignAttk* successfully compromises the TeaStore availability after five minutes.

The goal of the second experiment, the *MTD experiment*, is to verify the VM migration effectiveness in the presence of the *unalignAttk* attack. We applied different VM migration schedules to observe their possible impact on the desired metrics. In practice, we consider four scenarios: 30 minutes between migrations, 45 minutes between migrations, 60 minutes between migrations, and the system without MTD. We observe the Teastore server throughput while receiving a constant workload of 150 req/s. Note that, in some preliminary runs, we noticed that early attack triggering (during the Teastore warm-up phase) might cause premature failures. Therefore, we wait for the system to reach the expected throughput rate (i.e., 150 req/s) before the attack initiation. The warm-up varied from 8 to 15 minutes, depending on the scenario.

Results are presented in the figures below. Each MTD-enabled experiment plot presents vertical lines highlighting the VM migration trigger event. As mentioned before, 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, and in all scenarios, we noticed an immediate and severe drop in the throughput upon the attack triggering. The green shaded area corresponds to the warm-up phase. The gray shaded area represents the period that the system is under attack. The area without shade is related to the period when the attacker VM is running on the other PM (i.e., *Standby Node*).

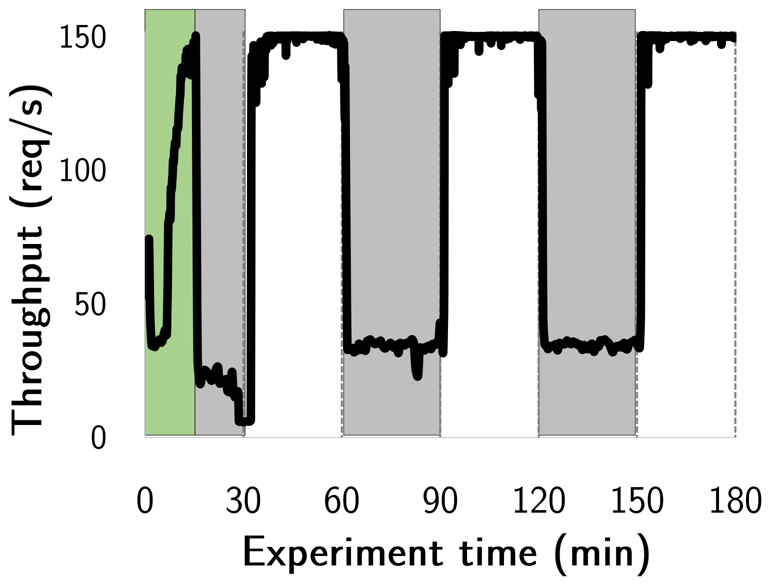


Figure 3.10: VM migration schedule = 30 minutes

Figure 3.10 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 (Figures 3.11 and 3.12) 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 (Figure 3.13). The specific reason for this persistent failure is outside the scope of this discussion. However, we presume the system drops the connection when perceiving consistent degraded service quality.

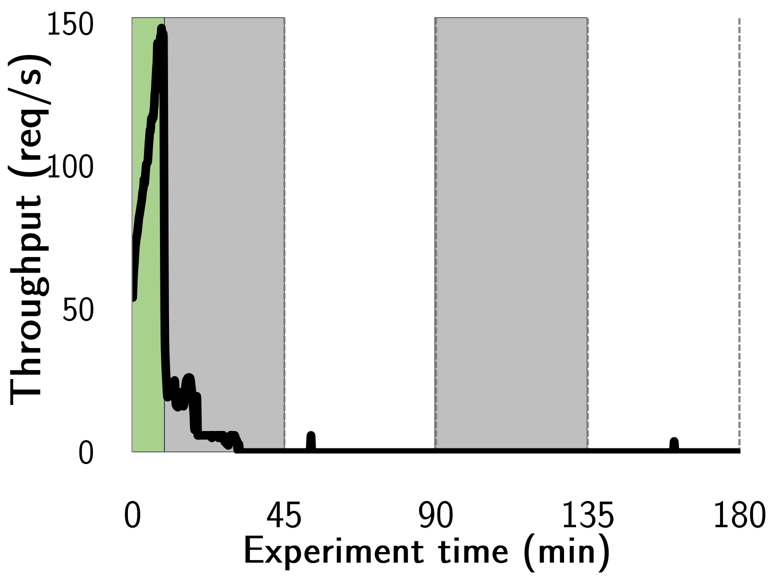


Figure 3.11: VM migration schedule = 45 minutes

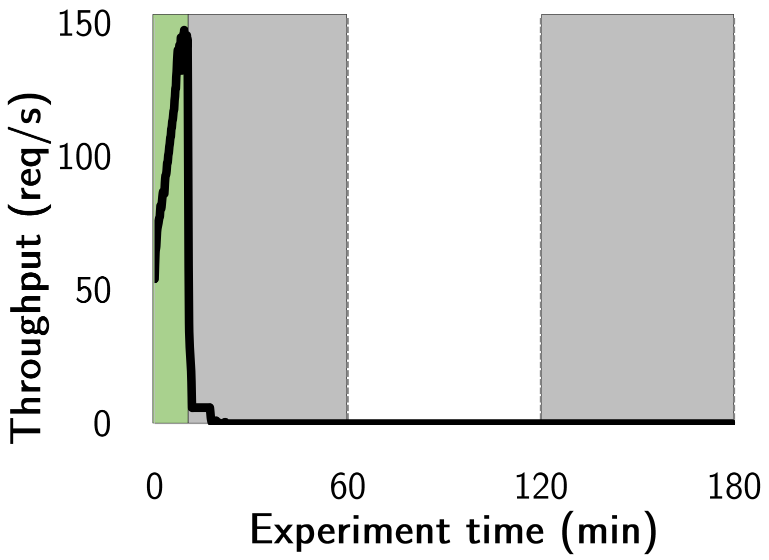


Figure 3.12: VM migration schedule = 60 minutes

We highlight the following conclusions:

1. Timing plays a crucial role in the VM migration-based MTD effectiveness.
2. In some scenarios, as presented in our experiment results, the system does not recover even after MTD 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 minutes of VM migration interval.

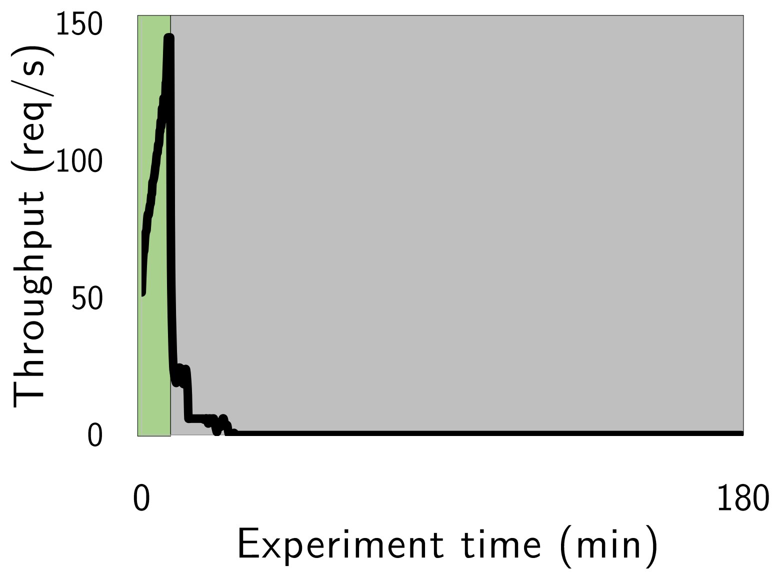


Figure 3.13: System without MTD

3.4 Threats to Validity and Limitations

The empirical observations discussed in this chapter are subject to threats to validity and limitations. Although we put our best effort into selecting and testing a pertinent setup, our results might be under some non-negligible limitations. Below, we comment on some of these limitations.

Testbed size

The considered testbed is minimal when compared to large data centers. Actually, it comprises only the necessary components to conduct the experiment and to guide the model design in the next chapters. As one may argue about the experiment's validity, we emphasize that the considered environment is a building block for larger virtualized environments. In fact, to allow VM migration, we need at least two physical hosts and one VM. As stated previously, there are advantages to using small-sized environments, such as the easier reproduction and instantiating of the proposed experimental campaign.

KVM-oriented aging workload

The software aging experiment adopts only a workload based on KVM hypervisor. Machida et al. [2012] also highlighted software aging indicators in a different hypervisor (*Xen*). It is not possible to ensure that the software aging behavior repeats in all available hypervisors. Nevertheless, KVM and *Xen* are among the most popular hypervisors nowadays, which increases the relevance of VM migration as software rejuvenation.

Selected attack workload

The proposed attack workload adopts only a non-persistent tactic, meaning that the attack progress does not resist VM migration. Besides that, here we are only considering a VM to host the attack. In a more comprehensive setting, the experimental campaign should include persistent tactic attacks as well as VM-targeted attacks.

3.5 Summary

This chapter introduced our baseline system architecture. We presented the details of using the VM migration technique as support for software rejuvenation and MTD. Whenever needed, the following chapters may include additional details of specific scenarios particularly analyzed in their context. Besides the fundamentals of VM migration technique, this chapter presented experimental observations to show the effectiveness of the technique in a practical way.

Our empirical observations show the validity of using time-based VM migration as support for software rejuvenation and MTD. In fact, the results obtained from the software rejuvenation experiment show an immediate system recovery from aging-related effects after VM migration completion. The complete set of outputs confirmed the efficiency of VM migration to counteract KVM software aging. Likewise, the MTD empirical observation validates the use of time-based VM migration as MTD against *host-based attacks*. Furthermore, the results highlighted the schedule of migrations is a determinant for the MTD efficiency.

The following chapter introduces our first set of analytical models, which will be adapted in the subsequent chapters. The proposed SPN models aim at evaluating the system performability, and are built on top of the baseline system architecture described above and consider software rejuvenation based on time-based VM migration scheduling. Given that previous works (e.g., [Melo et al., 2013b,a; Machida et al., 2013]) extensively covered this topic, we decided to take a step forward to contribute to the state-of-the-art in the field. Specifically, we propose a comprehensive performability evaluation of the time-based VM migration technique. We also consider additional aspects such as bursty workload occurrence.

Chapter 4

Performability of Virtualized Systems with VM Migration

Software aging imposes resource consumption overhead. Consequently, it also affects system performance. It is important to define rejuvenation schedules to minimize the probability of software aging-related resource exhaustion. However, in realistic situations, we should consider other aspects of system resource consumption (besides software aging) to provide better evaluations. For example, bursty workloads may lead to faster resource exhaustion.

We propose a set of SPN models for performability and dependability evaluation of a system with VM migration scheduling. Wang et al. [2007] presented a performability model for a system with varying workloads that provides a relevant background for our performability modeling, namely regarding some parameters (rates) for the evaluation. In their paper, Wang et al. cover the aspect of workload variation in a system with rejuvenation. We take a step forward and cover VM migration, burst workloads, and an extended set of metrics such as availability and reliability. Specifically, we aim to answer the following research question:

RQ_{main}*:* What are the performability and dependability levels of a virtualized system with VM migration subject to software aging and a bursty workload?

We divided this question into three subquestions:

- RQ_1*:* What is the VM migration scheduling policy that maximizes the system availability?
- RQ_2*:* What is the VM migration scheduling policy that maximizes the system throughput?
- RQ_3*:* What are the system reliability levels when applying the VM migration policies that maximize system availability?

We consider three case studies to exercise our models. The first shows the system steady-state availability evaluation; the second brings the system steady-state throughput results; the third is about the reliability evaluation. We consider a set of scenarios in each case study covering different *Asset Classes*. An *Asset Class* is the definition of the associated bursty workload conditions, namely, duration, intensity, and probability. Specifically, we consider five *Asset Classes* ranging from light bursts to heavier bursts. Note that these *Asset Classes* are only illustrative for our evaluation purposes.

This chapter presents the last version of our baseline availability model, which comes from a **series of papers that we published on this topic**, namely [Torquato and Vieira, 2018], [Torquato et al., 2020a] and [Torquato et al., 2022a]. The last one [Torquato et al., 2022a] provides the content for this chapter.

The rest of the chapter is organized as follows. Section 4.1 presents the research approach adopted and details on the target system environment. Section 4.2 discusses the proposed models for performability evaluation. Section 4.3 discusses and analyses the model results. Section 4.4 presents the limitations and threats to validity. Finally, Section 4.5 concludes this chapter.

4.1 Approach and Assumptions

Our approach for the performability assessment consists of four steps:

1. **Scope definition** - we evaluate the performability and dependability of a virtualized system (e.g., a private cloud) with software rejuvenation based on VM migration. The system is liable to suffer bursty workloads.
2. **Metrics definition** - in the performability and dependability domain, we focus on three specific metrics: availability, reliability, and throughput.
3. **Model design** - we adopted a hierarchical model approach, with the performance model (M2) receiving inputs from the availability model (M1). We also propose a specific model (BURST CYCLE MODEL) to cover the bursty workload occurrence uncertainties (i.e., intensity, cycle, duration, and probability).
4. **Model analysis and results** - we propose three case studies, one for each desired metric. In these case studies, we evaluate the impact of the VM migration schedule on system availability and performance. We also analyze the system reliability in the first month.

Figure 4.1 presents a flowchart of the proposed methodology including the keywords related to each step. The system architecture follows the definitions presented in Section 3.1. Compared to the current literature, we take a step forward by including the pre-copy phase of VM live migration. Besides that, we use marking-dependent firing rates to cover software aging effects and include the bursty workload occurrence. Next, we present further details on the environment considered, emphasizing the additional assumptions for the modeling.

4.1.1 Failure Modes

The VM is the most critical component for the system availability, meaning that the system is available only when the VM is running. Therefore, any system behavior that causes an interruption in the VM running results in system unavailability. The first failure mode is the VM non-aging failure. Software, PM host hardware, or operating system issues can lead the VM to service interruption, meaning unavailability. As the VM depends on the *Main Node* to run, any *Main Node* interruption will cause system unavailability. *Standby Node* failures

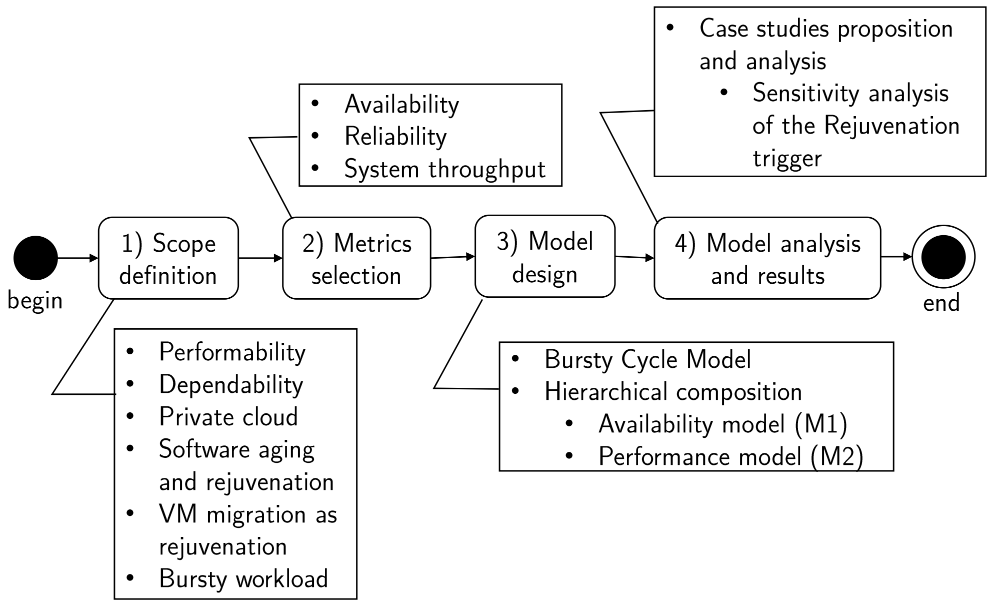


Figure 4.1: Flow of the Research Methodology

do not cause unavailability directly. However, *Standby Node* unavailability prevents VM migration. Likewise, we assume that, as long as the *Standby Node* is active, there is the capacity to receive the VM migration. As mentioned in the previous chapter, the system suffers a short downtime on each VM migration operation. Thus, frequent migration may have a severe impact on overall system availability. Besides that, the system may fail due to resource exhaustion because of software aging. We assume that the operations for system repairing after a resource exhaustion failure comprise software rejuvenation actions like OS reboot or application restart.

4.1.2 Bursty Workload Modeling

Bursty workload occurrence accelerates the depletion of resources, leading to quicker failure due to resource exhaustion. The modeling of such an unexpected event as burst workloads is a complicated task because the burst characteristics are often related to the targeted system. Depending on the target, the burst duration, probability, and intensity can be higher or lower. Therefore, setting up a generic approach for bursty workload modeling is unpractical due to its myriad possibilities. However, proposing a set of possible scenarios of bursty workloads can help support the design of Service Level Agreements (SLAs) or internal rejuvenation policies.

We consider the following four main characteristics of the bursty workloads: **cycle** - the supposed time between bursts; **burst probability** - the probability of burst occurrence; **burst duration** - time that the system spends under the bursty workload; and **burst intensity** - the severity of the workload submitted to the system. We take these four characteristics into account in a specific model named ”Burst cycle”.

Figure 4.2 presents a diagram of the BURST CYCLE model. The circles represent the system states, the continuous arcs represent the time delay for the state transition, the dashed arcs represent the state transition based on probability (instead of time delay), and the gray arc with a circle represents an attribute of a specific system state.

The initial state of the BURST CYCLE model is the ***start*** state, which represents the start of the time cycle between burst occurrences. After the **Cycle**, the system state goes from ***start*** to ***end***. In the ***end*** state, based on probability (***burstProbability***), the system can restart the cycle (going back to the ***start*** state), or suffer the bursty workload (moving the system state from ***end*** to ***underBurst***). The ***underBurst*** state has the ***burstIntensity*** attribute, which, as mentioned earlier, indicates the severity of the workload submitted by the bursty workload. After the time delay ***burstDuration***, the cycle for the next burst begins. We incorporate the behavior of the BURST CYCLE model using an SPN model, as presented in the Section 4.2.

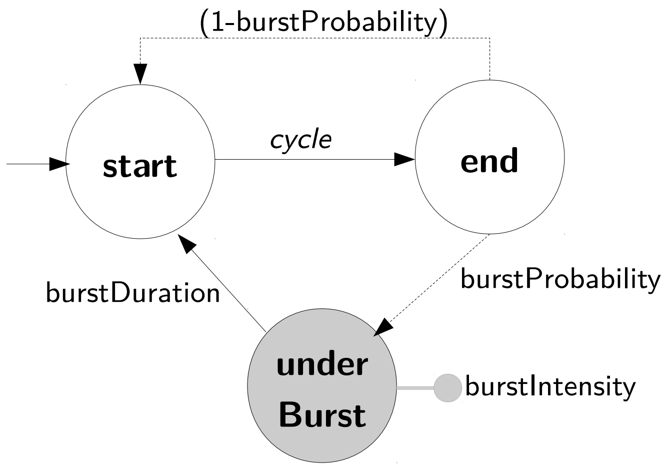


Figure 4.2: A diagram for the BURST CYCLE model

We apply the BURST CYCLE model for the bursty workload. We used parameters from previous studies for our model evaluation (see Tables 4.3 and 4.4 in Section 4.3), but it is possible to customize the BURST CYCLE model parameters. However, as it is hard to find specific datasets to use as input for the analysis in most situations, the BURST CYCLE model approach enables the system managers to set up reasonable scenarios based on their knowledge about the environment. Therefore, they can build better SLAs and internal policies, considering the possibilities of bursty workload occurrence. Note that, in our research, the *VM* is the target for the bursty workload and not the *Main Node*. Thus, a VM migration also switches the workload to the VM migration target host during a burst.

4.2 Models

This section presents the models proposed for the evaluation. The first is a SPN model for availability evaluation, while the second is a performance evaluation

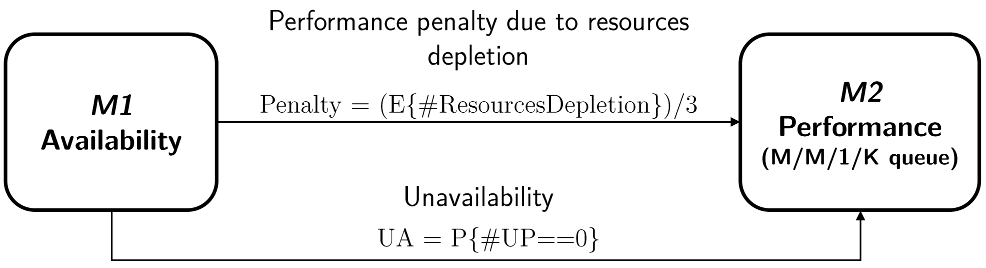

Performance penalty due to resources depletion
Penalty = (E{#ResourcesDepletion})/3
M1
Availability
M2
Performance
(M/M/1/K queue)
Unavailability
UA = P{#UP==0}

Figure 4.3: Models relationship

model, which is an M/M/1/k queue. The models obey the relationship presented in Figure 4.3. The availability model (*M1)* provides two inputs to the performance model (*M2*). The first is the performance penalty (`Penalty`) due to resource depletion. We compute `Penalty` by observing the (`ResourcesDepletion` place) of the availability model. The number of tokens inside the `ResourcesDepletion` place serves as an indicator of the level of resource depletion. Besides that, *M1* also provides the system unavailability (`UA`) as input for *M2.* We compute `UA` observing the probability of the absence of tokens in the place `UP`, which is the place used to represent the system availability. More details of the interactions between *M1* and *M2* are in the next sections. Note that we consider M/M/1/k as this is one of the most used queuing models for client-server applications. Nevertheless, it is possible to adapt *M2* to other scenarios by including the effects of `Penalty` and `UA` in other queueing models (see Section 4.2.2 for details). Literature exists to support the design of other queuing models based on Petri Nets [Kounev, 2006; Bause, 1993; Siddiqui et al., 2020].

We use two separated models for performability and dependability evaluation to deal with the *stiffness* problem caused by the use of transitions with considerable differences in the firing delay magnitude [Bobbio, 1990]. In our modeling framework, we have the mean time to failure of the *Main Node*, which is above a thousand hours, and the system service time, measured in milliseconds. By using the model decomposition, we can mitigate *stiffness* as the evaluations of performability (with delays of milliseconds) and dependability (with transition delays of months) are performed separately.

4.2.1 Availability Model

The proposed availability model has three sub-models (Figure 4.4): i) Clock model; ii) Burst cycle model; and iii) System model. These models interact using guard functions and transitions with marking-dependent firing rates, which will be later explained.

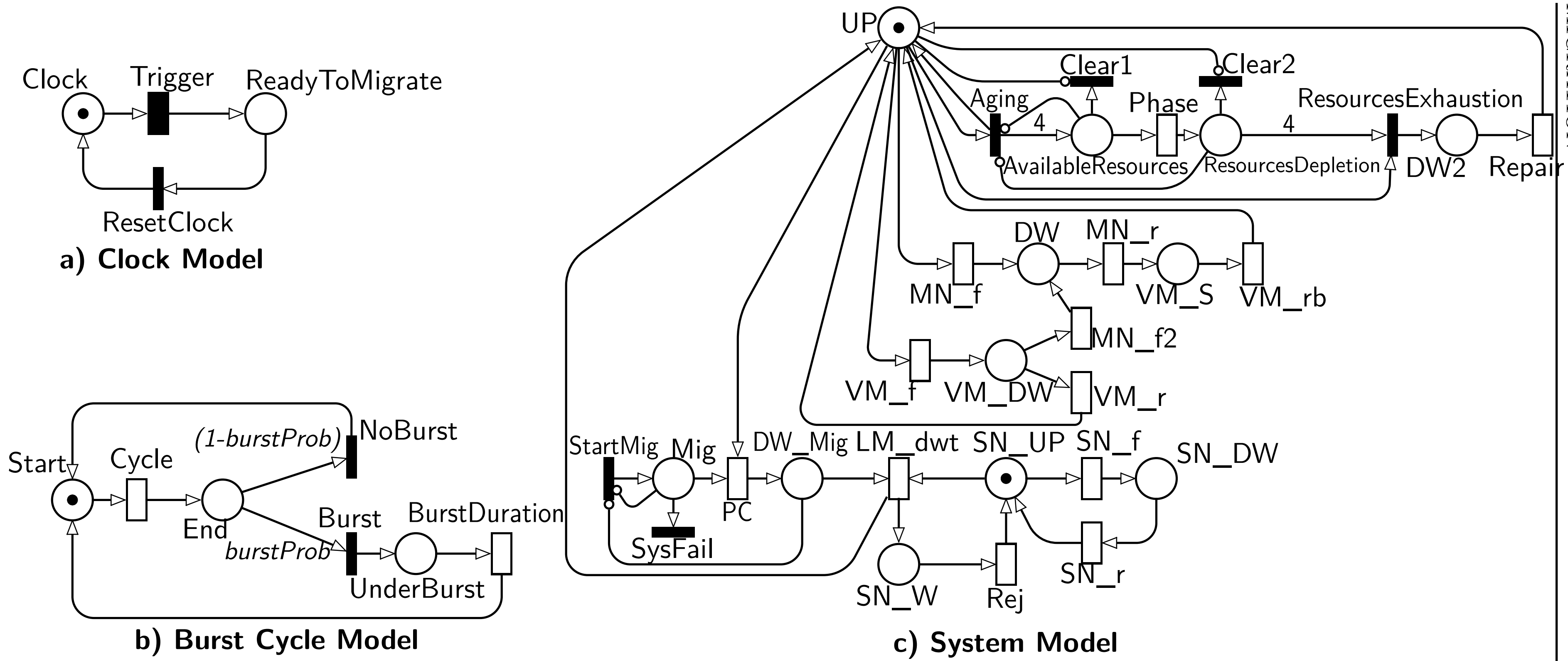


Figure 4.4: Availability Model

The first model is the Clock model. The Clock model has the `Clock` and `ReadyToMigrate` places and the `Trigger` and `ResetClock` transitions. The Clock model represents the behavior of a software component responsible for the VM migration scheduling process. Thus, at the initial state, the `Clock` place has a token enabling the firing of the deterministic transition `ReadyToMigrate`, representing that the time counting for the VM migration is active. Then, the `Trigger` transition firing removes the token from the `Clock` place and puts a token in the `ReadyToMigrate` place. The `ReadyToMigrate` place with a token represents that the system reaches the planned schedule for VM migration. However, besides the planned schedule, the VM migration (`StartLM` transition firing) depends on a few more conditions: i) *Main Node* and *VM* running (token in the `UP` place); and ii) *Standby Node* running (token in the `SN_UP` place). We embedded all these conditions in the `StartLM` transition using guard functions. Table 4.1 contains the meaning and the associated guard functions of all the immediate transitions. The transition `ResetClock` represents the start of the time counting for the next VM migration. The `ResetClock` firing depends on the presence of the token in the place `ReadyToMigrate` and in the place `Mig`, representing that the system clock restarts its cycle right after the beginning of the VM migration. In this model, we assume that the clock works in this cycle permanently.

Table 4.1: Immediate transitions and associated guard functions

Transition	**Meaning**	**Associated Guard Function**
`StartMig`	VM migration start	`(#ReadyToMigrate>0)` *AND* `(#UP>0)` *AND* `(#SN_UP)`
`ResetClock`	Start time counting for next VM migration	`(#Mig>0)`
`SysFail`	System fail during migration	`(#UP==0) OR (#SN_UP==0)`
`Burst`	Immediate transition which represents the burst occurrence	no guard functions
`noBurst`	Immediate transition, which indicates that there is no burst occurrence in the cycle iteration	no guard functions
`Aging`	Start of the resources depletion	no guard functions
`Clear1` and `Clear2`	Resources depletion clearance due to software rejuvenation or recovery after a failure	no guard functions
`ResourcesExhaustion`	Resources exhaustion failure	no guard functions

The Burst Cycle model characterizes the bursty workload. It obeys to the behavior described in the Section 4.1.2. The token in the Start place enables the transition Cycle firing, representing the supposed time duration between bursts. To improve the uncertainty aspect of this modeling process, we prefer to use the exponential distribution instead of deterministic in the Cycle transition. Cycle transition firing removes the token from the Start place and deposits a token in the End place. The presence of the token in the End place enables the transitions Burst and noBurst concurrently. As presented in Kuchárik and Balogh [2019], we assigned different weights on each arc for the transitions Burst and noBurst to represent the probability of burst occurrence. The arc from End to the transition Burst has $weight = burstProb$ ($burstProb$ is a variable related to the burst occurrence probability), and the arc from End to noBurst transition has $weight = (1 - burstProb)$. In the case of the noBurst firing, the token goes back to the Start place, representing the cycle restart. Otherwise, in the case of Burst firing, the transition removes the token in the End place and puts a token in the UnderBurst place, representing that the system is suffering a bursty workload. The BurstDuration transition represents the time duration of the bursty workload as long as the system is under a burst (i.e., the model with token presence in the place UnderBurst), it suffers a resource depletion acceleration. This acceleration is related to the **Burst intensity** mentioned in Section 4.1.2. We model this behavior using a marking-dependent firing delay on the transition Phase of the System model. The Phase transition represents the resource depletion progress. Table 4.2 presents the details of the transitions with marking-dependent firing delays. The BurstDuration firing removes the token from the UnderBurst place and puts a token in the Start place, representing the cycle restart.

The System model intends to cover three system aspects: i) *behavior of non-aging failures and their repairs*; ii) *VM migration*; and iii) *resources depletion due to software aging*. Next, we highlight the sections of the model that represent each one of these behaviors.

First, we consider system non-aging failures (e.g., hardware or Operating system failures). At the initial state, the System Model has a token in the UP place. UP place with tokens represents the *Main Node* and the hosted *VM* running. And, the absence of tokens in the UP place represents the system's unavailability. From the initial state, the *Main Node* can suffer a non-aging failure (MN_f transition firing). The MN_f transition firing removes the token from the place UP and puts a token in the place DW. The *Main Node* repair has two steps: 1) *Main Node* recovery (MN_r transition firing, moving the token from DW to VM_S place) and 2) *VM* reboot (VM_rb firing, returning the token to the UP place). The system can also suffer a *VM* non-aging failure. The transition VM_f firing represents a *VM* non-aging failure occurrence. After a *VM* failure, there are two possibilities to recover the system: a VM repair (transition VM_r firing), or a subsequent *Main Node* failure (MN_f2 firing, moving the token from VM_DW to the DW place). We also cover the non-aging failure and repair processes in the *Standby Node*. SN_UP and SN_DW places represent the *Standby Node* status related to the non-aging failures and repairs. SN_UP place with tokens represents the *Standby Node* running and ready to receive *VM* migration. The transitions SN_f and SN_r represent a *Standby*

Table 4.2: Transitions with marking-dependent firing delays

Transition	Marking-dependent firing delay	Meaning
`Phase`	`IF(#UnderBurst>0): phaseDuration/BurstIntensity ELSE phaseDuration`	If the system is under bursty workload, the resource depletion progress phase is accelerated by the factor of the **Burst intensity**. Otherwise, the resource depletion progress follows the expected time delay (`phaseDuration` variable).
`PC`	`IF(#ResourcesDepletion==0): Precopy ELSE Precopy+Precopy*(#ResourcesDepletion/3);`	This marking-dependent firing delay captures the influence of the number of dirty memory pages on the VM live migration process [Salfner et al., 2011; Strunk, 2012].

Node non-aging failure and repair, respectively. *Standby Node* unavailability (token in the `SN_DW` place) affects the system availability indirectly as it prevents the software rejuvenation for aging failures avoidance.

Second, `StartLM` transition represents the VM migration start, and it has an associated guard function (see Table 4.1). We assume the Pre-copy VM live migration [Clark et al., 2005] as the *VM* migration method. The pre-copy algorithm has two main phases: i) Pre-copy phase (transition `PC`) - transfer of the memory pages from the *Main Node* to the *Standby Node*; and ii) Downtime phase (transition `LM_dwt`) - transfer of the processor state and VM migration acknowledgment. `StartMig` firing deposits a token in the `Mig` place. We used inhibitor arcs[1] to indicate that a new migration can only occur after the finishing of the previous. `Mig` transition with tokens represents that the VM migration is in the Pre-copy phase. Dur-

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

ing the Pre-copy phase, the system continues to run (the token stays on the `UP` place). `SysFail` transition serves to represent possible system failures (i.e., *Main Node*, *VM* or *Standby Node* failures) during the Pre-copy phase. We also embed this behavior using guard functions. A system failure during the Pre-copy phase implies VM migration abort. Thus `SysFail` transition removes the token from `Mig` place. As presented in Maziku and Shetty [2014]; Liu et al. [2011]; Akoush et al. [2010]; Voorsluys et al. [2009], the amount of dirty memory pages affects the VM migration latency. To represent this behavior, we used a marking-dependent firing delay in the `PC` transition (see Table 4.2). The marking-dependent firing delay increases the supposed delay for the Pre-copy phase (`Precopy` variable) observing the status of system resources depletion (i.e., number of tokens in the `ResourcesDepletion` place). After the completion of the Pre-copy phase (firing of `PC` transition), the system enters the Downtime phase (token in the `DW_Mig` place). In the Downtime phase, the system is unavailable. We represent this behavior by removing the token from `UP` place after `PC` transition firing. The system returns to be available after the *VM* migration completion (`LM_dwt` transition firing). As mentioned in Section 3.1, the previous *Main Node* (i.e., *VM* migration source) will pass through software rejuvenation before assuming the role of the *Standby Node*. Thus, `LM_dwt` firing puts a token in the `SN_W` representing that the previous *Main Node* is waiting for the software rejuvenation. Transition `Rej` represents the software rejuvenation action, and its firing replaces a token in the `SN_UP`. This behavior represents the rejuvenation completion and that the *Standby Node* is ready to receive *VM* migrations.

Finally, about the *resources depletion due to software aging*: to represent the resource depletion behavior, we adopted a four-phase Erlang distribution, as the Erlang distribution is suitable to represent IFR behavior [Gupta et al., 2010]. We used an Erlang subnet with four phases to represent the IFR in the System Model. The Erlang subnet is in the upper part of the System Model. The places `AvailableResources` and `ResourcesDepletion` are related to the system resources depletion status. The number of tokens in the `AvailableResources` denotes the amount of resources available, and the number of tokens in the `ResourcesDepletion` place denotes the resources depletion status.

At the initial state, the transition `Aging` fires swapping[2] the token from the `UP` place. The same transition deposits four tokens in the place `AvailableResources`. The number of tokens denotes the amount of resources still available for the *Main Node* usage. As time passes, the *Main Node* starts to accumulate software aging status. The transition `Phase` firing represents the resources consumption progress, which removes the tokens from the `AvailableResources` and deposits tokens in the `ResourcesDepletion` place. If the software aging status persists in the system, it can suffer a resource exhaustion failure (`ResourcesExhaustion` transition). We highlight that the `Phase` firing rate is also adjusted when the system is under bursty workload, meaning faster resource exhaustion. `ResourcesExhaustion` firing removes the token from the `UP` place and puts a token in the `DW2`. After a resource exhaustion failure, the system recovery has three phases: i) detection of resource exhaustion, ii) software management to clean residuals, and iii) com-

[2]Receiving and returning

plete OS reboot. We model these steps in a single exponential transition named `Repair`. After `Repair` firing, a token returns to the `UP` place, representing that the system is available again.

We obtain two metrics from the availability model. The first is the system availability (A). We compute the system availability as the probability of token presence in the place `UP` ($A = P\{\texttt{\#UP} > 0\}$). We use *Availability* to obtain secondary metrics as unavailability (UA) and annual downtime in hours per year[3] (Dwt). The expressions are as follows: $UA = 1 - A$ and $Dwt = 8760 \cdot UA$. Note that, as we are computing steady-state availability, it is possible to obtain the downtime for other intervals (besides one year). For example, we can compute the monthly downtime in minutes[4] using $Dwt = 43200 \cdot UA$. And, as mentioned earlier, besides the availability-related metrics, we also computed the *Penalty*, using the following expression $Penalty = E(\texttt{\#ResourcesDepletion})/3$, which is a normalized value of the expected number of tokens in the `ResourcesDepletion` place. We take account of resource depletion accumulation when the place `ResourcesDepletion` has tokens. Therefore, the possibilities are `ResourcesDepletion` with one, two, or three tokens. For normalization of the *Penalty* metric, we divided the expected number of tokens in place `ResourcesDepletion` by three.

4.2.2 Performance Model - M/M/1/k Queue

In some situations, it is important to understand the system performance when considering bursty workloads. Thus, we present a queueing model for such evaluation. In this case, the performance may degrade due to software aging accumulation issues, thus the computed metrics are related to system performability [Meyer, 1992]. This performance model aims to provide system throughput results. The queue model receives a variable named `Penalty` from the availability model output. We obtain `Penalty` observing the steady-state expected number of tokens in place `ResourcesDepletion`. We use the variable `Penalty` to adjust the service time according to the system resource depletion. Besides that, we also cover the influence of system unavailability in the performability metric.

We consider that the *VM* runs a user application or a service that receives and processes requests obeying an M/M/1/k queue model [Kleinrock, 1975]. Machida et al. [2013] adopted the same approach to a similar problem. Figure 4.5 presents the SPN model used for M/M/1/k queue metrics calculation. Transition `arrival` represents transaction arrival in the system. The transaction acceptance depends on the available buffer space (i.e., queue capacity) and system availability.

We parameterized the place `buffer` with k tokens. The k variable represents the queue capacity. The transition `arrival` firing removes one token from the `buffer` place and deposits one token in the `queue` place, representing a transaction arrival in the system. Transition `service` represents the transaction processing. After `service` firing, the token returns to the `buffer` place, representing the finishing of a transaction processing.

[3]We consider a year with 365 days.

[4]Considering a month with 30 days. $30 \cdot 24 \cdot 60 = 43200$

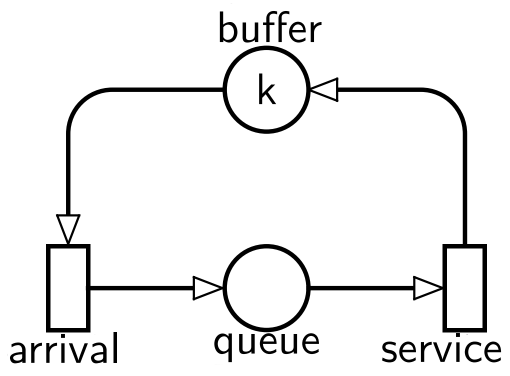


Figure 4.5: Performance Model - SPN for a M/M/1/k queue

System throughput is the rate of served transactions in a limited period. We computed system throughput by observing the effective number of transactions that enter the system. The first step of the evaluation is to use the variable *Penalty* in the performance model. In some of our previous experiments [Torquato et al., 2017], we noticed that a generic Web server under software aging effects has a service rate of about one request per second. Based on our previous experimentation, we consider that when the *Penalty* assumes its maximum value, the system service time (i.e., firing rate of transition `service`) will be decreased to one request per second. We changed the firing rate of the transition `service` accordingly to each proposed scenario. We obtain the effective rate of transactions accepted in the queue ($RTAQ$). We obtain $RTAQ$ using the following expression $RTAQ = P\{\texttt{\#buffer > 0}\} \cdot \lambda$, where λ is the firing rate of the transition `arrival`. However, we still have to consider system unavailability in system steady-state throughput. Thus, we used the expression $ST = RTAQ \cdot A$ to compute the system throughput (ST), where A is the system availability.

4.3 Case Studies

We used the TimeNet tool for the availability model design and evaluation [Zimmermann, 2017] and the Mercury tool [Maciel et al., 2017] for the performance analysis. TimeNet has a friendly graphical interface and provides instantaneous results, while Mercury has an easy-to-use script language that facilitates sensitivity analysis using non-linear parameter variation.

We used the values in Table 4.3 as default values for our evaluations. We obtained these values from [Wang et al., 2007] and [Kim et al., 2009][Torquato et al., 2018b]. Note that these values are only for reference and should be adjusted whenever real-scenario values are available. However, these are the most representative values that we can find to feed the models as they were published in reputed journals.

Table 4.3: Parameters used in the timed transitions

Parameters		**Values**
Availability Model		
Transition Name	***Description***	***Mean time***
`Trigger`	Interval to VM Live Migration	1 $\rightarrow$ 720 hours
`Cycle`	Supposed cycle between burst occurrence	24 hours
`BurstDuration`	Bursty workload duration	60, 120, 240, 360 or 480 seconds[1]
[1] Depending on the scenario		
`AgingPhase`	Time to Aging (Phases)	62.5 hours[2]
[2] We adjust the time to aging observing the burst occurrence		
`Repair`	Time to system recovery after a resources exhaustion failure	1 hour
`MN_f, MN_f2`	*Main Node* Failure Delay	1236.706 hours
`MN_r`	*Main Node* Repair Delay	1.094 hours
`SN_f`	*Standby Node* Failure Delay	1236.706 hours
`SN_r`	*Standby Node* Repair Delay	1.094 hours
`VM_f`	Virtual Machine Failure Delay	2880 hours
`VM_r`	Virtual Machine Repair Delay	30 minutes
`VM_rb`	Virtual Machine Reboot Delay	5 minutes
`PC`	VM Live Migration pre-copy phase time	72 seconds[3]
[3] We adjust the service rate accordingly to the number of tokens in the `ResourcesDepletion` place		
`LM_dwt`	VM Live Migration Downtime	4 seconds
`Rej`	Rejuvenation Node Delay	2 minutes
Performance Model		
Transition Name	***Description***	***Rate***
`arrival`	Transaction arrival	1000 requests per second
`service`	Service rate	1500 requests per second[4]
[4] We adjust the service rate accordingly to the *Penalty* variable		

In the case studies, we focus on finding the rejuvenation schedule to maximize system availability and system throughput. First, we search for the best rejuvenation schedule using a graphical sensitivity analysis, which explores the output of the models by varying the time interval for the VM migrations (`Trigger` firing delay) from 1 to 720 hours (a month) using a one-hour step. Second, we present a sensitivity analysis for the system throughput metric. Thirdly, once we find the availability-oriented rejuvenation schedule, we propose the last case study to verify system reliability in the first month of the system running.

For all the scenarios, we assume that the service hosted in the virtualized environment is liable to suffer a bursty workload. Depending on the asset, the burst may be more or less likely to occur. To represent the different asset classes, we propose five different scenarios, as shown in Table 4.4.

Table 4.4: Asset Classes definitions

Asset Class#	**Burst Probability (%)**	**Burst Intensity**	**Burst Duration**
0	0.01	2000	60 seconds
1	0.1	4000	120 seconds
2	1	6000	240 seconds
3	5	8000	360 seconds
4	10	10000	480 seconds

The case studies are: Availability (Section 4.3.1), System Throughput (Section 4.3.2), and Reliability (Section 4.3.3).

4.3.1 CS #1 - Availability

Our goal in this case study is to study the rejuvenation schedule that maximizes the system availability and thus answer RQ_1. There are two main problems for availability in the scenarios covered in our study. The first is when applying frequent migrations: as each migration has an associated downtime, frequent migrations will degrade the steady-state availability. The second is due to resource exhaustion failures due to software aging and bursty workloads, where less frequent migrations may allow the system to reach resource exhaustion failures. Note that, in some situations, VM migration during bursty workloads may accelerate the exhaustion of the resources due to VM migration overhead. As mentioned earlier, we capture the influence of bursty workloads in the VM migration process using transitions with marking-dependent firing delays.

Figure 4.6 presents the availability results for each proposed asset class (the details are in Table 4.4). The black line represents the system availability when applying rejuvenation, and the gray line represents the system availability without rejuvenation (Baseline). We notice that the system availability has a peak in all the scenarios, which is the specific rejuvenation trigger that maximizes system availability. After the peak, we see a system availability decrease, tending to the baseline scenario. This is an expected result, as scarce VM migrations (i.e., longer time between migrations) lead the system to the baseline conditions (i.e., the system without rejuvenation). Table 4.5 presents these specific rejuvenation policies and their results regarding the annual downtime. We also notice that more severe bursty workloads produce worse availability results. The results from Asset Class #0 and #1 are nearly the same. Therefore, with a lighter bursty workload, we can apply the same rejuvenation policy to maximize system availability.

Table 4.5: Results - Availability

Asset Class#	**Baseline Downtime (h/yr)**	**Rej. Trigger (h)**	**Downtime (h/yr)**	**Downtime Reduction (h/yr)**
0	41.14	19	10.56	30.58
1	41.25	19	10.65	30.60
2	43.06	18	12.78	30.28
3	53.04	16	22.33	27.71
4	68.39	15	44.51	23.88

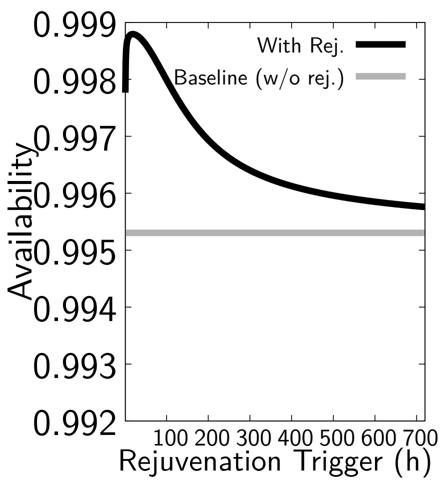
To provide a more comprehensive overview of the impacts of bursty workloads on the system availability, Figure 4.7 presents a comparison of the downtime reduction for all the scenarios. As we can see, in scenarios with more severe bursts, the downtime reduction is lower. In such cases, it is important to set up mechanisms to improve resiliency against bursty workloads.

In the analysis above, we fixed the VM migration downtime. However, in a real-world scenario, VM migration downtime may vary due to various reasons (e.g., amount of memory pages to be transferred, dirty pages rate, network bandwidth, VM migration technique). Therefore, it is important to study the VM migration downtime variation in the system's steady-state availability.

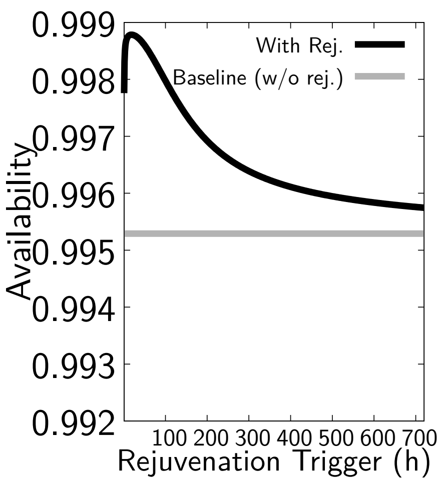
For a **sensitivity analysis** of the VM migration downtime parameter, we considered a VM migration downtime variation from 0.5 seconds to one minute with a 0.5-second step. The other parameters remain the same as presented in Table 4.3. We fixed the rejuvenation trigger (RT) using the results presented in Table 4.5. Figure 4.8 presents the outcomes.

We notice a similar linear increase in the system downtime in all observed scenarios. The difference in the curves of Asset 0 and Asset 1 (scenarios with lower risk of bursty workload occurrence) is negligible. In these curves, the results of 60 seconds of downtime for each VM migration is about 75% greater than the results with 0.5 seconds. In the higher risk scenario (Asset 4), the same difference is 22%. Meaning that the relative impact of longer downtime is lower when compared to lower-risk scenarios. Table 4.6 summarizes the sensitivity analysis results.

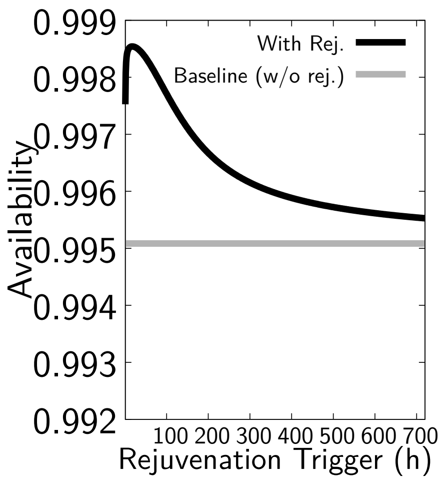
Presumably, the higher the risk of bursty workload occurrence, the worse the unavailability results. The results presented above indeed confirm the expected behavior. Nevertheless, they highlight, in a quantitative manner, the actual benefits achieved from software rejuvenation deployment. Furthermore, although we are considering uncertainties in a bursty workload scenario, the model is ready to capture expected events; e.g., a scheduled large backup operation or an expected



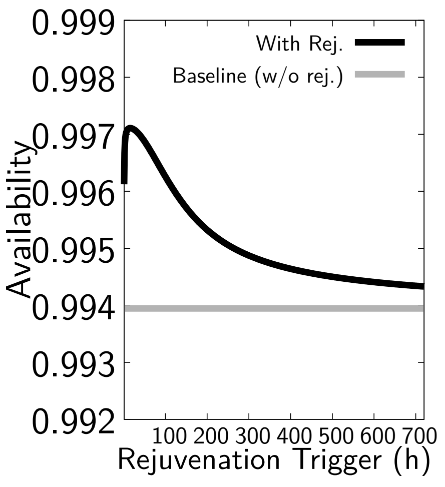
(a) Asset Class #0



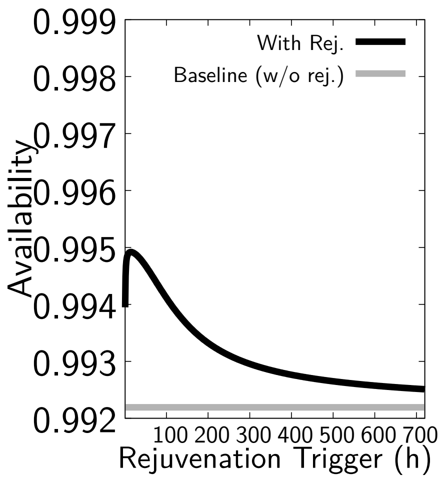
(b) Asset Class #1



(c) Asset Class #2



(d) Asset Class #3



(e) Asset Class #4

Figure 4.6: Availability of each scenario

season of high workload exposure. In this scenario, it is possible to adjust the **Burst Cycle Model** parameters to represent such situations.

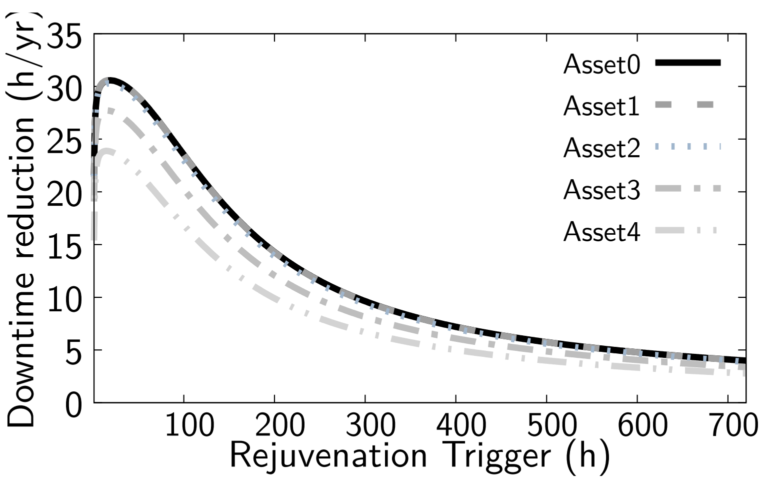


Figure 4.7: Downtime reduction (h/yr)

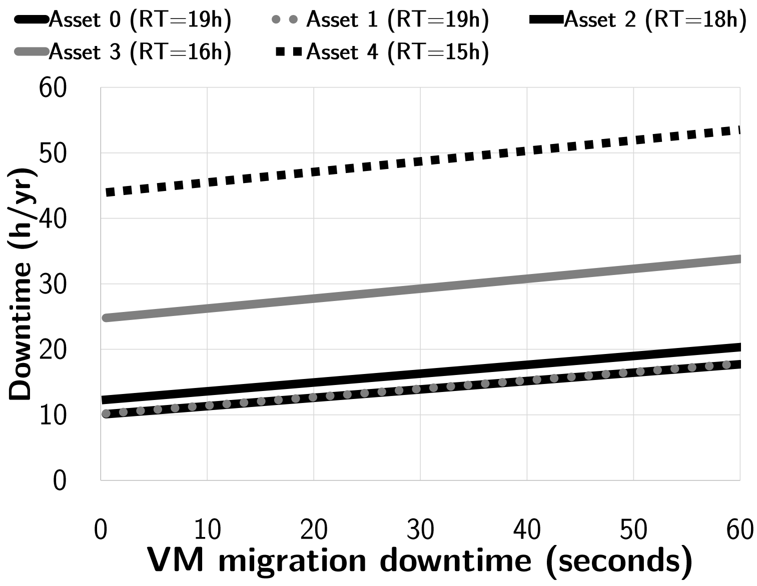


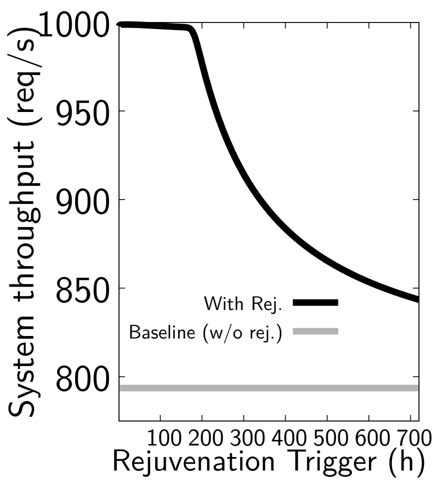
Figure 4.8: Sensitivity analysis of the VM migration downtime parameter

Table 4.6: Summary of VM migration downtime parameter sensitivity analysis

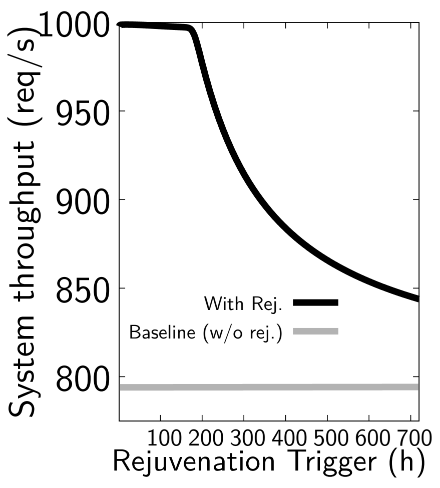
Asset Class#	**Unavailability with 0.5 seconds of downtime per migration (h/yr)**	**Unavailability with 60 seconds of downtime per migration (h/yr)**	**Relative difference**
0	10.12	17.72	75.14%
1	10.21	17.81	74.47%
2	12.31	20.33	65.17%
3	24.80	33.80	36.32%
4	43.94	53.53	21.80%

4.3.2 CS #2 - System Throughput

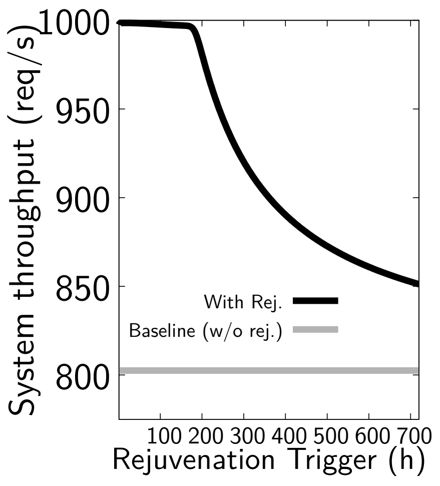
The goal of this case study is to find the rejuvenation policy that maximizes the system throughput, aiming at answering RQ_2. Figure 4.9 shows the results for all



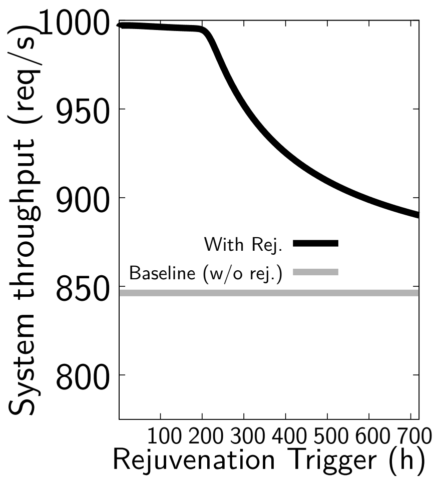
(a) Asset Class #0



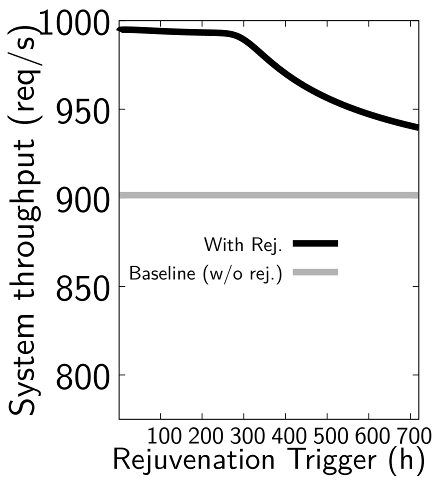
(b) Asset Class #1



(c) Asset Class #2



(d) Asset Class #3



(e) Asset Class #4

Figure 4.9: System throughput of each scenario

the scenarios considered. We noticed that in scenarios with shorter rejuvenation triggers, the system throughput stays at higher levels. After a certain point, we noticed a drop in the system throughput rate. We can draw the following conclusions from these results: i) systems with shorter migration intervals tend to persist in a lower *Penalty*. Thus, the service rate persists in higher levels, compensating for the lower availability levels due to frequent migrations, and ii) the baseline throughput is higher in systems with more severe bursty workloads. In the model analysis, we noticed that in such cases, after a burst, the system fails quickly. Therefore, the system steady-state *Penalty* is lower than in scenarios with a lighter bursty workload, and the steady-state system throughput tends to be higher than in the other scenarios.

Table 4.7 presents the best rejuvenation schedule for the proposed scenarios. The adopted policies for system throughput maximization are close to the results for system availability maximization. The last column presents the percentual improvement when comparing the baseline results and the results with rejuvenation. Like the previous case study, we noticed that the improvement is lower in scenarios with light bursty workloads than in the others in scenarios with heavy bursty workloads.

Table 4.7: Results - Sys. throughput (req/s)

Asset Class#	**Baseline System throughput (req/s)**	**Rejuvenation Trigger (h)**	**System Throughput (req/s)**	**Improvement**
0	793.5799	18	998.7940	25.86%
1	794.0027	19	998.7837	25.79%
2	802.5243	17	998.5409	24.43%
3	846.2617	16	997.1087	17.83%
4	901.4916	15	994.9192	10.36%

For the sake of comparison, we plot the throughput improvement of each scenario in Figure 4.10. As in the availability results, the results for *Asset Class* #0 and *Asset Class* #1 are nearly the same.

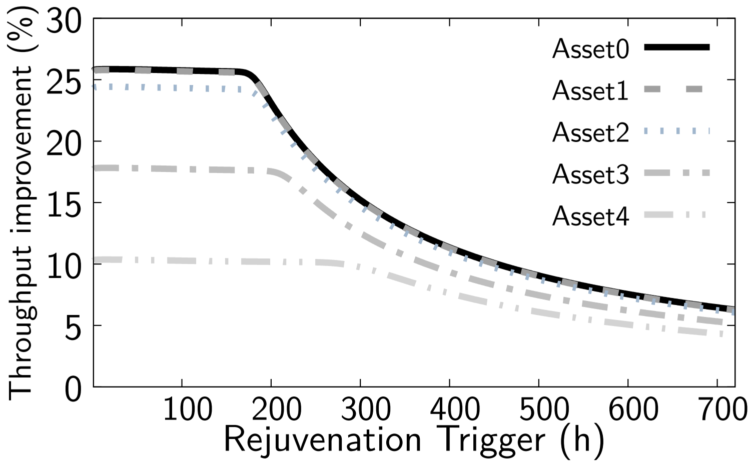
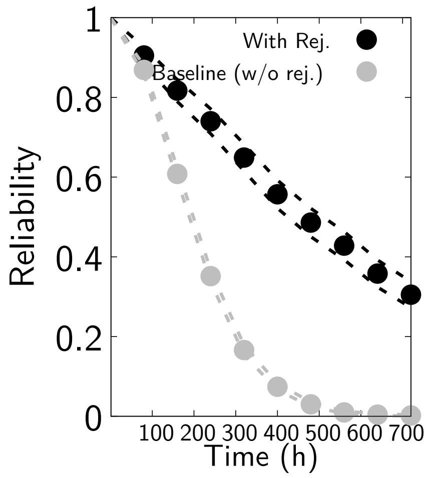
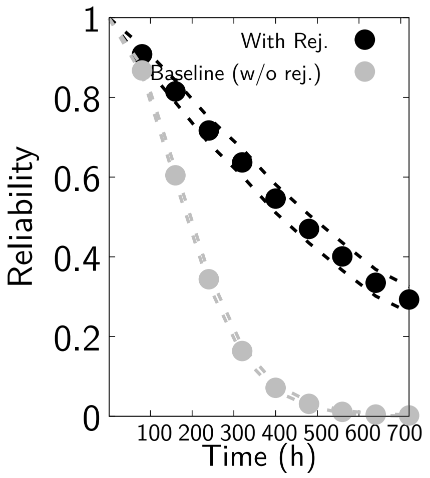


Figure 4.10: System throughput difference comparison

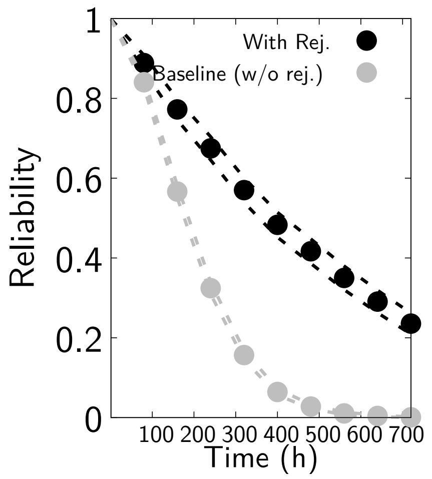
4.3.3 CS #3 - Reliability



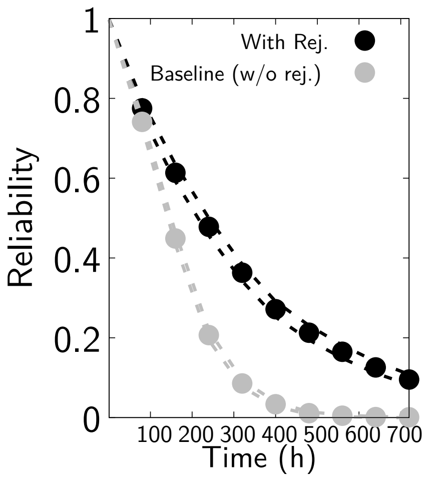
(a) Asset Class #0



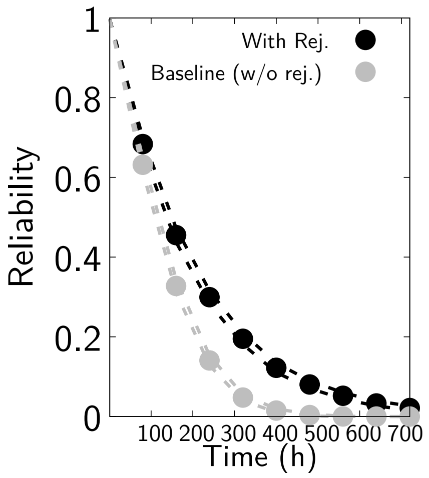
(b) Asset Class #1



(c) Asset Class #2



(d) Asset Class #3



(e) Asset Class #4

Figure 4.11: Reliability of each scenario

System reliability is related to service continuity [Avizienis et al., 2004], or the period that the system passes free from failures. In this case study, we investigate the system reliability when applying the availability-oriented rejuvenation policies (Table 4.5) to answer RQ_3. To conduct the reliability evaluation, we use the availability model without the repair transitions. Thus, we compute the reliability using the following expression: $\mathit{Reliability} = P\{\texttt{UP} > 0\}$. However, different from steady-state availability, system reliability is a transient metric. Thus, our goal is to calculate the probability of the system staying failure-free in its first month of running.

The results obtained are presented in Figure 4.11. The black dots represent the reliability results for the system with rejuvenation, and the gray dots represent the reliability results for the system without rejuvenation. The dashed lines represent the 95% confidence interval. We also performed linear regression in the Reliability results to extract functions representing the reliability curve ($R(t)$, where t is the time) when applying software rejuvenation policies. Table 4.8 presents a summary of the reliability results. The table also presents the coefficient R^2, which determines, in a range from 0 to 1, the fraction of the total variation explained by the obtained regression model.

We noticed a steeper reduction in scenarios with heavier bursty workloads. The rapid reliability decrease in the *Asset Class # 4* shows that the probability of a failure-free system is almost null at the 720th hour of continuous run. Therefore, after this point, the rejuvenation mechanism produces no improvement in the system reliability when compared to the baseline scenario. We also noticed that the reliability results for the first two scenarios (asset classes #0, #1) are nearly the same. In the *Asset Class* #0 scenario, the probability of a failure-free system in its first-month running is about 30%, while in the *Asset Class* #1 scenario, the probability is about 29%. Using the quadratic, polynomial, and logarithmic models for linear regression, we can achieve R^2 values above 0.999, meaning that the proposed functions can represent the reliability curve with substantial fidelity. Therefore, we can use these functions to approximate the reliability results for the desired scenarios.

To understand how long the system survives without rejuvenation, we calculated the *depletion point* (i.e., $\mathit{Reliability} = 0$) of the baseline scenarios. The *depletion point* results are in Table 4.9. These results highlight that without software rejuvenation, the system *failure-free* expected time is way lower when compared to the rejuvenation scenario. In all our rejuvenation-enabled evaluations, even in the scenario with a heavier workload, the system does not reach the *depletion point* after one month of running.

Table 4.8: Reliability results

Asset Class #	**Reliability without rejuvenation (720h)**	**Reliability with rejuvenation (720h)**	**Reliability with rejuvenation (linear regression)**	R^2
0	0.002 ± 0.001	0.305 ± 0.033	$R(t) = ((-6.286\text{e}{-4}) \cdot t + 1.003)^2$	0.9992
1	0.002 ± 0.001	0.293 ± 0.032	$R(t) = (4.350\text{e}{-7}) \cdot t^2 + (-1.319\text{e}{-3}) \cdot t + 1.008$	0.9991
2	0.001 ± 0.001	0.235 ± 0.026	$R(t) = (6.875\text{e}{-7}) \cdot t^2 + (-1.558\text{e}{-3}) \cdot t + 1.004$	0.9996
3	0.001 ± 0.001	0.095 ± 0.013	$R(t) = \exp(-3.270\text{e}{-3} \cdot t + 2.007\text{e}{-2})$	0.9996
4	0	0.021 ± 0.005	$R(t) = \exp(-5.393\text{e}{-3} \cdot t + 5.590\text{e}{-2})$	0.9995

Table 4.9: *Depletion point* results

Asset Class #	**Depletion point**
0	480 h
1	400 h
2	400 h
3	320 h
4	240 h

4.4 Threats to Validity and Limitations

Lack of experimental validation

The model-based evaluation provides flexibility to assess scenarios without direct intervention on a running system. The best scenario is when the model results are validated adequately against experimental results. However, this approach works better in performance evaluation scenarios, where the observed events are measured in response time or system throughput. On the other hand, system availability evaluation monitors failures and crashes. In a standard setting, the occurrence of such events is unexpected. Therefore, correctly measuring availability becomes difficult, as we need a representative sample of failures to compute components' MTTF. To mitigate this issue, we perform sensitivity analysis in various scenarios to observe system behavior under different circumstances.

Burst occurrence obeys a cycle

The bursty workload occurrence only follows a cycle. Therefore, the proposed model falls short of taking random burst occurrence into account. An approach to this problem is adjusting transition `Cycle` to other probability distributions. We decided to control model uncertainties (i.e., including more random events), aiming to achieve more consistent results. As a mitigation for this problem, we exercise the model under different burst probabilities.

4.5 Summary

This chapter presented SPN models for the performability and dependability evaluation of a virtualized system subject to software aging and bursty workload. The considered system applies VM migration scheduling as support for software rejuvenation. Our results include steady-state availability, steady-state system throughput, and system reliability.

About our main research question, RQ_{main}*: What are the performability and dependability levels of a virtualized system with VM migration subject to software aging and bursty workload?*, we concluded that such levels vary depending on the studied scenario. There is a specific rejuvenation schedule to maximize system availability and throughput. In scenarios with lighter bursty workload conditions (burst probability of 0.01% and 0.1%), the performability results are nearly the same. However, in the scenarios with heavier bursty workload conditions (burst probability of 1%, 5%, and 10%), the system performability degradation due to the bursty workload is substantial. In such scenarios, the rejuvenation policies tend to produce lower system performability improvement.

We covered the main aspects of software aging and rejuvenation and the uncertainties related to the bursty workload occurrence. Besides that, we also considered important details, such as the influence of burst occurrence in resource exhaustion and the influence of resource consumption levels in the VM migration process.

As VM migration is a standard maintenance tool for system managers, our research may help them to improve their understanding of the performability, availability, and reliability impacts of applying VM migration while considering significant concerns such as software aging and bursty workloads. The models and results presented here can be extended to similar scenarios. Moreover, they may be helpful in setting up virtualized environment management policies and SLAs.

A comprehensive evaluation of time-based VM migration deployment must also comprise the security aspect. To reach this goal, in the following chapter, we adapt the availability model proposed before and propose the RiskScore metric for security evaluations. This metric focuses on the steady-state behavior of the security status of the system.

Chapter 5

Availability and Security of VM Migration-Enabled Rejuvenation

Previous works highlighted indicators of software aging accumulation in Cloud components [Matos et al., 2012a]. Software aging is a cumulative process that can lead software to hangs or other failures [Dohi et al., 2020], whereas software rejuvenation is used to counteract software aging [Huang et al., 1995]. VM migration scheduling is an approach to reduce the downtime related to VMM software rejuvenation [Torquato et al., 2018a] [Torquato et al., 2017]. However, potential security issues of applying VM migration as support for VMM rejuvenation are still not understood. Besides that, security-aware software rejuvenation scheduling is hard to achieve due to the uncertainty related to security events [Cotroneo et al., 2014]. Pietrantuono and Russo [2019] also highlights the importance of studying the security impacts caused by software rejuvenation policies.

There are several studies on the availability evaluation of VM migration scheduling as support for VMM software rejuvenation [Melo et al., 2013a][Torquato et al., 2018b]. Mainly, these aim at finding the optimal rejuvenation schedule to maximize system availability, but none of them deal with security issues. From a security perspective, there are also some works on VM migration security, as presented by Oberheide et al. [2008], for example. However, none of those covers software aging and rejuvenation aspects.

Our work aims to evaluate the security impact caused by VM migration scheduling as support for VMM software rejuvenation. This way, this chapter intends to address the following research question:

RQ_{main} - *What is the security risk impact of applying different VM migration policies for VMM software rejuvenation purposes?*

On top of this, we consider two sub-questions:

- RQ_{s1} - *What is the VM migration policy that reduces the system security risk?*
- RQ_{s2} - *What are the trade-offs between availability and security when using VM migration scheduling as support for VMM rejuvenation?.*

To answer these questions, we propose an availability model based on SPN for systems with VM migration scheduling for VMM software rejuvenation purposes.

From the proposed availability model, we extract a security measure named RiskScore that, instead of assuming the attacker behavior (which is very difficult to characterize), is based on the time that the system spends on risky states (from a security perspective). Note that although availability is dependability and also a security attribute [Avizienis et al., 2001], both dependability events (e.g., failures, crashes, or hangs) and security events (e.g., malicious activities or security attacks) can impact it in this chapter, the availability evaluation is considered only from the dependability perspective. From the security perspective, we propose and evaluate the RiskScore.

We present three case studies to validate our proposal. The first considers the *Man-in-the-middle* security threat, the second considers *Denial of Service* attacks, and the last one is a combination of both threats. We provide a set of scenarios in each case study covering different VM migration scheduling alternatives to support the analysis of the tradeoff between availability and security risk.

To our knowledge, this is the first research effort on the analysis of the tradeoff between availability and security risk in virtualized systems with VM migration as VMM rejuvenation. We present a comprehensive set of results that provides some understanding of the availability and security risk tradeoffs raised by such a rejuvenation technique. Our security evaluation approach can be adapted for other threat models without requiring availability model modification. **This chapter is adapted from Torquato et al. [2019b]**.

The rest of this chapter is organized as follows. Section 5.1 presents our proposed security evaluation approach. Section 5.2 elaborates on the proposed model. Section 5.3 presents and discusses the results. Section 5.4 presents the limitations and threats to validity. Finally, Section 5.5 concludes the chapter.

5.1 Approach and Assumptions

As presented in Chapter 3, the system architecture considered has three main components: VM - a virtual machine running the desired application; *Main Node* - a physical machine that hosts the VMs; and *Standby Node* - a physical machine used to receive VM migrations. The selected architecture covers the main components of virtualized environments. Indeed, complex virtualized infrastructures in Cloud Computing also have similar architectural components [Machida et al., 2013; Torquato et al., 2018b].

The RiskScore metric reflects the probability of the system to be in a condition that enables (or improves the likelihood of) a successful specific attack (i.e., to measure the security risk associated with the time spent[1] in risky states) and does not assume characteristics as attack probability or rate. In other words, the assumption is that the attack's success is related to the time spent in a risky (or vulnerable) state. This way, the RiskScore metric captures the elapsed time in a condition (or state) that raises or enables a successful security attack. As we have different preconditions for different security attacks, Our approach allows

[1]The time spent in a state is also known as *mean sojourn time* or *mean waiting time*

the computation of the RiskScore metric for different threats using the same availability model.

To clarify the proposed approach, let us consider an illustrative example. Consider a Virtual Machine with migration capabilities. The Virtual Machine has three main states. The `UP` state means that the VM is running, the `DW` state represents that the VM is down, and `MG` represents that the VM is migrating. The state machine diagram is presented in Figure 5.1, which also includes the transitions between states. In this example, we neglect possible VM failures during migration.

Let us suppose that such a system is liable to suffer an attack related to data stealing during migration. So, in this case, the only state that raises a security concern is `MG`. Therefore, the RiskScore is a measure based on the probability of the system being in the `MG` state. With this, the system manager may adjust the α parameter (which is related to the frequency of migrations) to achieve the desired levels of RiskScore.

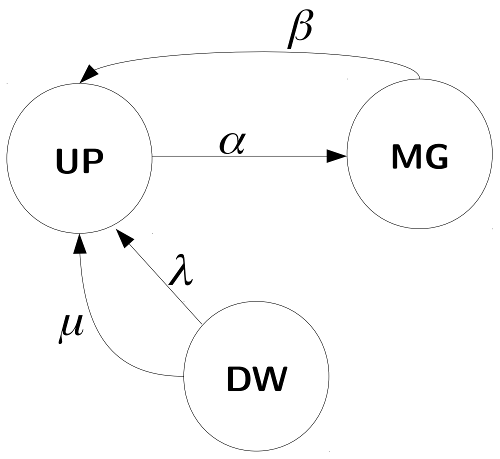


Figure 5.1: Illustrative example - State-machine diagram

This security evaluation approach has two main advantages: i) ***focus on the system state rather than on the attacker*** - we assume that the behavior of the attacker during the attack is unpredictable and, therefore, our security evaluation method computes the security levels only from the system state; and ii) ***security evaluation using unaltered availability models*** - the security evaluation approach does not require the modification of the availability models. In fact, as we obtain security levels from the system state, the evaluation is made from the steady-state probability of the system being in a specific state using the proper model reward measures.

5.2 Model

Figure 5.2 presents the proposed model. We adopt a monolithic model, but for this explanation, let us divide the two main areas of this monolithic model: a) the Clock Model composed by the places `Clock` and `Schedule`, and transitions

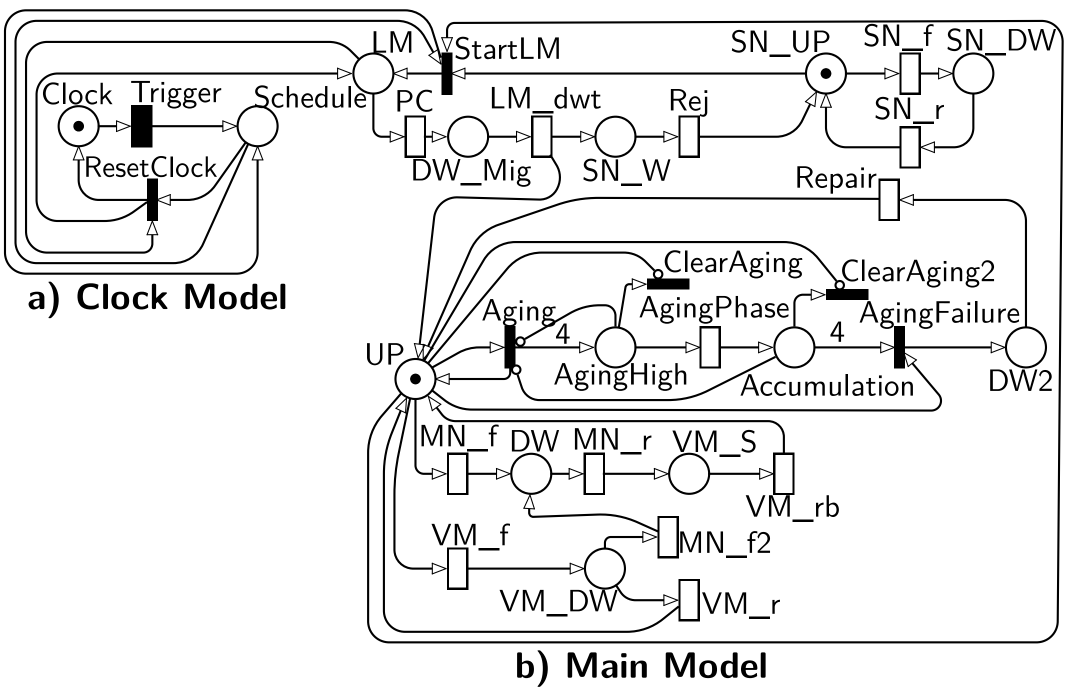


Figure 5.2: SPN model

Trigger and ResetClock; and b) the Main Model composed by the other places and transitions.

The Clock Model represents the behavior of the rejuvenation scheduling on the environment. The Clock can be any type of component capable of counting time and communicating with the system. The token in the Clock place represents that the time counting to VM Migration submission is active. The deterministic transition Trigger represents the time interval between submitting VM Migration requests to the environment. When the deterministic time is reached, the Trigger transition fires, putting a token in the Schedule place. The Schedule place with tokens represents that the VM Migration is about to start. However, the VM Migration start (StartLM transition firing) depends on two more conditions: i) *Main Node* and *VM* running (a token in the UP place), and ii) *Standby Node* running (a token in SN_UP place). Once those conditions are satisfied, the VM Migration starts (StartLM firing). StartLM firing removes the tokens from the places UP and SN_UP and puts a token in place LM. StartLM also swaps[2] the token in the place Schedule. We use this strategy to avoid including guard functions in the model, therefore improving its readability. We highlight that the token swap does not affect token aging because the StartLM transition is an immediate transition. When the system starts the migration, the Clock component has to start the time counting for the next VM Migration (ResetClock firing). ResetClock firing swaps the token from the LM place and moves the token from the Schedule place to the Clock place, thus restarting the VM migration interval time counting.

Regarding the VM Migration process, when the LM place has tokens, the transition PC becomes enabled. This transition is related to the time of the *copy-phase* of the

[2]receives and gives back

VM migration algorithm. We parameterized this transition based on [Clark et al., 2005] with the mean time of 72 seconds. The transition `LM_dwt` represents the *downtime* of the VM migration. As the time of VM migration may vary depending on the workload, we used exponential transitions instead of deterministic transitions. When the VM migration finishes (`LM_dwt firing`), the system returns to operation (a token is deposited in the `UP` place), and the previous *Main Node* will pass through software rejuvenation (a token is deposited in the `SN_W` place). VM migration will be enabled again after the software rejuvenation action. This behavior is represented by `Rej` transition firing, moving the token from the `SN_W` place to the `SN_UP` place.

We assume that the *Main Node* and the *VM* are operational at the start of the model analysis. This assumption is represented by the token in `UP` place. At this point, if the *VM* fails (represented by `VM_f` transition), the system goes out-of-service (the token is removed from the `UP` place and deposited in the `VM_DW` place). If the *VM* is correctly repaired, then the system returns to activity. The delay time for *VM* repair is represented by the `VM_r` transition. However, the *VM* failure may be followed by a *Main Node* failure. This behavior is represented when the `MN_f2` transition fires and puts a token in the `DW` place. A *Main Node* failure can also occur before a *VM* failure (`MN_f` transition firing). The system recovery after a *Main Node* is made in two steps: first, the *Main Node* repair (represented by `MN_r` transition firing), after the *Main Node* repair the VM is stopped (a token is deposited in the place `VM_S`. The second step of the system recovery is the *VM* reboot (`VM_rb` transition firing).

A 4-phase Erlang sub-net represents the aging accumulation process. This type of subnet can represent the Increasing Failure Rate (IFR) of a software aging accumulation process [Machida et al., 2013]. We highlight that other distributions can be applied in the software aging modeling process [Levitin et al., 2018]. However, the inclusion of such distributions may require a complete model redesign and is out of the scope of this chapter. We selected Erlang sub-nets in the software aging modeling because it is widely adopted for this purpose [Machida et al., 2010; Melo et al., 2013b]. The `AgingPhase` transition represents the phases of the Erlang sub-net. A failure caused by software aging occurs when the accumulation status reaches critical levels. In the model, this behavior is represented when the `AgingFailure` transition fires, removing the token in the `UP` place and putting it in the `DW2` place. The `Repair` transition represents the recovering process after a software aging failure. The `ClearAging` and `ClearAging2` transitions represent events that clear the software aging accumulation process. We consider that the *Main Node* and the *VM* repairs comprise rejuvenation actions. Therefore, the software aging accumulation cleanup occurs when the *Main Node* or the *VM* fail. The other case of software rejuvenation occurs on VM Live Migration. After VM Live Migration, the *VM* arrives on a fresh state VMM. Therefore, the software aging status is removed. We used inhibitor arcs[3] from the `UP` place to the `ClearAging` and `ClearAging2` transitions to represent the behaviors related to software rejuvenation.

[3]arc terminating in a circle instead of an arrow

Table 5.1: Immediate transitions and their meaning

Transition	**Meaning**	**Input arcs**	**Output arcs**
`ResetClock`	Restart time counting for VM migration	`Schedule` (1) `LM` (1)	`Clock` (1) `LM` (1)
`StartLM`	Start of VM migration process	`SN_UP` (1) `UP` (1) `Schedule` (1)	`Schedule` (1) `LM` (1)
`Aging`	Start of software aging accumulation	`UP` (1) `AgingHigh` (0) `Accumulation` (0)	`UP` (1) `AgingHigh` (4)
`AgingFailure`	System failure occurrence due to software aging	`UP` (1) `Accumulation` (4)	`DW2` (1)
`ClearAging1`	Software aging accumulation clearance	`AgingHigh` (1) `UP` (0)	Transition without output arcs
`ClearAging2`	Software aging accumulation clearance	`Accumulation` (1) `UP` (0)	Transition without output arcs

The model also covers *Standby Node* failures, which affect system availability indirectly as it prevents VM migration, making a software aging failure more likely to occur. The transitions `SN_f` and `SN_r` represent the *Standby Node* failure and repair events, respectively. The places `SN_UP` and `SN_DW` are related to the *Standby Node* availability and unavailability, respectively.

Table 5.1 summarizes the immediate transitions, their meaning, and their input and output arcs with respective weights[4]. For the sake of readability, we present the list of the places used in our model. Table 5.2 presents the information about the places. The column **meaning with tokens** represents the system state when the considered place has tokens.

Our availability model comprises software aging and rejuvenation behavior, non-aging failures, and details of VM migration (e.g., pre-copy and downtime phases). The inclusion of complex system behavior interactions may lead the model to a state explosion problem. To avoid this type of problem, we need to apply simplifications in what aspects should be considered in the models. For example, differently from Chapter 4, the model here (i.e., Figure 5.2) neglects the occurrence of bursty workloads. Besides that, the performance evaluation is out of the scope of this chapter.

[4]Associated number of tokens. We use zero (0) for inhibitor arcs.

Table 5.2: Places description

Place	Meaning with tokens
`Clock`	Timer counting for the next migration
`Schedule`	Timer reaches the VM migration interval time
`LM`	VM migration is started
`DW_Mig`	VM migration is on the downtime phase
`SN_W`	Standby Node is waiting for the rejuvenation action
`SN_UP`	Standby Node is up
`SN_DW`	Standby Node is down
`UP`	Main Node and VM are running
`AgingHigh`	VMM starts to accumulate software aging
`Accumulation`	The number of tokens in this place represents the VMM software aging accumulation status
`DW2`	The system is down due to a software aging failure
`DW`	The system is down due to a Main Node non-aging failure
`VM_DW`	The system is down due to a VM non-aging failure
`VM_S`	VM is waiting for a reboot after a Main Node repair

5.3 Case Studies

The numerical evaluations of the models are obtained using the TimeNET tool [Zimmermann, 2017]. Table 5.3 contains the parameters used in our evaluation. These parameters are based on previous experimental studies and other consolidated works [Kim et al., 2009; Torquato et al., 2018b; Clark et al., 2005].

For the three case studies, we compute the system unavailability using the following: $Unavailability = 1 - Availability$. $Availability$ is obtained through $Availability = P(\#UP > 0)OR(\#LM > 0)$, where we are computing the probability of token presence in the places `UP` or `LM`. In other words, we obtain the system availability, observing the probability of the system being running or in the *copy-phase* of VM migration. Unavailability is the probability that the system is out of such a state.

The case studies are: Man-in-the-middle (Section 5.3.1), Denial of Service (Section 5.3.2), and the combination of both (Section 5.3.3).

5.3.1 CS #1 - Man-in-the-middle Attack

In Man-in-the-middle (MITM) attacks, the attacker has access to the data link between two communication endpoints [CAPEC, 2023]. With such access, the attacker can deploy attacks to change the network traffic or to eavesdrop on the communication [Conti et al., 2016]. Even with proper VM migration data traffic encryption, an attacker may recognize the migrating VMs in the network.

We consider an attacker who has the necessary skills to hijack the VM migration route and to perform a malicious action (e.g., secretly copying the data in traffic, changing or destroying the data in the VM migration packets). As

Table 5.3: Parameters used in the timed transitions

Parameters		**Values**
Transition Name	***Description***	***Mean time***
`MN_f, MN_f2`	*Main Node* Failure Delay	1236.706 h
`MN_r`	*Main Node* Repair Delay	1.094 h
`SN_f`	*Standby Node* Failure Delay	1236.706 h
`SN_r`	*Standby Node* Repair Delay	1.094 h
`VM_f`	Virtual Machine Failure Delay	2880 h
`VM_r`	Virtual Machine Repair Delay	30 min
`VM_rb`	Virtual Machine Reboot Delay	5 min
`PC`	VM Live Migration pre-copy phase time	72 s
`LM_dwt`	VM Live Migration Downtime	4 s
`Rej`	Rejuvenation Node Delay	2 min
`SARec`	Software Aging Recovery Delay	1 h
`AgingPhase`	Time to Aging (Phases)	62.5 h
`Trigger`	Interval to VM Live Migration	1 $\rightarrow$ 168 h

we are using VM migration scheduling for VMM rejuvenation purposes, frequent migrations may raise concerns regarding this type of attack. We compute the RiskScore of this case study with the following expression: $RiskScore = MigrationProbability$. The $MigrationProbability$ is the probability of the system on a VM migration. We obtain $MigrationProbability$ by observing the probability of token presence in places `LM` or `DW_mig`. We use the following metric: $MigrationProbability = P(\#LM > 0)\ OR\ (\#DW\_Mig > 0)$, where $P\{\}$ refers to probability and $\#LM$ and $\#DW\_Mig$ refer to the number of tokens in the `LM` and `DW_Mig` places, respectively.

Figure 5.3 presents the results obtained. The Y-axis represents the system Unavailability, and the X-axis represents the rejuvenation trigger considered (in hours). We add a secondary Y-axis on the right side with the values of the RiskScore. The black dotted line represents unavailability, and the gray dotted line is the RiskScore. We vary the rejuvenation trigger from one hour to 168 hours (a week) with a half-hour step. We highlight that the RiskScore in this case study only considers the probability that the system is performing a VM migration.

Figure 5.3 shows a valley for the system unavailability values. This point represents the rejuvenation trigger that maximizes system availability. It shows the balance between too frequent rejuvenation that may impair availability due to each migration downtime and less frequent migrations that raise the probability of software aging failures. Besides that, it is noticeable that, as expected, less frequent migrations reduce the security risk associated with Man-in-the-middle (MITM) attacks.

Unavailability and RiskScore are non-beneficial metrics (i.e., the lower the better). The plot in Figure 5.4 presents the normalized values for Unavailability versus the normalized values for RiskScore. We decided to normalize the

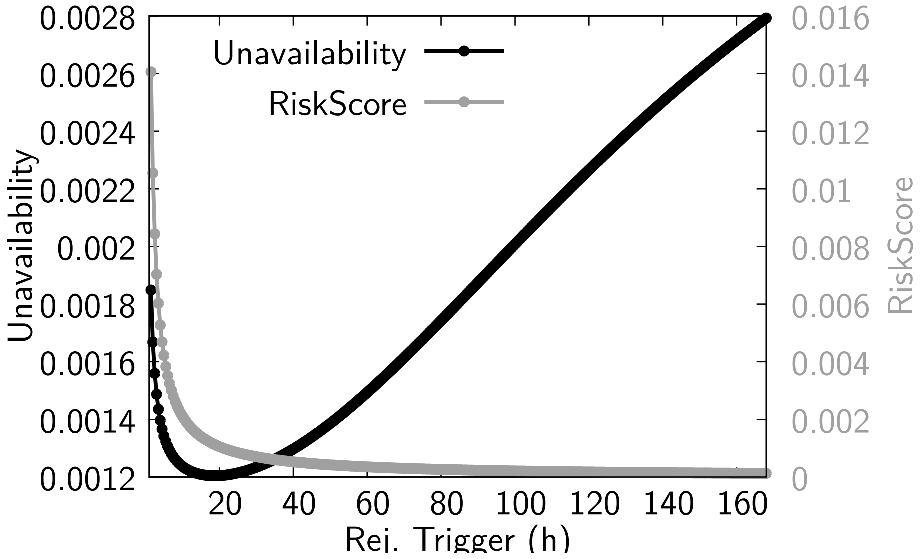


Figure 5.3: Case study 1 results - Man-in-the-middle

data to reduce possible bias due to the difference in magnitude of the metrics. We apply a simple normalization, as follows: $Norm(UA_{(n)}) = UA_{(n)}/Max(UA)$, where $Norm(UA_{(n)})$ is the normalized value of UNAVAILABILITY when applying rejuvenation interval of n hours, $UA_{(n)}$ is the actual UNAVAILABILITY value when using rejuvenation interval of n hours, and, $Max(UA)$ is the maximum observed value of UNAVAILABILITY in the considered rejuvenation interval range of values (i.e. from one hour to 168 hours, using a half-hour step).

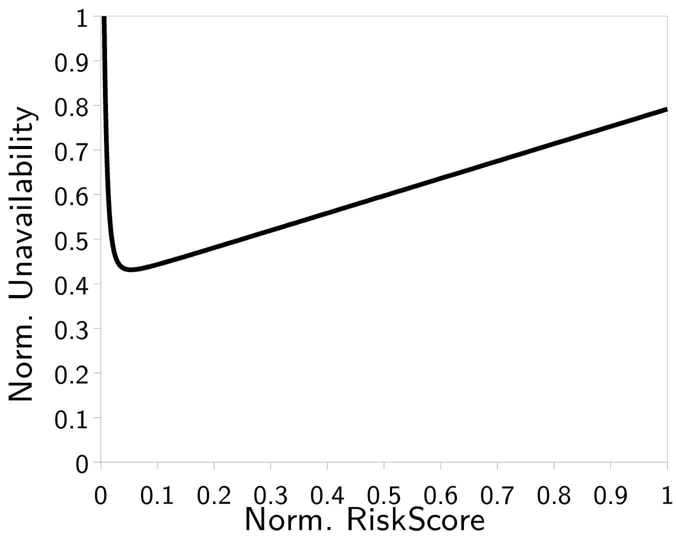


Figure 5.4: Case study 1 results - Man-in-the-middle - Normalized plot

In Figure 5.4, the point in the curve with the minimum distance to the origin (0,0) represents the configuration that globally reduces the combination of UNAVAILABILITY and RISKSCORE. We computed the Euclidean distance of all the points of the curve to the origin point. The point with the shortest distance corresponds to the rejuvenation policy with a VM migration interval of 20 hours. The detailed results are presented in Table 5.4 (Scenario #0).

Our analysis also aims at finding the rejuvenation policies that minimize the values of UNAVAILABILITY or RISKSCORE or even a composition of both. By composition, we mean the desired proportion of each metric. For example, a system manager may want to deploy a specific VM migration policy that considers both metrics (UNAVAILABILITY and RISKSCORE) equally (50% UNAVAILABILITY and 50% RISKSCORE), or some other values. We computed some intermediate scenarios using proportions for both metrics. To avoid decision bias, we used the normalized data mentioned earlier. The results are shown in Table 5.4, which includes the following: **Scn #** - Proposed scenario; **Criteria** - considered proportion for each metric in the composition; **Rej. Trigger** - specific VM migration policy observing the desired criteria; **Unavailability (UA)** - steady-state unavailability; **Downtime (h/yr)** - estimated downtime in hours per year; **RiskScore** - steady-state risk score; and **Risk classification** - a comparison between the obtained RISKSCORE and the maximum RISKSCORE observed in the analysis[5] (MAXRS). The **Risk classification** result provides a proportion of the increase/reduction impact due to the selected VM migration policy when compared to the scenario with the worse security levels. Thus, **Risk classification** results are useful to understand the actual levels of security improvement when applying the VM migration policies.

Table 5.4: Evaluation scenarios of Case study 1 - Man-in-the-middle

Scn#	**Criteria**	**Rej. Trig.**	**Unavail.**	**Down-time (h/yr)**	**RiskScore**	**Risk classification**
0	*Optimal policy*	20 h	0.00120484	10.5544	0.00105544	5% of MAXRS
1	100% UNAVAIL.	19 h	0.00120442	10.55072	0.00111098	5.27% of MAXRS
2	75% UNAVAIL. AND 25% RISKSCORE	22.5 h	0.00120813	10.58322	0.00093818	4.45% of MAXRS
3	50% UNAVAIL. AND 50% RISKSCORE	27 h	0.00122098	10.69578	0.00078183	3.71% of MAXRS
4	25% UNAVAIL. AND 75% RISKSCORE	35.5 h	0.00126506	11.08193	0.00059464	2.82% of MAXRS
5	100% RISKSCORE	168 h	0.00279284	24.46528	0.00012566	0.59% of MAXRS

The results presented for this case study (above) and for the following ones (next sections) will provide the necessary inputs for answering RQ_{s1} and RQ_{s2}. The answer to RQ_{s1} is on the tables with the case study results in the scenario that minimizes RISKSCORE metric (100% RISK row on each of the results table - Scn#5 row of Tables 5.4 and 5.5). The other plots and tables in each case study provide the information for answering RQ_{s2}.

[5]Variation of Rej. Trigger from one hour to 168 hours with a half-hour step.

From the results above, we can observe that the rejuvenation policy decision depends on the system manager criteria about UNAVAILABILITY and RISKSCORE. We provide a comprehensive set of results to support such a decision. We highlight the following conclusions from this case study: i) a VM migration interval of 20 hours provides the best overall rejuvenation policy considering both metrics equally; ii) when putting more weight on UNAVAILABILITY in the rejuvenation policy decision, the VM migration interval tends to be shorter than in scenarios with more weight on RISKSCORE; and iii) longer VM migration intervals provide the best RISKSCORE result. We recall that the maximum limit of our analysis is 168 hours. As Man-in-the-middle attacks depend on VM migration, less frequent migrations will reduce the value of RISKSCORE.

5.3.2 CS #2 - Denial of Service (DoS) Attack

Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks are the main threats to Cloud Computing availability [Subashini and Kavitha, 2011]. DoS attacks usually aim to flood the network resources or impair the application running on the VMs and are highly devastating in Cloud environments due to their flexibility. In fact, when receiving a high workload, the Cloud environment starts to allocate more resources (i.e., servers and VMs) to handle the incoming requests; therefore, by flooding just one of the servers, the DoS attack may end up affecting the overall Cloud availability.

The considered DoS attack consists of a high workload demand for an application running upon a VM. So, the RISKSCORE metric for this case study has to cover both aspects: network and application state. The reasoning is that the VM migration process affects the network state, and software aging affects the application state. Besides that, software aging accumulation forces the system to use more memory, which affects the VM migration network overhead as there are more dirty memory pages.

We compose the RISKSCORE for this case study as follows. $RiskScore = (w1 \times RiskMigration) + (w2 \times RiskAging)$. $w1$ and $w2$ are the assigned weights for VM migration security risk and software aging security risk, respectively. As VM migration produces high bursty network traffic, we assign more weight to *RiskMigration*. In practice, the weight configuration is $w1 = 0.6$ and $w2 = 0.4$. *RiskAging* is the expected software aging accumulation. We compute the *RiskAging* from the expected number of tokens in the place `Accumulation` (see model in Fig. 5.2). The *RiskMigration* is obtained using the equation: $RiskMigration = MigProbability \times RiskAging$, where *MigProbability* is the probability of the system performing a VM migration (as in the previous case study). We decided to add the *RiskAging* in the *RiskMigration* to capture the following behavior: migration of a VM with more accumulation of software aging will produce a higher impact in the network than a VM with less software aging accumulation.

Figure 5.5 presents the results obtained. As in the previous case study, the black lines are related to the system UNAVAILABILITY, and the gray lines are related to the RISKSCORE metric. We also added a secondary y-axis with the scale for the RiskScore values. The plot also contains two continuous lines for the baseline (i.e., without software rejuvenation) architecture. The variation of the VM migration scheduling obeys the one presented earlier, from one hour to a week (168 hours) with half-hour steps.

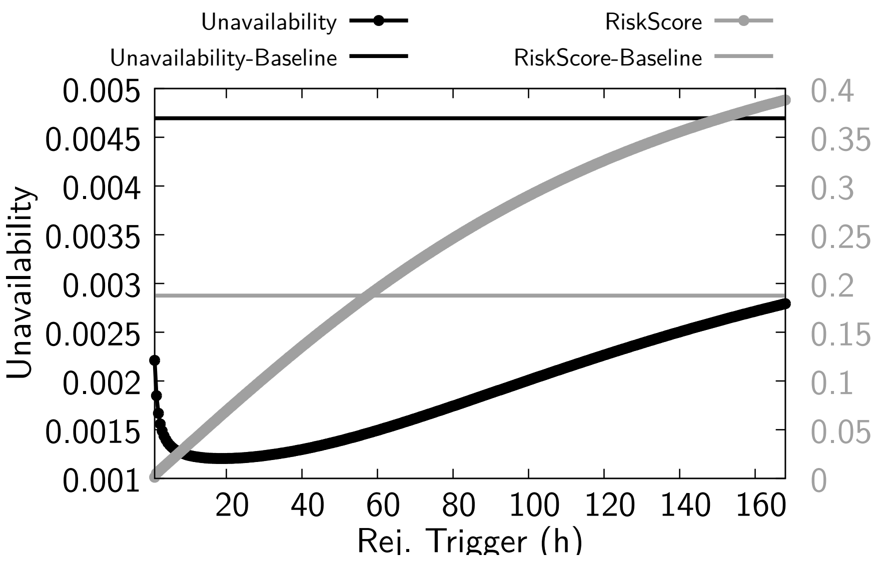


Figure 5.5: Case study 2 results - Denial of Service

The gray lines interception represents the point where the VM migration scheduling is no longer beneficial for the RiskScore when compared to the baseline. The exact intercept is when the rejuvenation interval is equal to 57 hours, which means that larger rejuvenation intervals produce worse security results than the system without VM migration scheduling. Figure 5.6 presents the normalized results for both metrics, Unavailability and RiskScore. We computed the same set of scenarios of the previous case study. The results are presented in Table 5.5 (see the previous section for the meaning of the columns in the table).

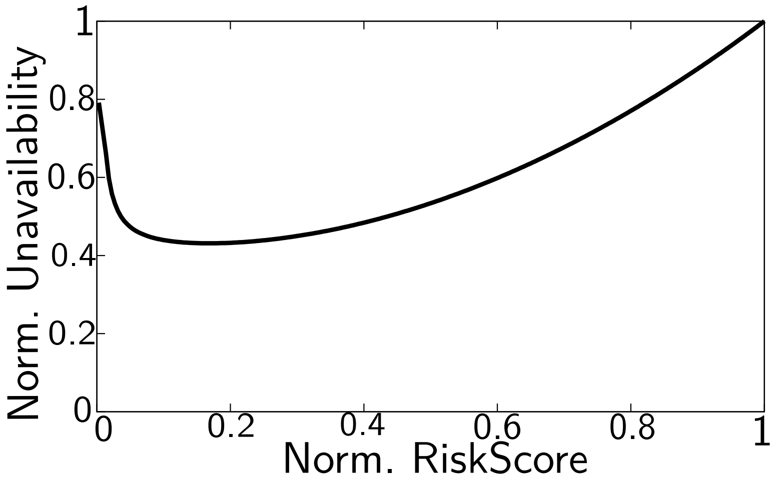


Figure 5.6: Case study 2 results - Denial of Service - Normalized plot

We highlight three key conclusions from the results. First, the best overall rejuvenation policy, which globally minimizes both Unavailability and RiskScore,

Table 5.5: Evaluation scenarios of Case study 2 - Denial of Service

Scn #	**Criteria**	**Rej. Trig.**	**Unavail-ability**	**Down-time (h/yr)**	**RiskScore**	**Risk classi-fication**
0	*Optimal policy*	12 h	0.00122172	10.7023	0.041906	10.79% of MaxRS
1	100% Unavail.	19 h	0.00120442	10.5507	0.06600	17% of MaxRS
2	75% Unavail. AND 25% RiskScore	10.5 h	0.00123266	10.7981	0.03670	9.45% of MaxRS
3	50% Unavail. AND 50% RiskScore	6.5 h	0.00129305	11.3271	0.02276	5.86% of MaxRS
4	25% Unavail. AND 75% RiskScore	4 h	0.00139666	12.2347	0.01402	3.61% of MaxRS
5	100% RiskScore	1 hr	0.00221065	19.3652	0.00116	0.3% of MaxRS

has a VM migration interval of 12 hours. Second, RiskScore raises with the increase of the VM migration interval. As the time that a VM spends migrating is substantially lower than the time that a VM spends running, the software aging accumulation is the most relevant aspect for the RiskScore metric in this scenario. More frequent migrations may produce network overhead but also avoid severe software aging accumulation. Therefore, more frequent migrations result in less software aging accumulation, and thus a lower RiskScore. The last conclusion is that the criteria focused on minimizing RiskScore produces more reduction when compared to the previous case study.

5.3.3 CS #3 - Composition of Attacks

In the previous case studies, we presented the analysis for the two threats separately (DoS and MITM). However, in some contexts, we may need to deal with both threats simultaneously. Therefore, we present a study case that combines the three metrics of interest: Unavailability, RiskScore of MITM (RS-MITM) and RiskScore of DoS (RS-DoS). Figure 5.7 presents a plot with the results considering different rejuvenation policies. The plot basically merges the plots from Figures 5.3 and 5.5.

Figure 5.7 shows that the RS-MITM value is smaller than the RS-DoS value. RS-MITM considers only VM migration as a security risk. Therefore, its absolute values are related to the steady-state probability that the system is performing a VM migration. From a stationary perspective, the time that the system spends on a migration state is lower than the time that it spends on a normal operation (i.e., running and accumulating software aging effects).

It is, however, hard to set up a direct comparison between both RiskScore metrics due to the different nature of the threats. Just comparing the absolute values of the two metrics may not be enough for a tradeoff analysis. For example, the consequences of an MITM attack may be more critical than the consequences of a DoS attack (depending on the business context; e.g., MITM attacks can

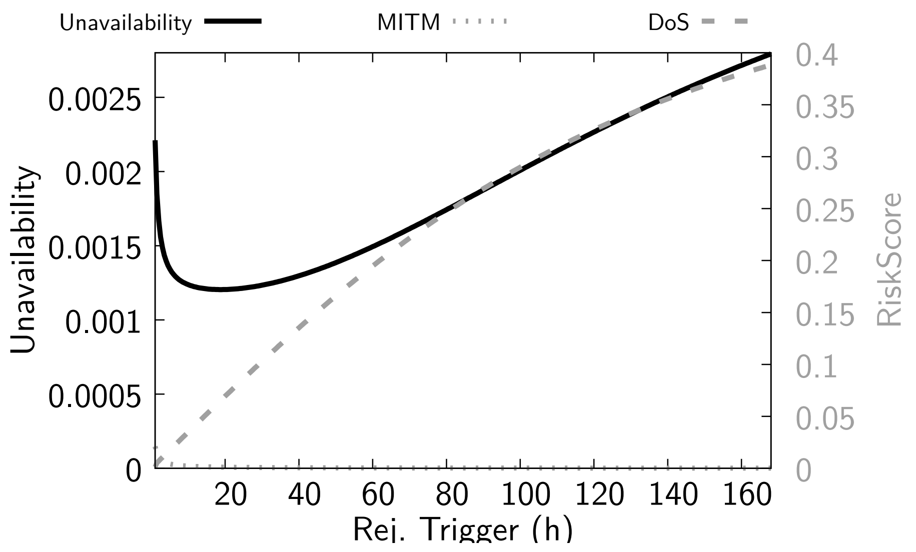


Figure 5.7: Case Study 3 results - Composition

try to affect system confidentiality, which can be devastating for some types of organizations). If we compare only the absolute values, we tend to neglect the influence of each threat on the overall system security. This way, we normalized the data to provide a more fair trade-off analysis. The normalization allows us to deal with a maximum and minimum range of our data. The normalization approach is the one used in the previous case studies. Figure 5.8 presents a 3D plot with the results for each normalized metric. We added shades (in gray) for each point of the 3D plot in the xy and yz planes.

Due to the different nature of the considered threats (MITM and DoS), we decided not to attempt proposing a general (i.e., capable of representing both threats at once) formula for *RiskScore* calculation for this particular case study. The results of such a possible general formula may hide how much RiskScore is due to the MITM or DoS threats.

We can use the data from Figure 5.8 to find the optimal policy when considering Unavailability, RS-MITM, and RS-DoS. We obtain the best overall policy by finding the point with the shortest distance from the origin (point (0,0,0)). We computed such a distance using the Euclidean Distance. We also present some alternative scenarios with different weight configurations for the three metrics. Table 5.6 presents the results. The columns **% UA** and **% MITM** and **% DoS** show the weight for each metric. The columns **MR-MITM** and **MR-DoS** represent the RiskScore reduction when compared to the maximum observed RiskScore of MITM and DoS, respectively. As in the previous case studies, we assume that the VM migration interval ranges between 1 and 168 hours. The remaining columns have the same meaning as presented in the previous cases.

We highlight the following from the results: i) the policy that globally reduces the three metrics has the VM migration interval of 13.5 hours; ii) results from scenarios 5 and 7 reveal that migration policies for RS-MITM and RS-DoS are incompatible, thus trying to reduce one metric will increase the other (the

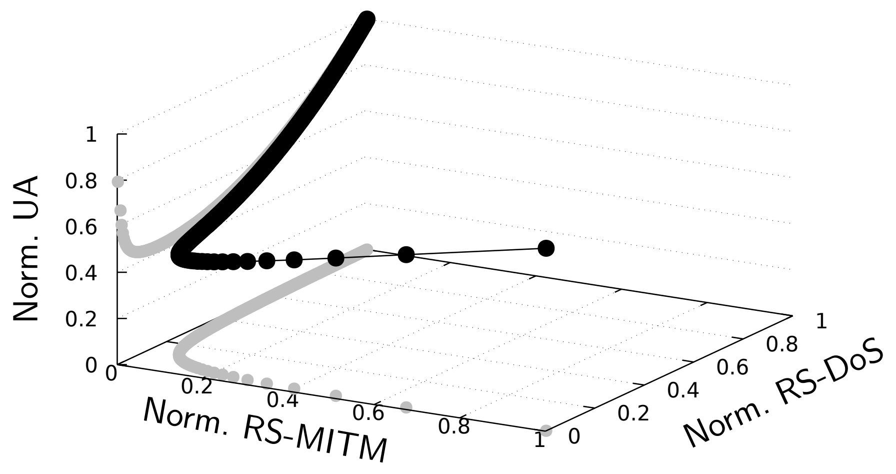


Figure 5.8: Case study 3 results - Composition - Normalized plot

results presented offer relevant information regarding finding the best balance); and iii) an interesting approach for decision making in such complex scenarios is the definition of unacceptable service levels instead of desired service levels, which allows us to remove the worse solutions from the decision-making process.

Table 5.6: Evaluation scenarios of Case study 3 - Composition

Scn #	**% UA**	**% MITM**	**% DoS**	**Rej. Trigger**	**UA**	**Down-time (h/yr)**	**RS-MITM**	**% MR-MITM**	**RS-DoS**	**% MR-DoS**
0	*Optimal policy*			13.5 h	0.00121414	10.64	0.00156355	7.41	0.04709	12.13
1	100%	0%	0%	19 h	0.00120442	10.55	0.00111098	5.27	0.06600	17.00
2	50%	25%	25%	13 h	0.00121635	10.66	0.00162367	7.70	0.04537	11.68
3	33.33%	33.33%	33.33%	12 h	0.00122172	10.70	0.00175895	8.34	0.04191	10.79
4	25%	50%	25%	15.5 h	0.00120785	10.58	0.00136182	6.46	0.05400	13.90
5	0%	100%	0%	168 h	0.00279284	24.47	0.00012566	0.59	0.38828	100
6	25%	25%	50%	8.5 h	0.00125502	10.99	0.00248305	11.77	0.02974	7.66
7	0%	0%	100%	1 h	0.00221065	19.36529	0.02108543	100	0.001165	0.30

5.4 Threats to Validity and Limitations

Lack of experimental validation

The proposed models lack validation against experimental results. Unfortunately, up to this date, we are unable to compare model results and experimental results. The complete validation against experimental results became even more difficult because we were observing steady-state metrics. Obtaining steady-state metrics from experimental results requires a long observation time to collect a representative sample of results. In this research, we decided to focus on model interactions between dependability and security and to provide a set of interesting evaluation scenarios. Previous consolidated works such as Kim et al. [2009] and Machida et al. [2013] follow the same approach.

Parameters adopted from previous works instead of from experimental results

In the current research, the model input parameters came mainly from previously published works. In an ideal setup, the model parameter values are the result of system observation and experimentation. The main problem here is that some of these parameters are unavailable or hard to obtain. Collecting MTTF of hardware and software components requires a long experimental campaign, as these failures may take months to occur [Schroeder and Gibson, 2007]. The application of experiment acceleration (e.g., fault injection [Hsueh et al., 1997]) may be a way to reach this goal. Actually, we took a similar path in a previous paper [Torquato et al., 2019c]. Nevertheless, for a complex model, such as the one presented in this chapter, the use of such an approach is prohibitive as we take several components into account. As a mitigation, in this chapter, we conducted a sensitivity analysis of rejuvenation trigger parameter. Hopefully, the proposed range of analysis covers a set of representative scenarios.

Performance overhead evaluation

Quality of Service (QoS) of the applications running on a VM is likely to drop during the migration process. The memory pages copying and transferring through the network require extra computation. Therefore, clients may notice performance oscillations in scenarios with frequent migrations. The proposed model neglects the evaluation of performance overhead due to VM migration. An approach to overcome this limitation is through the use of hierarchical composition (as presented in Chapter 4). As a mitigation, it is possible to adjust the delay of transition `PC` to represent longer VM migrations (i.e., migrations of VMs running heavier workloads).

5.5 Summary

This chapter presented a comprehensive evaluation of the availability and security of virtualized systems with VMM rejuvenation enabled by VM migration scheduling. Our results provide a trade-off analysis in three case studies: i) Man-in-the-middle (MITM), ii) Denial of Service (DoS), and iii) a combination of MITM and DoS attacks. Each case study provides a set of scenarios comprising the optimal rejuvenation scheduling and intermediary configurations of weights for security and availability, providing more information for the decision-making process.

Our security evaluation approach is flexible and can consider other threat models and be used in other modeling frameworks (e.g., performance models and Markov Chains). The final results show that reducing the risk associated with the Man-in-the-middle attack is incompatible with reducing the risk related to Denial of Service, as policies reducing one tend to increase the other.

Regarding our main research question, RQ_{main} - *What is the security impact caused by applying different VM migration policies for VMM software rejuvenation purposes?*, we perceived that the security impact depends on the security threats considered. Specifically about our case studies, we highlight: i) when considering only MITM attacks, VM migration frequency determines the system RiskScore levels; ii) when considering only DoS attacks, the time spent on a state with software aging accumulation is the most critical factor for system RiskScore; and iii) when considering both threats, rejuvenation policies that reduce DoS RiskScore are preferred because the system tends to stay more time in a state with software aging accumulation than "in migration".

This chapter closes our contributions focused only on software rejuvenation. In the following chapters, we shift the focus of the VM migration capabilities for the MTD purposes. Based on the knowledge obtained, we consider two scenarios for the following evaluations: **Persistent** and **Non-Persistent** tactics (these scenarios are inspired in the definitions of the *Persistence* tactic proposed in the MITRE ATT&CK matrix [MITRE, 2023]). We decided to consider both scenarios as they are representative of the cybersecurity landscape. In the following chapters, we detail the problem and the proposed models for each scenario (**Persistent** in Chapter 6 and **Non-Persistent** in Chapter 7).

Chapter 6

Time-based VM Migration as MTD against Persistent Attacks

VM migration is an usual strategy for MTD in the cloud [Torquato and Vieira, 2020]. The standard approach is to move VMs to prevent them from attacking co-resident VMs or the underlying physical host [Wang et al., 2016; Penner and Guirguis, 2017]. Specifically, we consider an attacker adopting a persistent tactic to compromise the hypervisor of the host machine. Therefore, to defend against this threat, we propose a heterogeneous-hypervisor VM migration [Liu et al., 2008].

Wang et al. [2016] proposed an algorithm for selecting proper MTD timing to minimize the associated costs. Their work provided insights on how to evaluate different MTD timing approaches. Penner and Guirguis [2017] proposed a set of MTD mechanisms against Multi-Armed Bandit (MAB) policy attacks. Our work considers a similar threat model, but instead of MAB policies, we consider a multi-stage attack (i.e., the attacker may first reconnoiter the environment before launching the attack). Connell et al. [2018] presented a Markov model for performance evaluation of an MTD deployment in a virtualized environment. Their work provided insights on i) how to model the MTD problem and ii) validation through simulation. Unlike these works, we decided to follow a straightforward approach of applying the usual VM migration scheduling as MTD. By *straight-forward* we mean without requiring specific MTD implementations of third-party software.

This chapter presents a PN-based model for the evaluation of the probability of *insider* attack success in an Infrastructure as a Service (IaaS) cloud with MTD based on VM migration scheduling. The main goal is to evaluate the availability and security of the MTD deployment in different scenarios. These scenarios comprise different system architectures (i.e., environments with different number of available physical machine pools) and different VM migration scheduling policies. The analysis shows the benefits and drawbacks of the scheduling policies regarding availability and probability of attack success. Furthermore, we propose a *tolerance level* metric that, based on the probability of attack success, supports the selection of VMs as *candidates* for MTD deployment, considering their expected runtime. In practice, the *tolerance level* metric measures how long it takes for the VMs to reach a selected probability of attack success. Section 6.1 presents more details about *tolerance level* metric.

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

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

To address these questions, we consider three case studies. The first focuses on reducing the attack success probability when enlarging the system architecture from one physical machine pool to four physical machine pools. The second case study investigates the reduction of the attack success probability due to the variation of VM migration scheduling. In the last case study, we follow a slightly different approach. Instead of presenting results about specific scenarios, we conduct a comparison between the results of the model and the results from a simulation of the system behavior. Therefore, the last case study serves as a partial validation of the results as the multiple evaluation approaches (i.e., simulation and modeling) produce similar results.

At the time of this chapter writing, this was the first effort into the investigation of how different *environmental configurations* affect the security and availability levels of an environment with VM migration scheduling as MTD against *insider* attacks. VM migration scheduling is already used in other contexts, such as software rejuvenation [Bai et al., 2020], load balancing [Hu et al., 2010], and sustainability [Ghribi et al., 2013]. Our contribution may be of interest to system managers who intend to design multi-objective VM migration scheduling policies. **This chapter is adapted from Torquato et al. [2021a]**.

The rest of this chapter is organized as follows. Section 6.1 presents the evaluation approach, including the system architecture, the threat model, and the specific VM migration-based MTD strategy. Section 6.2 elaborates on the proposed model. Section 6.3 presents the proposed case studies. Section 6.4 presents the limitations and threats for validity. Finally, Section 6.5 concludes the chapter.

6.1 Approach and Assumptions

Section 3.2 already provided the general **architecture** for VM migration-based MTD. Nevertheless, as the approach for persistent and non-persistent tactics differ a little, we decided to include the full details here. In practice, we consider a set of architectures, ranging from *1N* (with one physical machine pool - ***baseline***) up to *4N*[1] (with four different physical machine pools). Figure 6.1 displays the system configuration for a *2N* architecture. The considered architecture has the following characteristics:

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

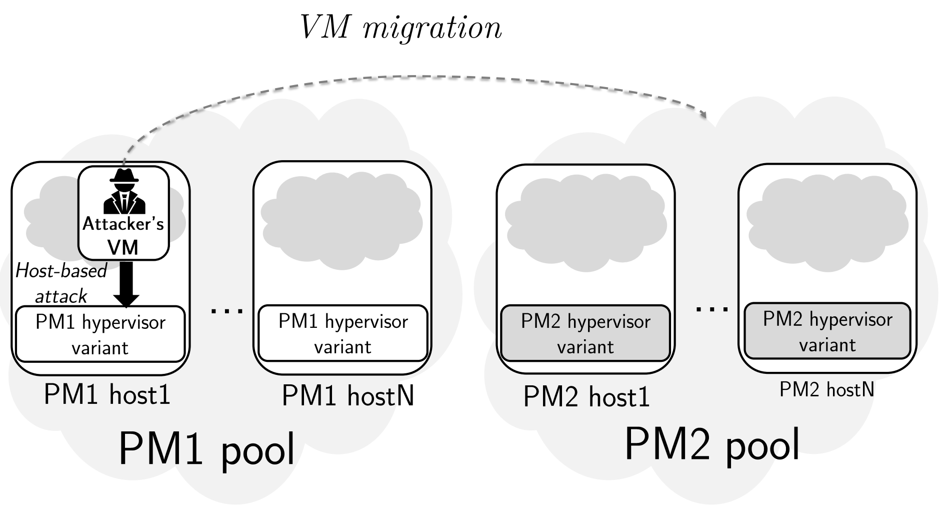


Figure 6.1: System configuration - *2N* architecture

The *attacker* has authorized control of one VM in the environment. In this **threat model**, the goal of the attacker is to perform an *insider* attack targeting the underlying hypervisor[2], which is the middleware between the VMs and the physical host. When assuming the control of the hypervisor, the *attacker* can monitor or compromise co-resident VMs. We assume that the *attacker* will not resign until the attack succeeds. The *insider* attack has two phases: i) *reconnaissance* - when the *attacker* tries to identify the hypervisor variant of the host; and ii) *attack* - when the *attacker* starts to perform malicious actions against the host hypervisor. In the *attack* phase, the *attacker* adopts a try-and-error approach. Consequently, the longer the time spent on the same physical host, the greater the chance of attack success. Besides, once the *attacker* reconnoiters the host hypervisor, it is possible to continue the try-and-error approach, ignoring the failed attempts (i.e., the *attacker* accumulates knowledge).

The **adopted MTD** consists of a time-based VM migration policy that moves the *attacker* VM between the available physical machine pools[3]. The MTD goal is

[1]We decided to scale the model up to *4N* because we found that there are four major hypervisors (i.e., the target of the attack) used in server environments - Xen, Hyper-V, ESXi, KVM.

[2] Note that some research works may refer to this attack as VM escape [Zhang, 2012].

[3] In other papers (e.g., [Chang et al., 2020]), this approach is named as Migration-based

to reduce the probability of attack success through dynamic changes in the attack target (i.e., the hypervisor variant of the physical host). In particular, we consider the VM migration between heterogeneous hypervisors [Liu et al., 2008]. Focusing on simplifying the MTD deployment, the VM migration policy follows a circular approach for any of the considered architectures. The circular approach consists of migrating the VM from one physical machine (PM) pool to the next physical machine pool. For example, let us suppose that the *attacker* is in the PM1 pool in a *3N* architecture. We first migrate the VM from the PM1 pool to the PM2 pool. Then, from PM2 to PM3, in the next migration round. Finally, from PM3 to PM1, restarting the VM migration cycle.

For illustration purposes, consider the example in Figure 6.2 that shows an attack and defense flow in a *2N* architecture. ***Stage 1*** is the initial state. We suppose that the VM of the attacker is in the PM1 pool. After the *reconnaissance* phase, the attacker starts the attack phase. The time spent in the attack phase is counted as attack progress. Then, the VM migration schedule arrives, moving the VM of the attacker from the PM1 pool to the PM2 pool, thus starting ***Stage 2***.

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

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

The *tolerance level (TL)* metric shows how long the system takes to reach a specific probability of attack success. For example, let us suppose that in a scenario without MTD (baseline), $TL(10\%) = 10h$. These hypothetical results indicate that the system took 10 hours to reach a probability of attack success of 10%. From a system management perspective, assuming that the system can tolerate up to 10% of attack success, all the VMs with expected runtime above 10 hours should be considered for the MTD deployment. It is worth highlighting that, in all our evaluations, we assume the system is under attack during the entire evaluation.

The proposed MTD brings benefits as it is easy to use (assuming that VM migration is a common task in contexts such as IaaS cloud management). However,

Dynamic Platform.

unless we have very frequent migrations, which may affect VM availability, the *attacker* will eventually be successful. Hopefully, the MTD may provide valuable extra time to enhance the defensive mechanisms against *insider* attacks.

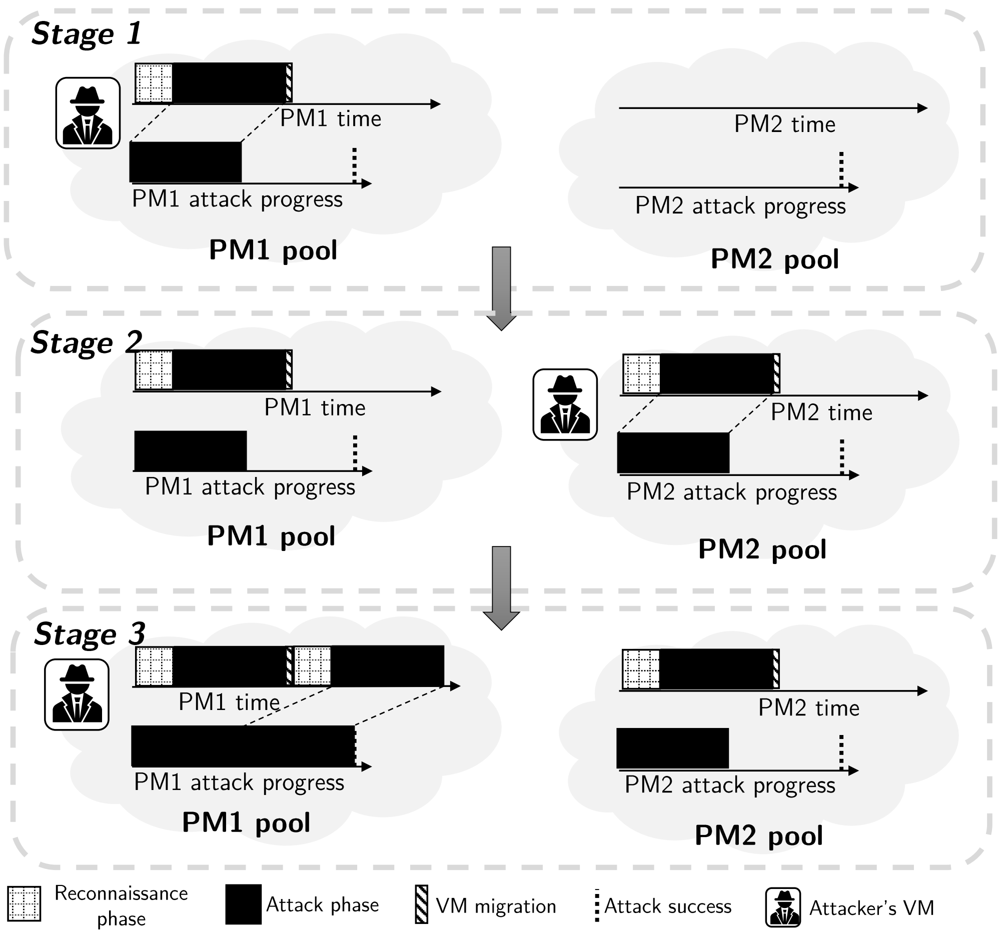


Figure 6.2: Example of attack and defense flow - *2N* architecture

6.2 Model

The proposed model has two submodels: a) CLOCK model and b) SYSTEM model (see Figure 6.3). The models interact through the guard functions described in Table 6.1[4]. For simplicity, this section presents only the models related to the *2N* architecture. Nevertheless, we consider up to the *4N* architecture in our evaluations. We emphasize that each architecture has its own model, and all the models follow the same approach used in the *2N* architecture model. The only difference is that the other architectures *3N* and *4N* have three and four physical machine pools, respectively. *1N* architecture has only one physical machine pool and does not support VM migration. Nevertheless, we present the results from *1N* architecture as ***baseline*** results.

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

Table 6.1: Guard functions

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

The CLOCK model represents the system component responsible for triggering VM migration. To assure that the model represents the predefined VM migration scheduling, transition `Trigger` adopts a deterministic firing delay. Its firing delay is the configured delay between VM migrations. Thus, transition `Trigger` firing represents that the system reached the time interval for VM migration. Transition `Trigger` firing removes a token from place `Counting` to the place `Schedule`, thus activating guard function $Gf2$.

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

The SYSTEM model in Figure 6.3 represents the *2N* architecture. Transitions and places related to the PM1 pool have an explicit mention of `PM1`, and transitions and places related to the PM2 pool have `PM2` in their names. In this model, we consider that the *attacker* VM is initially in the PM1 pool (place `Arrival_PM1` with one token). From this state, we have two possibilities. The first is when the VM migration schedule arrives before the attacker finishes the *reconnaissance* phase. In this case, obeying $Gf2$, the token from place `Arrival_PM1` is moved to place `VMMigPM1_2` through transition `MigPM1_2_1` firing. The second possibility is that the attacker finishes the *reconnaissance* phase (transition `Recon_PM1` firing). Transition `Recon_PM1` firing removes the token from place `Arrival_PM1` and puts a token in place `Attack_PM1`.

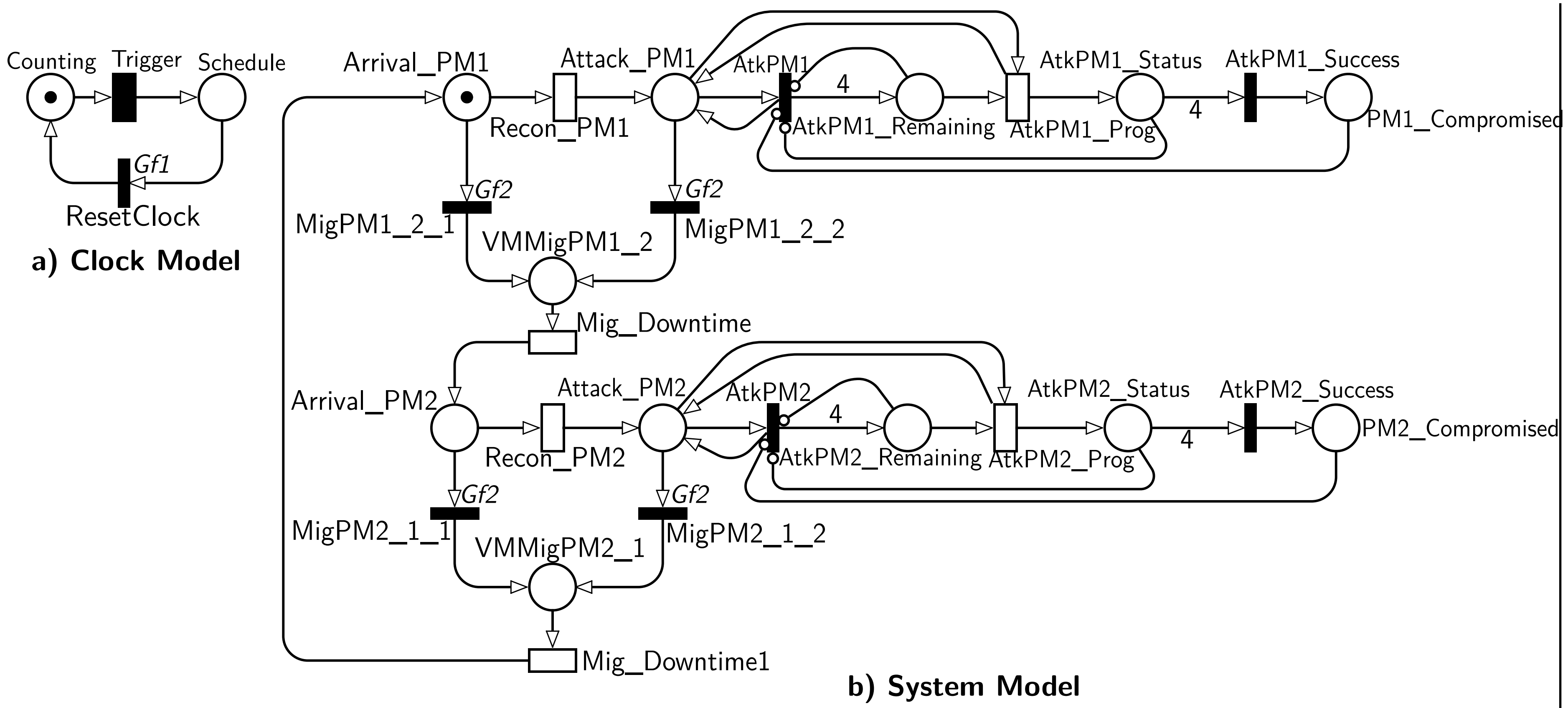


Figure 6.3: Model for probability of attack success evaluation (*2N* architecture)

Following the same approach of the previous chapters, we model the *attack* phase using a four-phase Erlang sub-net. The main goal is to reproduce the IFR behavior for the attack progress accumulation. More details will follow. The Erlang sub-net has the immediate transition `AtkPM1`, transition `AtkPM1_Prog`, and the places `AtkPM1_Remaining` and `AtkPM1_Status`. There are two reasons for this choice. First, we need to preserve the attack progress even after a VM migration. Thus, we need a model with a *enabling memory policy* [Marsan et al., 1998]. The Erlang sub-net (specifically the place `AtkPM1_Status`) serves as a *memory* of the attack progress. Secondly, a four-phase Erlang sub-net can represent an IFR [Trivedi, 1982]. As mentioned before, the attacker adopts a try-and-error approach. Thus, the attack success probability increases as long as the attacker stays in a specific physical machine pool. Therefore, we approximate this increasing probability using the four-phase Erlang sub-net. It is worth highlighting that previous works also adopted hypoexponential distributions to represent IFR [Machida et al., 2013; Torquato et al., 2020a].

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

Transition `AtkPM1_Success` immediately fires when the place `AtkPM1_Status` receives the fourth token. Transition `AtkPM1_Success` firing collects four tokens from place `AtkPM1_Status` and puts one token in the place `PM1_Compromised`. This event denotes that the attacker successfully compromised a physical machine in the PM1 pool. We add an inhibitor arc[6] from place `PM1_Compromised` to transition `AtkPM1`. This inhibitor arc denotes that the *attacker* does not need to pass the try-and-error approach again once the system is compromised.

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

The attacking approach in the PM2 pool follows the same procedure described above. When the system triggers another VM migration, the *attacker* VM is

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

moved back to the PM1 pool. The attacker can leverage the obtained knowledge from his first attempts in the previous *attack* phase in the PM1 pool.

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

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

6.3 Case Studies

This section presents two case studies. The first one estimates the probability of attack success in each proposed system architecture (Section 6.3.1). The second shows the results considering different VM migration scheduling policies (Section 6.3.2). We consider policies with *30 minutes*, *1 hour*, *6 hours*, *12 hours* and *24 hours* between VM migrations. Our results highlight the tradeoff between availability and security. For example, applying the policy *30 minutes* in a system with four physical machine pools, the probability of attack success at 24 hours is less than 1%. When applying the policy *12 hours* in the same conditions, the probability is 23%. However, the system downtime due to VM migrations at 24 hours is about 8 seconds for the policy *12 hours* and 3 minutes for the policy *30 minutes.*

We used the TimeNet tool [Zimmermann, 2017] for model design and evaluation. Table 6.2 presents the default values used for our evaluations. We obtained these values from the recent papers [Torquato et al., 2020a; Chen et al., 2020]. Note that these values are only for reference and should be adapted whenever measurement-based results are available. Nevertheless, we find these values reasonable to represent the considered threat model and MTD defense based on previous works published in reputed journals. We discuss this topic more in Section 6.4.

Table 6.2: Parameters used in the timed transitions

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

6.3.1 CS #1 - Varying Number of Available Physical Machine Pools

Considering the proposed threat model and MTD approach (see Section 6.1), we can observe that the higher the number of available physical machine pools, the longer the delay for an attack success. However, this case study aims to understand the attack success when using different system architectures. In practice, this case study has two goals: i) to evaluate how long the system survives the attack and ii) to quantify the benefits of enlarging the architecture. The results from this case study provide the answer to our *RQ1 - What is the reduction in the probability of attack success when using different system architectures?*

Figure 6.4 presents the results of the probability of attack success in the different architectures. These results comprise the first fifteen days (360 hours) of *attacker* presence in the environment.

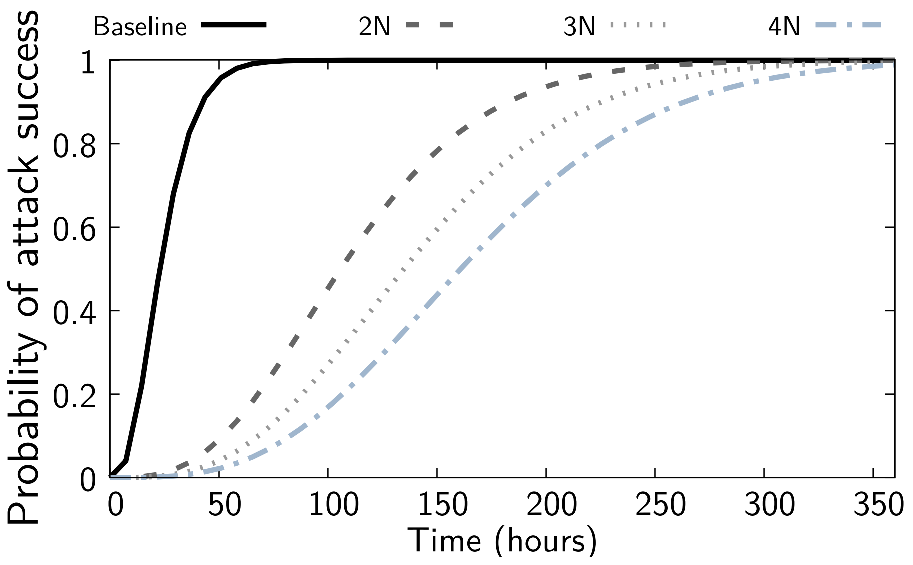


Figure 6.4: Probability of attack success - varying number of available physical machine pools

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

For comparison purposes, Table 6.3 presents when the system reaches 1%, 50%, and 90% of the probability of attack success (i.e., *tolerance levels*) in each proposed architecture. These findings may help to perceive the benefits of MTD deployment. The probability of an attack success decays significantly in the MTD-enabled environment. We noticed that the smaller architecture with MTD (*2N*) is about three times more *secure*[7] than the baseline architecture in the first week of attack presence (up to 168 hours), which provides an answer to RQ_1.

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

Table 6.3: When the system reaches probability of attack success of 1%, 50%, and 90% (tolerance levels)

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

The results presented in Table 6.3 also provide a partial answer for RQ_3. Depending on the business model, system managers may be more or less concerned about security. Thus, the system manager may define appropriate tolerance levels of probability of attack success for their respective environments. For example, in a business with a high associated security risk, the system manager may set the tolerance level at 1%. In this situation, considering the scope of our evaluation, the system should flag the VMs with an expected runtime above 6 hours as candidates for MTD deployment. After further verification of other relevant aspects (e.g., verification of what the client started the VM), the system manager can decide whether or not such a VM should follow the VM migration scheduling.

Figure 6.5 presents the results of the reduction in the probability of attack success. This reduction is the difference between the probability of attack success of each MTD architecture (*2N*, *3N*, and *4N*) and the ***baseline*** architecture. As expected, *4N* architecture provides a more significant reduction than the others. The interesting conclusion is that, after fifteen days (i.e., 360 hours) of attacker presence in the environment, the reduction is less than 1%. Therefore, in scenarios with VMs that impose long-running execution times (above fifteen days), a *4N* architecture is not enough to reduce the probability of attack success. In such scenarios, we encourage the system managers to apply alternative policies to mitigate the vulnerabilities related to *insider* attack. The deployment of periodical software rejuvenation and routines to clean up the hypervisor may improve system security in such scenarios.

6.3.2 CS #2 - Varying VM Migration Schedule - *4N* architecture

In the previous case study, we noticed security improvements due to the MTD deployment. However, as the MTD is based on VM migration scheduling, policies with frequent migrations may also affect system availability. In fact, Clark et al. [2005] shows that, even in live migration mode, each VM migration has an associated downtime. In some cases, the accumulated VM migration downtime may be unacceptable. Furthermore, the threat model and MTD defense impose a tradeoff between security and availability, as more frequent migrations may delay the attack success longer but also impose more system downtime, and less frequent migrations may reduce the system downtime due to MTD policy, increasing the probability of attack success. Thus, in the context of RQ_2, we want to evaluate different VM migration schedules to verify the impact on availability and probability of attack success.

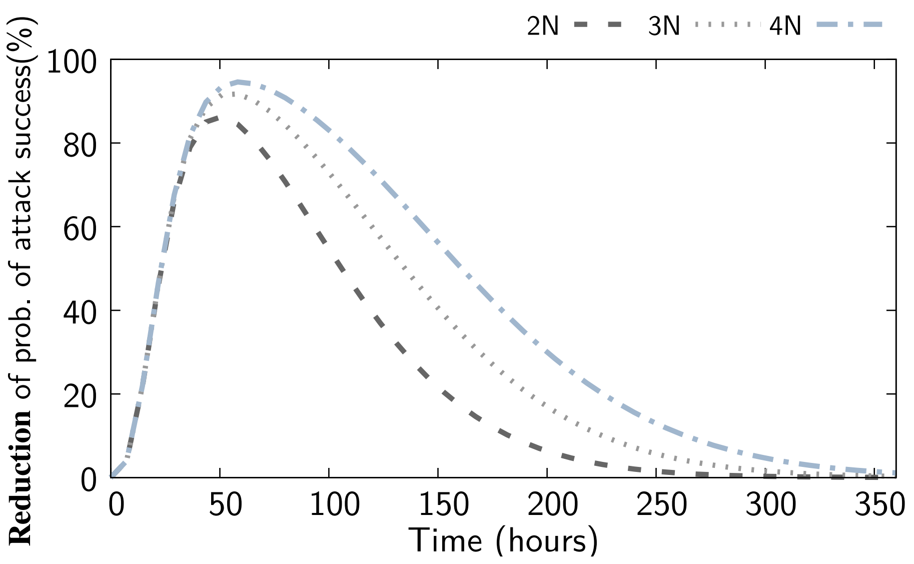


Figure 6.5: Reduction of probability of attack success due to the number of available physical machine pools

Figure 6.6 presents the probability of attack success results for different VM migration trigger intervals. Figure 6.7 presents the reduction due to each VM migration scheduling policy. The evaluation below presents the results for the *4N* architecture, as they were the most illustrative among the other considered architectures. As expected, more frequent migrations (e.g., VM migration scheduling of *30 minutes* and *1 hour*) produce a more noticeable reduction in the probability of attack success. In the first hours of attacker presence, less frequent migration policies provide no significant decrease in attack success probability. Besides that, Figure 6.7 shows that less frequent migrations produce only a slight reduction effect on the probability of attack success.

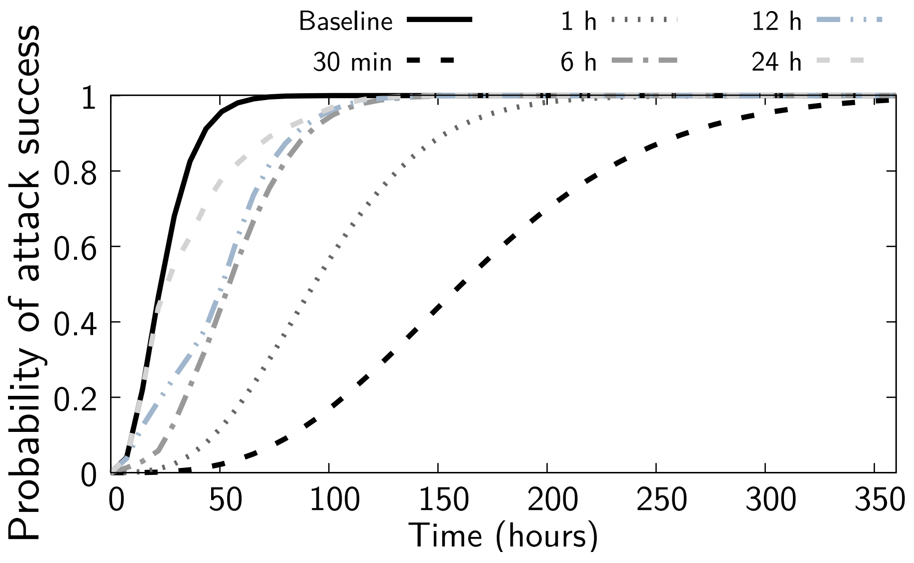


Figure 6.6: Probability of attack success - varying VM migration trigger - *4N* architecture

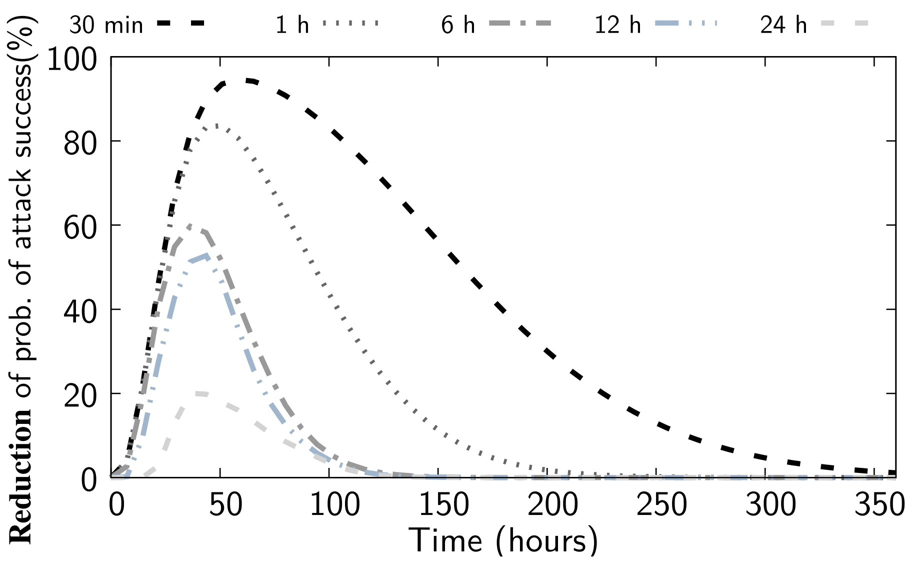


Figure 6.7: Reduction of probability of attack success due to the variation on VM migration trigger

System managers may be interested in two aspects of the VM migration scheduling when designing the MTD policy: i) **system availability**, to understand how the VM migration scheduling affects system downtime; and ii) when the system reaches the **tolerance level** (TL) of the probability of attack success. We summarized these results in Table 6.4, where we can see that, in systems with a tolerance level of 1%, the managers may adopt MTD policies with more frequent migrations (e.g., VM migration trigger of *30 minutes* or *1 hour*). Because the other ones provide no security improvement when compared to the *baseline* results.

Table 6.4: Tolerance levels and system unavailability - Different VM migration schedules - *4N* architecture

Trigger	**TL *1%***	**TL *50%***	**TL *90%***	**Avail. with VM migrations**	**Downtime in the first 15 days (due to VM migrations)**
30 min	41 h	161 h	263 h	0.997778	48 min
1 h	24 h	94 h	152 h	0.998889	24 min
6 h	6 h	54 h	89 h	0.999815	4 min
12 h	6 h	52 h	83 h	0.999907	2 min
24 h	6 h	23 h	71 h	0.999954	1 min

Availability results reveal a significant system downtime in scenarios with frequent migrations, but the downtime in the first fifteen days (i.e., 360 hours) is below 4 minutes when using VM trigger above *6 hours*. However, we highlight that the downtime of each VM migration is usually short, as the VM memory state is preserved during VM migration [Clark et al., 2005]. Therefore, VM migration downtime may produce a more severe impact on systems with an intense load of external requests. In scenarios without a network buffer, these requests will be lost due to environment unavailability during migrations. Nevertheless, there may be scenarios where this downtime is acceptable (e.g., systems that do not require high availability).

6.3.3 CS #3 - Validation with Simulation Results

Following the idea of Connell et al. [2018], we implemented a simulation environment using SimPy[8]. SimPy provides a simulation framework using standard Python language. We used SimPy to simulate the attack progress and the interruptions due to VM migration occurrence. We implemented the simulation script by hand (i.e., without using automation or conversion frameworks). The proposed model guided the simulation implementation, where we used variables to store the system state and events to represent the attack progress and VM migration.

Figure 6.8 presents: i) the comparison of model (black line) and simulation (gray dashed line) results, and ii) *error* - the difference between the model and simulation results. The simulation results dotted lines represent the 95% confidence interval. The results below are only from the *2N* architecture. However, the comparisons between simulation and model results for the other architectures have similar results.

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

The *error* results remain under 0.2 in all considered scenarios of VM migration scheduling. The maximum *error* for the policy *30 minutes* is 0.151552, and the maximum *error* for the policy *1 hour* is 0.051828. For all the other scenarios, the maximum *error* is about 0.01. Figures 6.8(e), 6.8(g), and 6.8(i) show that the simulation results are nearly the same as the model results.

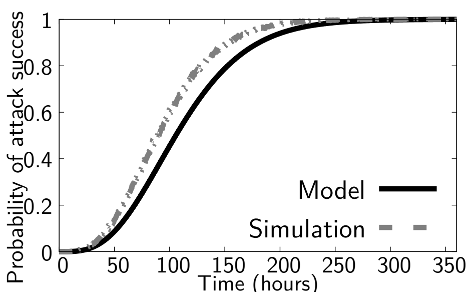
6.4 Threats to Validity and Limitations

Model-based evaluations are less accurate than measurement-based evaluations

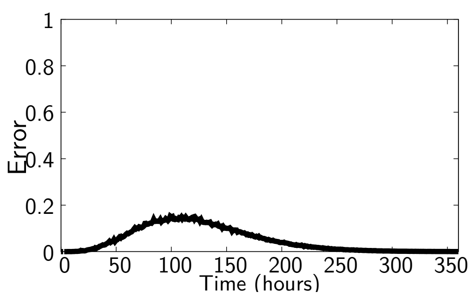
Model-based evaluation suits our needs as we aim to evaluate different architectures, which may be a significant challenge for measurement-based evaluation. Besides that, as presented in Chapter 5 and [Nguyen et al., 2020], model-based evaluation is helpful when combining security and dependability metrics. As our research line evaluates the probability of attack success and availability, SPN models seem to be a reasonable evaluation method.

The availability evaluation is limited

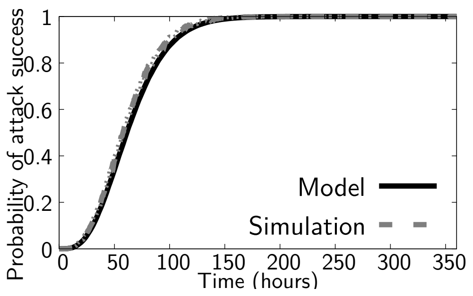
[8]Available at: https://simpy.readthedocs.io/en/latest/



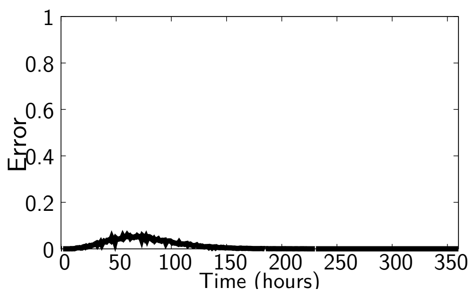
(a) `Trigger` = 30 min



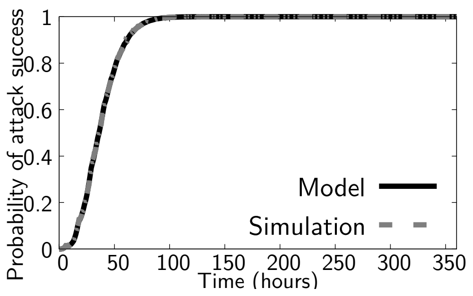
(b) `Trigger` = 30 min (*error*)



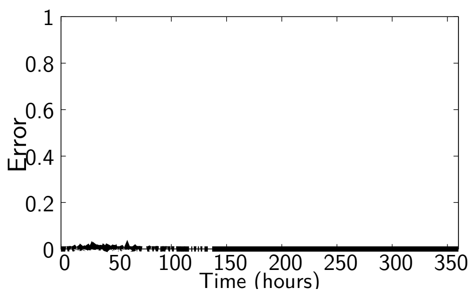
(c) `Trigger` = 1 h



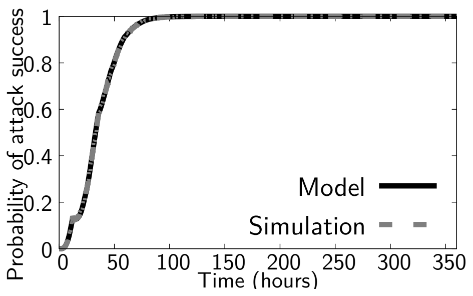
(d) `Trigger` = 1 h (*error*)



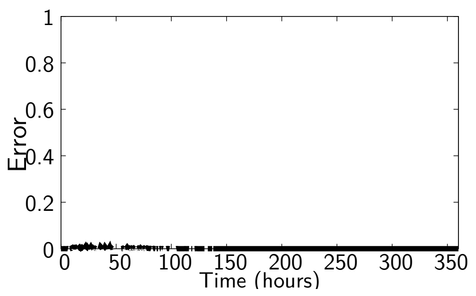
(e) `Trigger` = 6 h



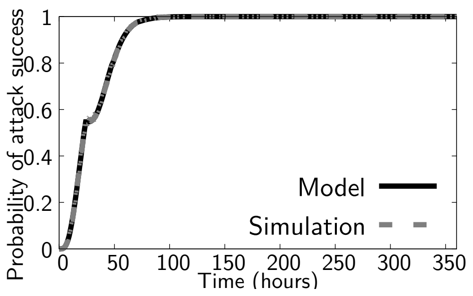
(f) `Trigger` = 6 h (*error*)



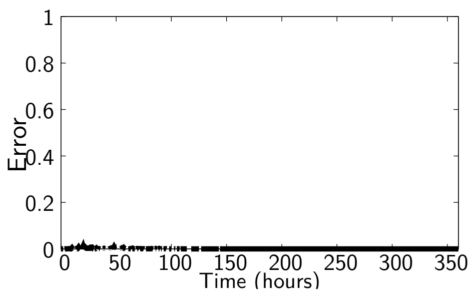
(g) `Trigger` = 12 h



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



(i) `Trigger` = 24 h



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

Figure 6.8: Model and simulation results - 2N architecture

A more comprehensive availability evaluation of VM migration effects is needed to understand the real impacts of a VM migration scheduling policy. Two factors are missing in our evaluation. Firstly, other relevant dependability events such as failures, crashes, and software aging. Secondly, the VM migration failure probability. Despite the relevance of these aspects, note that the focus of this work is the security evaluation (i.e., probability of attack success evaluation). The inclusion of other relevant aspects of VM migration scheduling may raise difficulties (e.g., largeness [Trivedi and Bobbio, 2017]) in the model evaluation.

Default values for the duration of the attack and reconnaissance phases

In the best scenario, we should have obtained these parameters through experimentation. However, such experimentation has a non-negligible associated cost. An approach is to hire a *red team* [Diogenes and Ozkaya, 2018] to test a real testbed. We decided to follow a lower-cost approach by collecting the parameters from published papers. We noticed that similar approaches were used in several papers of model-based security evaluations, such as Mendonça et al. [2020]; Wang et al. [2013]; Alavizadeh et al. [2018a].

6.5 Summary

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

Three research questions guided this research. The first one is about the effect of the architecture enlargement on the probability of attack success. We found out that, in the first week of attacker presence, the MTD minimum architecture has a probability of attack success three times lower than the system without MTD. The second research question focused on the availability aspect of the evaluations. We noticed that the proposed MTD deployment results in a tradeoff between *availability* and *probability of attack success* metrics. Therefore, the selection of a specific policy should follow each environmental requirement. The last research question brought the concept of the proposed metric *tolerance level*. In this question, we aimed to answer how long it takes for the system to reach a specific probability of attack success. Using the proposed *tolerance levels*, it is possible to select VMs as *candidates* for MTD deployment. Moreover, the set of results quantifies the benefit of enlarging system architectures. In some scenarios, the benefit of using large architectures is negligible.

This chapter brought an MTD evaluation while the system is under a persistent attack. However, there are attacks that adopt the **non-persistent tactic**. In such cases, the MTD action completely flushes the attack progress. Therefore, the attacker needs to restart the attack after each VM migration. In the following chapter, we evaluate the non-persistent tactic scenario through a SPN model and its analysis.

Chapter 7

Time-based VM Migration as MTD against Non-persistent Attacks

Assuming an insider that controls a VM and intends to collect sensitive information from the host (i.e., the physical machine that runs the *attacker's* VM), in some contexts, VM migration forces the attacker to restart the attack as the VM is moved to a different host [Sianipar et al., 2018]. However, VM migration (even in live migration mode) produces system downtime [Clark et al., 2005]. Depending on the VM migration frequency, the associated downtime may turn unacceptable. This way, the combined evaluation of security and availability impact due to VM migration as MTD is required to support MTD deployment.

The inspiring works [Connell et al., 2017, 2018] tackled similar problems. They present a model for evaluating the performance and availability of MTD. We consider some relevant aspects missing in their works, such as varying VM migration failure probability and increased attack success rates, for the specific context of non-persistent attacks. Besides that, we provide secondary metrics (i.e., *tolerance levels* and *effectiveness limits*) to support VM migration-based MTD design.

This chapter presents a SPN model for the evaluation of the probability of attack success and availability of an MTD based on VM migration scheduling. Our main goal is to provide an overview of how different aspects may impact MTD protection and system availability in the presence of non-persistent attacks. Namely, our evaluation focuses on the following aspects: i) VM migration scheduling, ii) VM migration failure probability, and iii) different levels of attack success rates. Specifically, the following research questions guided this research:

- RQ_1: What is the availability and attack success probability impact of different VM migration scheduling policies?
- RQ_2: Assuming a specific VM migration scheduling, how long does MTD protection last?
- RQ_3: What is the impact of different VM migration failure probabilities on the probability of attack success when deploying VM migration-based MTD?
- RQ_4: How do increased attack success rates affect MTD protection?

To answer these questions, we present three case studies. The first focuses on the impact of VM migration scheduling on the probability of attack success and availability. The second investigates the impact on the probability of attack success due to different levels of VM migration failure probability. The last shows how the protection level deteriorates when considering increased attack success rates.

To our knowledge, this is the first research to evaluate MTD based on VM migration that comprises different VM migration failure probabilities and attack success rates in the context of non-persistent attacks. Besides that, our secondary metrics, *tolerance levels* and *effectiveness limit*, may be beneficial to support the planning of MTD policies. Moreover, our case studies focus on the comparison of the different VM migration scheduling policies. We decided to conduct these comparisons to provide a comprehensive overview of the MTD impact under different conditions. **This chapter is adapted from Torquato et al. [2020b]**.

The rest of this chapter is organized as follows. Section 7.1 presents the evaluation approach focusing on the assumptions behind this research. Section 7.2 discusses the SPN model proposed. Section 7.3 presents the case studies. Section 7.4 presents threats to the validity and limitations of our work. Finally, Section 7.5 presents chapter conclusions.

7.1 Approach and Assumptions

The threat model considered is as follows. The *attacker* controls one VM and aims to conduct a host-based attack. In a successful host-based attack, the *attacker* targets the underlying physical machine [Fernandes et al., 2014]. *Host-based* attacks could affect the co-hosted VMs and the shared resources like CPU or memory. In this threat model, the *attacker* final goal is to collect sensitive information from the shared resources (namely, CPU or RAM). For example, the *attacker* can try to conduct a memory dumping [Schatz, 2007] or to exploit vulnerabilities similar to Meltdown and Spectre [Hill et al., 2019], like what is presented in Sianipar et al. [2018]. Thus, assuming the classic definitions in Avizienis et al. [2004], a successful attack causes a data *confidentiality* violation.

The proposed MTD mechanism is VM migration scheduling. The idea is to move the *Attacker* VM away from a physical machine, thus interrupting the attack progress. As mentioned, we assume that the goal is to compromise system confidentiality by collecting data from CPU or RAM. It is expected that the data stored in these components (henceforth, *target data*) change dynamically. This way, it is unlikely that the *target data* remains the same between *Attacker* VM migrations. To enforce this behavior, we assume that, once the migration is done, there is a reset on the sensitive processes, forcing a total remap on CPU and RAM. Therefore, unless the *attacker* completes the attack, it is necessary to start it from the beginning when the VM arrives in another host (i.e., VM migration completion).

Figure 7.1 shows an example of an attack and defense flow with timely VM migrations. In this example, we have two physical machines $PM(S)$ as VM migration source and $PM(T)$ as VM migration target. We assume that the *Attacker* VM is initially on the $PM(S)$. As in the threat model explained before, **Stage 1** corresponds to the *attacker* arrival, and **Stage 2** represents the attack progress. However, in **Stage 3**, the system triggers a VM migration. When the *Attacker* VM arrives on the $PM(T)$, the attack progress is lost. Then, the cycle restarts with the $PM(T)$ assuming the role of $PM(S)$.

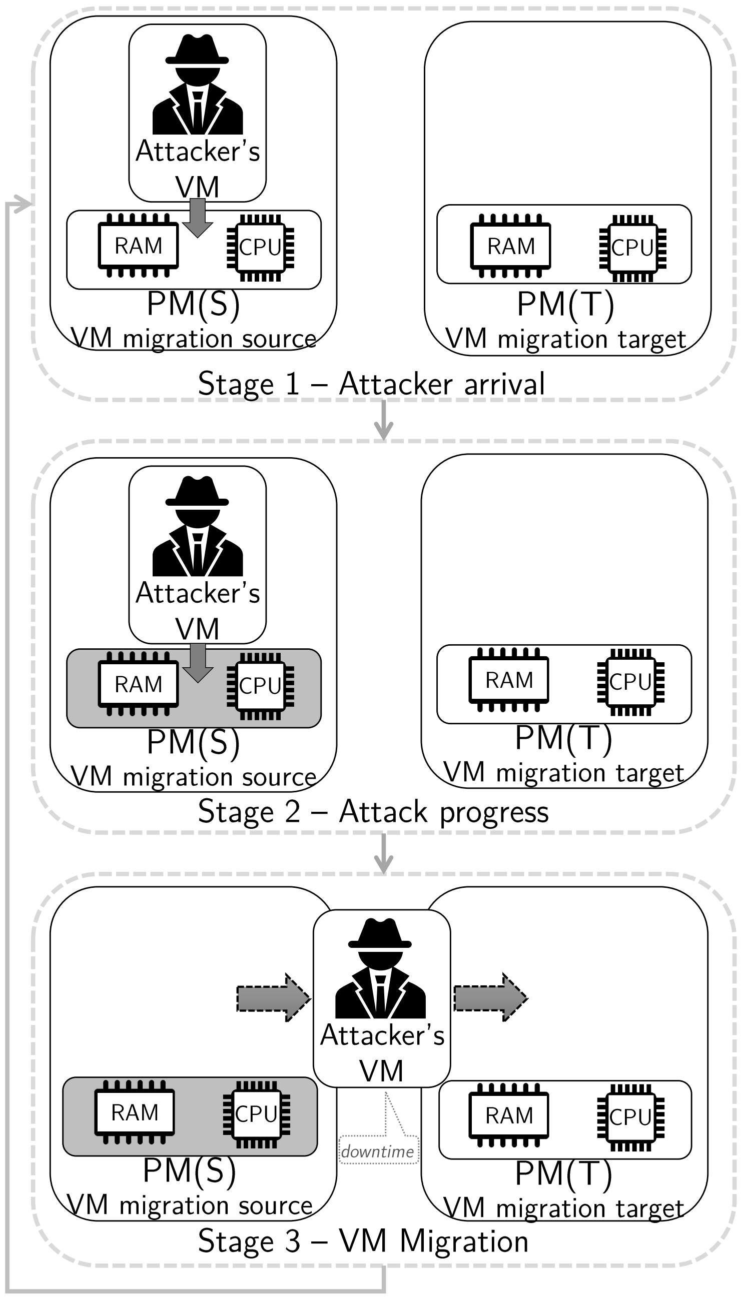


Figure 7.1: Example of attack and defense flow with timely VM migrations

We also assume that the attacker is *perseverant*, meaning that he will not resign until the system is compromised. Therefore, unless we have very frequent migrations (producing substantial system unavailability), the *attacker* will eventually succeed. Thus, in this scenario, the MTD deployment goal is to delay the attack success. This extra time may be valuable for managers to deploy attack detection mechanisms or harden the system architecture against the considered threat model.

As the MTD strategy is entirely based on moving the *Attacker* VM, an important question arises on how to detect the *attacker*. Moreover, once the *attacker* is detected, it is more straightforward to delete his VM instead of migrating it. System managers may define policies to select VMs as *candidates* for MTD deployment. To obtain maximum security, system managers may deploy MTD on all VMs in the environment (comprehensive deployment). However, in large data centers, comprehensive deployment may impose unacceptable costs.

To help in the selection of *candidates*, we also consider *tolerance levels*. *Tolerance level* (TL) results portray how long a VM may run to reach a specific attack success probability. For example, $TL(1\%) = 10$h means that the system reaches 1% of attack success probability after 10 hours of *attacker* presence. Therefore, assuming a *tolerance level* of 1%, all VMs with an expected runtime of 10 hours or more are *candidates* for MTD deployment. Besides that, we also provide *effectiveness limit* (EL) results. EL is the time instant when the system with MTD reaches the probability of attack success of 100%. However, we highlight that EL and *tolerance levels* provide only suggestions of what VMs should follow MTD. System managers may define additional criteria (e.g., VM user, resource consumption) for deciding the VMs for MTD deployment.

Our study focuses on MTD security modeling. This way, availability focuses only on the effects of VM migration (i.e., we neglect other causes of system unavailability). Besides that, our model also covers VM migration failure probability. In the case of a VM migration failure, we assume that the attack continues instead of being interrupted.

7.2 Model

Figure 7.2 presents the proposed SPN model. The model has two submodels: a) Clock model and b) System model. The model has embedded *guard functions*[1] described in Table 7.1.

Table 7.1: Guard functions

Guard	**Enabling function**
gf1	`(#VMMig>0) OR (#VMMigFail>0)`
gf2	`(#Schedule>0)`
gf3	`(#Counting>0)`
gf4	`(#VMMig>0)`

The Clock model represents the time counting for VM migration triggering. Place `Counting` with tokens enables the firing of transition `Trigger`. Transition

[1]Guard functions are Boolean expressions evaluated based on the net current marking. They disable the associated transition when the boolean expression returns *false*.

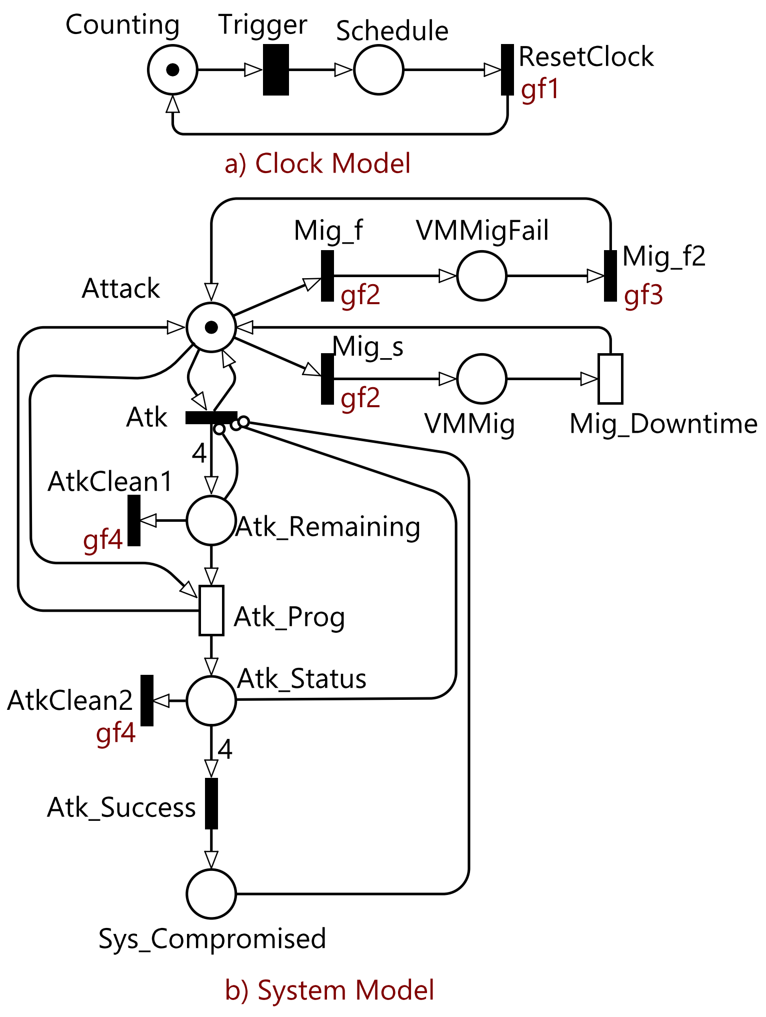


Figure 7.2: Proposed SPN model

`Trigger` has an associated deterministic delay, which represents the time interval between VM migrations triggering. Transition `Trigger` firing moves the token from place `Counting` to place `Schedule`. The token arrival on place `Schedule` represents the VM migration triggering. To control the model workflow, we use `gf1` to ensure transition `ResetClock` firing after the firing of transition `Mig_s` or `Mig_f` (indicating the time counting restart after VM migration triggering). Transition `ResetClock` firing moves the token back from place `Schedule` to place `Counting`, thus restarting the cycle of Clock model.

The System model covers the VM migration and the attack progress behaviors. We assume that the *Attacker* VM is active at the initial state of the evaluation (i.e., *Attacker* arrival stage of our threat model). Thus, at the initial state, the place `Attack` has one token.

Transition `Mig_s` represents VM migration success, and transition `Mig_f` represents VM migration failure. Both transitions have the guard function `gf2`. Their firing depends on tokens presence in places `Attack` and in the place `Schedule`. Besides that, the transitions `Mig_s` and `Mig_f` are concurrent. Therefore, just one of them fires when the enabling conditions are satisfied.

Transition `Mig_f` firing removes the token from place `Attack` to place `VMMigFail`. Place `VMMigFail` is just a control place that serves to avoid an infinite loop. For that reason, we associate `gf3` in transition `Mig_f2`. Note that both transitions `Mig_f` and `Mig_f2` are immediate, meaning that we do not consider system downtime in a VM migration failure. Alternatively, transition `Mig_s` firing moves the token from place `Attack` to place `VMMig`. The presence of tokens in place `VMMig` enables `gf4` (i.e., *attack* interruption, more details later on this section). Transition `Mig_Downtime` represents the system downtime due to a successful VM migration. After this downtime, the system returns to activity (i.e., transition `Mig_Downtime` firing, moving the token back to place `Attack`).

We use a coverage factor (*migFprob*) to express the percentage probability of VM migration failure. Transition `Mig_f` embeds coverage factor of *migFprob*. And, transition `Mig_s` embeds a coverage factor of $100-migFprob$. For example, assuming $migFprob=10$ means that, when the conditions are satisfied, transition `Mig_f` has 10% of chance of firing, while `Mig_s` has 90%.

Note that the modeling of attack progress is a rather difficult task due to *attacker's* behavior unpredictability. Our best attempt was, as Chen et al. [2020], to use combined exponential transitions for representing attack progress. We assume that the longer the *attacker's* VM stays on the same physical machine, the higher the probability of attack success. In the reliability models domain, this behavior is known as IFR [Trivedi and Bobbio, 2017]. Thus, using the same ideas of our availability and MTD models presented in the previous chapters (specifically, Chapters 4, 5 and 6), we adopt a four-phased Erlang sub-net to represent IFR. In the model, the Erlang sub-net places and transitions have the prefix `Atk`.

Transition `Atk` immediately fires when place `Attack` has tokens. Transition `Atk` firing swaps[2] the token from place `Attack`. It also puts four tokens in the place `Atk_Remaining` (representing the number of Erlang phases). Each time that transition `Atk_Prog` fires, it removes one token from place `Atk_Remaining` to the place `Atk_Status` (representing the attack progress). Transition `Atk_Prog` also swaps the token in the place `Attack` (i.e., the attack progress depends on the presence of *Attacker* VM on the physical host). Besides that, the guard function `gf4` enables the firing of transitions `AtkClean1` and `AtkClean2`. These transitions firing remove all the tokens from places `Atk_Remaining` and `Atk_Status`. This behavior represents the attack progress removal after VM migration.

When place `Atk_Status` stores the fourth token, the transition `Atk_Success` fires. Transition `Atk_Success` firing removes four tokens from place `Atk_Status` and puts one token in the place `Sys_Compromised`. Transition `Atk_Success` firing represents the attack success.

We use inhibitor arcs (arcs terminating in a circle instead of an arrowhead) from the places of the Erlang sub-net to the transition `Atk` to avoid infinite loop (i.e., continuous firing due to the token presence in place `Attack`).

Considering $P\{\#\texttt{P0} > 0\}$ as the probability of token presence in a place named `P0`, it is possible to compute our desired metrics as follows:

[2] receives and gives back

- **Probability of attack success** - Probability of succeeding in collecting sensitive data from CPU or RAM - = $P\{\#\texttt{Sys\_Compromised>0}\}$
- **Availability** - VM availability - = $P\{\#\texttt{Attack>0}\}$

7.3 Case Studies

This section presents the results of case studies. In the first (Section 7.3.1), we study the impact of different policies of VM migration scheduling on the probability of attack success. The second (Section 7.3.2) shows our analysis regarding different VM migration failure probabilities. The last (Section 7.3.39 considers different *attack* success rates. We design and evaluate the models using the TimeNet tool [Zimmermann, 2017]. Table 7.2 presents the default values used for our evaluations. We obtain these values from recent works [Torquato et al., 2020a; Chen et al., 2020; Kukrál et al., 2015].

Table 7.2: Parameters used in the model

Transition	**Description**	**Delay**
`Trigger`	Time for VM migration	30 minutes
`Atk_Prog`	Attack phase (Erlang)	6 hours*
`Mig_Downtime`	VM migration downtime	4 seconds
migFprob	VM migration failure probability	10%
* As we have four Erlang phases, *attack* total delay is of 24 hours		

7.3.1 CS #1 - Varying VM Migration Scheduling

VM migration is a usual task in cloud computing management. There are benefits due to VM migration deployment. Namely, the improvement of system sustainability (through server consolidation [Ahmad et al., 2015]), the security improvement (through moving target defense [Jia et al., 2014]), or dependability improvement (through software rejuvenation [Melo et al., 2013b]). Nevertheless, there are also drawbacks such as system downtime [Clark et al., 2005] or performance overhead [Voorsluys et al., 2009]. Observing these possibilities, the primary goal of the first case study is to evaluate the probability of attack success and availability of an MTD deployment based on VM migration scheduling (i.e., answering $RQ1$).

We compute the probability of attack success (Figure 7.3) during the first month (i.e., 720 hours) of *attacker's* VM presence in the environment. We consider five migration policies with different time interval between migrations (*30 minutes*, *1 hour*, *6 hours*, *12 hours* and *24 hours*). We also include the results without VM migration (*baseline*).

As expected, the VM migration scheduling produces a flattening effect in the curve of the probability of attack success. The significance of the flattening effect is related to the frequency of VM migrations. Frequent migrations, as in the policies *30 minutes* and *1 hour*, produce more reduction in the probability of

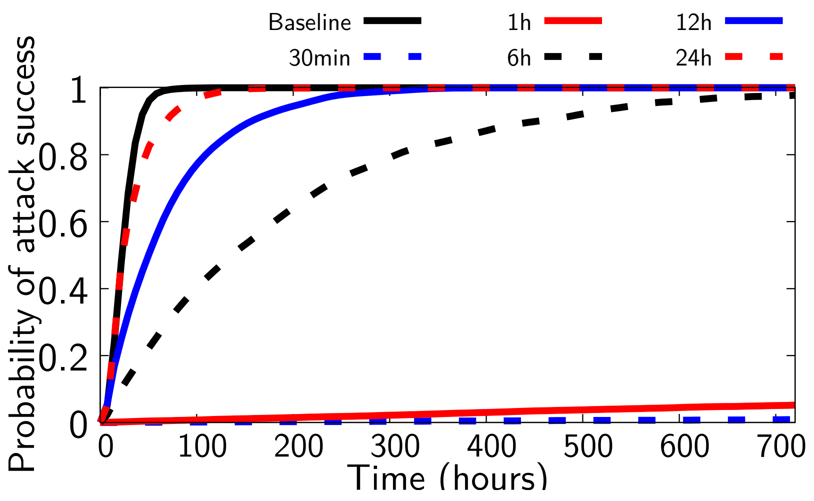


Figure 7.3: CS#1 - Probability of attack success results - different VM migration scheduling policies

attack success than the others. MTD protection becomes slighter when using VM migration policies with less frequent migrations. However, the quantification of the probability of attack success is essential to compare the proposed policies.

For example, the policy *24 hours* produces nearly the same result as the *baseline*. Specifically, the probability of an attack success at 720 hours (i.e., one month) of *attacker* presence is of 100% for *baseline* and policies *12 hours* and *24 hours*. For policy *6 hours*, the same probability is 97.75%. Finally for policy *1 hour* the result is 5.19%, and for policy *30 minutes*, 0.87%.

For a better understanding of MTD protection, we also calculate the reduction of the probability of attack success. This reduction is the difference between the probability of attack success with and without MTD (*baseline*). The results are in Figure 7.4. The reduction of probability of attack success is revealed when the MTD protection starts to decay. Moreover, it is possible to verify the EL of the MTD. For example, assuming the policy *24 hours*, the MTD produces no protection if the *attacker's* VM persists for more than 142 hours. EL results provide answers to RQ_2.

The plots in Figures 7.3 and 7.4 are useful to notice the impact due to MTD deployment policies. However, in practical situations, the system managers may have other concerns when considering this MTD approach. We list three aspects that may be of interest for managers: i) *deployment*; ii) *security benefits*; and iii) *imposed downtime*. About *deployment*, we see that the MTD approach is convenient as it is based on a usual management task (i.e., VM migration); about *security benefits*, the previous results demonstrate the MTD protection on each proposed VM migration policy, and regarding *imposed downtime*, Table 7.3 presents the steady-state availability and monthly downtime related to each VM migration policy.

Another relevant concern when designing MTD deployment policies is the MTD protection level. Some systems require more protection than others. *Tolerance*

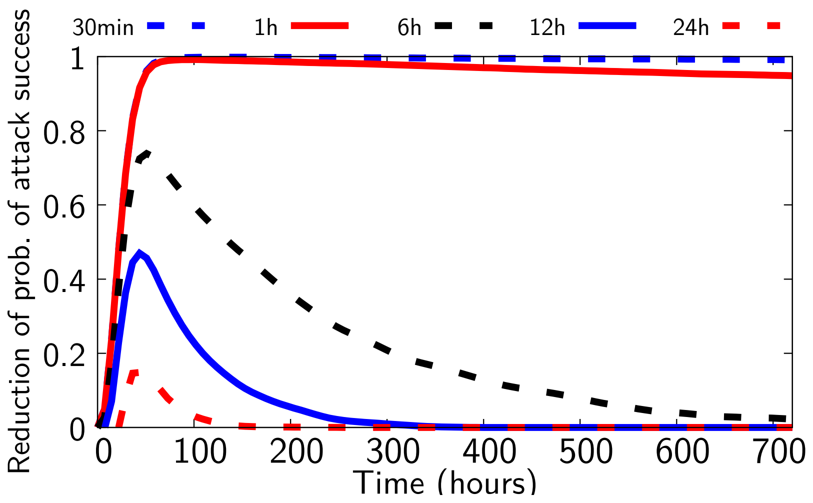


Figure 7.4: CS#1 - Reduction of probability of attack success - different VM migration scheduling policies

level results indicate how long the VM may run to reach a specific probability of attack success. Thus, these results may be useful in selecting VMs to follow MTD and observing their expected runtime. We also add the tolerance level results in the Table 7.3 that includes seven columns: **Policy** - indicating the VM migration policy; **Availability** - the expected steady-state availability of VMs which follow MTD deployment; **Dwt. 720h** - downtime due to VM migration in the first month of MTD deployment; results for *tolerance levels* of 1%, 25% and 75% are in the columns **TL 1%**, **TL 25%** and **TL 75%**, respectively; and **EL** column presents the results for the *effectiveness limit*. Note that we consider a year with 365 days and a month with 30 days.

For policies *30 minutes*, *1 hour*, and *6 hours*, some of the results of **TL 1%**, **TL 25%**, **TL 75%**, and **EL** were missing from our initial analysis (up to 720 hours). Thus, we performed a deeper evaluation (up to 1,000,000 hours with 100,000 sampling points) to obtain these values. For brevity, we omit the complete set of these results from the plots. Besides that, we decided to present only the plot of the first month because it was more significant for graphical comparison.

The *30 minutes* policy maintains the probability of attack success below 1% throughout the first month of *attacker* presence. It achieves the best security results. **EL** results show that its effect persists for more than eighty years. Besides that, **TL 1%** result shows that the probability of attack success only reaches 1% after one month and a half of *attacker* presence.

Frequent migrations may impose unacceptable system unavailability. In the case of the policy *30 minutes*, the downtime is about one hour and a half only in the first month of the MTD deployment. In a year, the accumulated downtime surpasses 17 days. We emphasize that this downtime is only due to VM migrations, and when considering other circumstances (e.g., preventive maintenance or system failures), it will probably be higher. Nevertheless, this downtime is sliced in small portions (i.e., around four seconds per migration according to Table 7.2). Thus, this MTD deployment may be suitable for systems without high availability and

Table 7.3: Summary of results of CS#1

Policy	**Avail.**	**Dwt. 720h**	**TL 1%**	**TL 25%**	**TL 75%**	**EL**
30 min	0.998000	86.4 min	1170 h (48.75 days)	26310 h ($\approx$ 3 yrs)	136061 h (15.53 yrs)	703047 h (80.25 yrs)
1 h	0.999000	43.2 min	126 h (5.25 days)	3620 h (5.03 months)	18980 h (2.16 yrs)	96620 h (11.03 yrs)
6 h	0.999833	7.2 min	5 h	58 h	263 h	1650 h (2.29 months)
12 h	0.999917	3.6 min	5 h	22 h	94 h	330 h
24 h	0.999958	1.8 min	5 h	15 h	41 h	142 h

high-continuity requirements.

Generally, the policy *1 hour* achieves similar results to the policy *30 minutes* in the first month of *attacker* presence. Significant differences between these approaches only become noticeable after a long period of *attacker* presence. The policy *1 hour* provides a slightly better availability result than the policy *30 minutes.* The policy *1 hour* maintains the probability of attack success below 10% in the first two months of *attacker* presence.

The policy *6 hours* appears to be a reasonable choice for systems with higher availability requirements. It maintains the monthly downtime below 10 minutes. From a security perspective, its protection benefits last for more than two months. When obeying this policy, the probability of attack success is under 25% in its first two days of running.

Policies *12 hours* and *24 hours* present poor MTD protection results. However, their adoption may be valuable in scenarios with general-purpose VM migrations. For example, in small data centers with preventive maintenance once or twice a day. In this situation, since the VM migration schedule is already in place, the results reveal the associated security benefits (considering the proposed threat model).

7.3.2 CS #2 - Varying VM Migration Failure Probability

Several problems may lead to VM migration failure. For example, the unavailability of physical machines to receive VM migration, issues in the network devices, and network congestion. This way, the probability of a VM migration failure (*migFprob* factor, as explained in Section 7.2, will depend on the environmental conditions. As our MTD is based on VM migration, *migFprob* may have a significant impact on MTD protection. This case study aims to investigate the impact of *migFprob* in the probability of attack success (i.e., answering RQ_3). For this purpose, we consider *migFprob* of 10% (same as the previous example), 30%, 50%, and 70%. We also add the *baseline* results.

For the analysis, we consider the system behavior when applying the policy *30 minutes*, as this policy achieved the best security results. Following the same approach as the previous illustrative example, Figure 7.5 presents the probability of attack success results, and Figure 7.6 shows the reduction of the probability of attack success due to MTD.

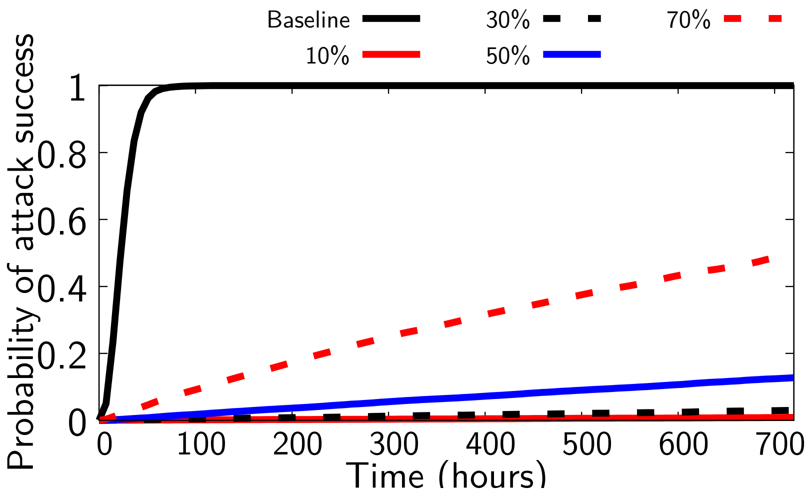


Figure 7.5: CS#2 - Probability of attack success results - different VM migration failure probabilities

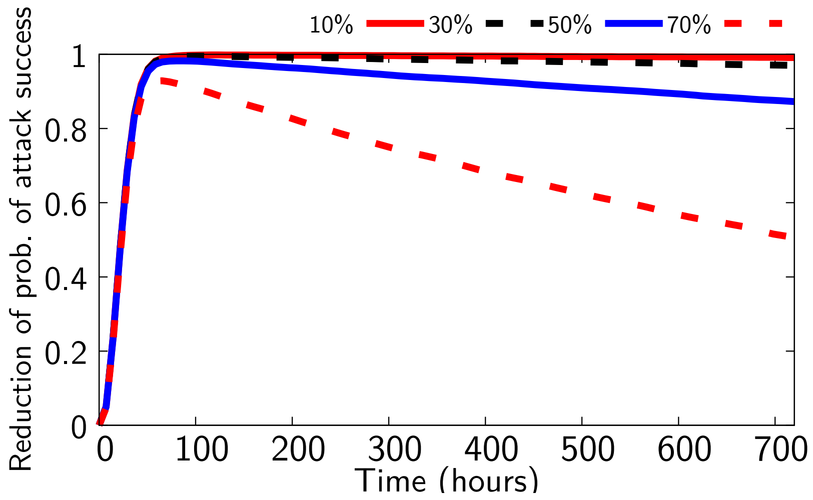


Figure 7.6: CS#2 - Reduction of probability of attack success - different VM migration failure probabilities

Figure 7.5 highlights the protective effect due to VM migration-based MTD deployment. The results show that, even in the scenario with only 30% of successful migrations (i.e., *migFprob*= 70%), it provides a significant reduction in the probability of attack success when compared to the baseline scenario. In fact, as shown in Figure 7.6, up to 75 hours, the reduction of the probability of attack success curve is nearly the same for the proposed values of *migFprob*. Thus, the MTD protection degradation due to high *migFprob* may be noticeable only after this period. Therefore, active monitoring of the VM migration success rate is needed for proper MTD deployment management.

Higher VM migration failure probabilities jeopardize the MTD protection, as expected. Interestingly, *migFprob* increasing from 10% to 30% causes a negligible difference in the probability of attack success, while the increase from 50% to 70% produces a significant impact. This behavior suggests a non-linear relation between *migFprob* and the probability of attack success.

We study the relation between *migFprob* and the probability of attack success in more detail. Figure 7.7 presents the probability of attack success at 720h for different *migFprob* values. *MigFprob* values range from 0% (system without VM migration failure) to 100% (*baseline* scenario) using a 10% step.

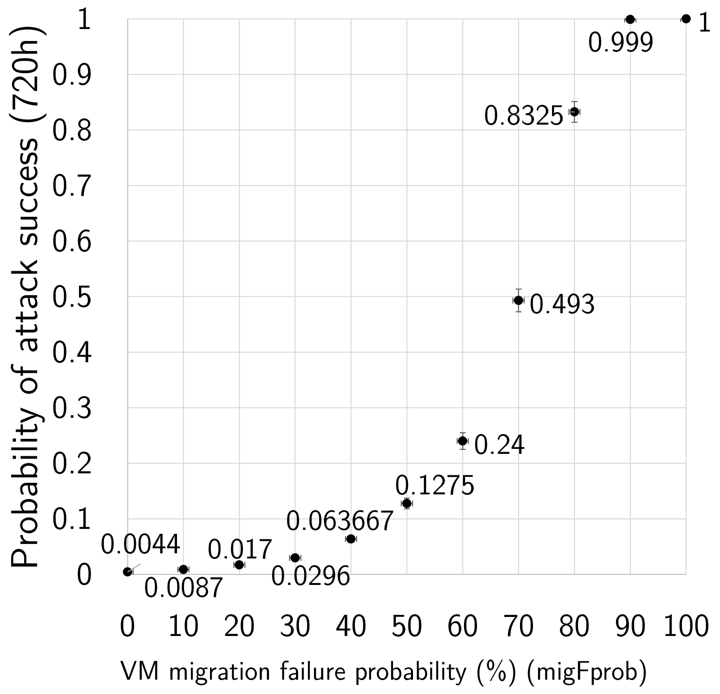


Figure 7.7: CS#2 - Impact of *migFprob* on the probability of attack success (at 720h)

To maintain a probability of attack success below 10% during the first month, *migFprob* should be below 40%. We notice a rapid increase in the probability of attack success when *migFprob* surpasses 50%. Therefore, system managers may stay alert to verify their migration success rate to avoid undermining the efficiency of MTD protection.

To conclude this illustrative example, we summarize results in Table 7.4. **PAS 720h** corresponds to the probability of attack success at 720 hours. The other columns have the same meaning as for Table 7.3. As in the previous case study, the results in Table 7.4 may be useful in the selection of VMs as *candidates* for MTD deployment (depending on the *tolerance level*). **EL** column results show

how long the MTD protection lasts in the different levels of VM migration failure probability. We can see the *migFprob* increasing causes more impact in the long term. For example, the *migFprob* increased caused a severe reduction in the EL. In the scenario with *migFprob* of 10%, EL surpasses 80 years, while in the scenario with *migFprob* of 70%, EL is less than one year.

Table 7.4: Summary of CS#2 results

migFprob	**PAS 720h**	**TL 1%**	**TL 25%**	**TL 75%**	**EL**
10%	0.87%	1170 h (48.75 days)	26310 h ($\approx$ 3 yrs)	136061 h (15.53 yrs)	703047 h (80.25 yrs)
30%	2.96%	250 h (10.41 days)	6520 h (9.05 months)	28980 h (3.30 yrs)	170281 h (19.43 yrs)
50%	12.75%	61 h (2.54 days)	1592 h (2.21 months)	6924 h (9.61 months)	39082 h (4.46 yrs)
70%	49.30%	14 h	300 h (12.5 days)	1416 h (1.96 months)	7424 h (10.31 months)

We emphasize that these results are only values for reference, and they may vary depending on the circumstances. Besides that, we discourage the maintenance of VMs with very long execution times. With a very long execution time, the *attacker* may find other paths to conduct an attack. Moreover, benign VMs may suffer from software aging problems [Grottke et al., 2008].

7.3.3 CS #3 - Varying *Attack* Success Rates

Depending on the reward upon an attack's success, some companies are more susceptible to attracting *attackers* with higher technical ability than others. The *attacker* technical ability is directly related to the time needed to reach an attack success. Therefore, *attackers* with higher technical skills have an increased chance of attack success. Consequently, we consider the different technical abilities of an *attacker* as different rates for attack success.

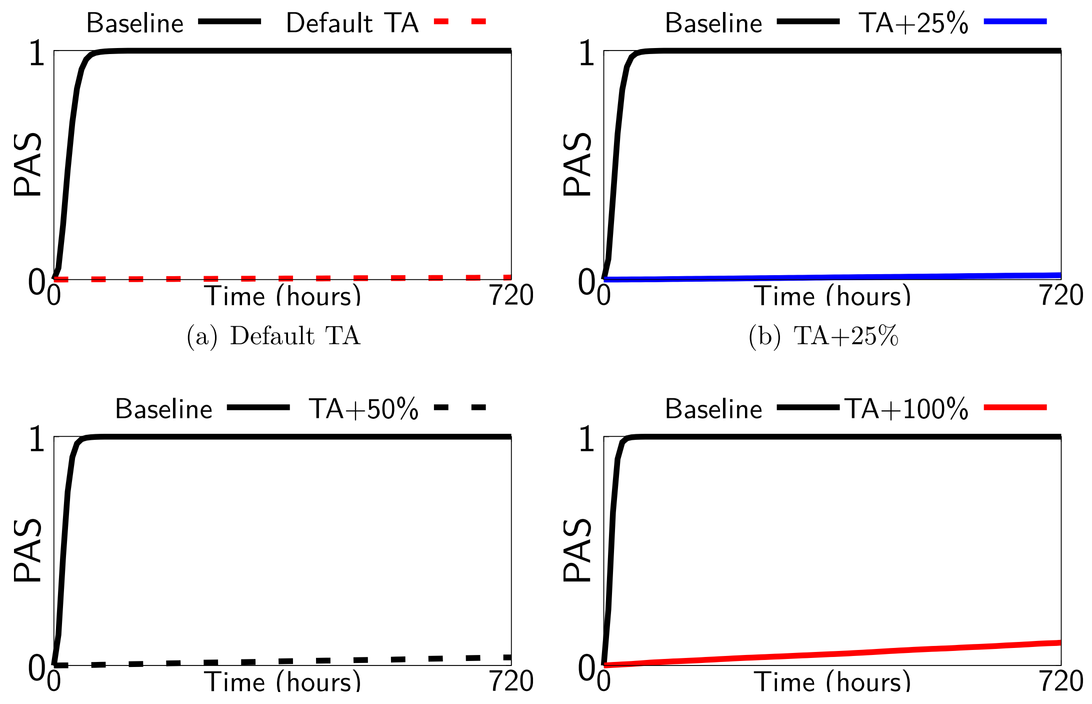
This case study aims to investigate how this increase in the attack success rate affects MTD protection (i.e., answering RQ_4). Observing what was presented in Maciel et al. [2018], we also propose four levels of *attacker* technical abilities (TA), meaning four different rates for attack success. Table 7.5 displays the time for attack success and the associated Erlang phase delay.

We consider the policy *30 minutes* and *migFprob* of 10% in the results. Figure 7.8 presents the results of the probability of attack success (PAS). Note that the *baseline* results are different in each scenario due to different rates of attack

Table 7.5: Attack success rates for the proposed technical abilities

***Attacker* TA**	**Time for attack success**	**Erlang phase (transition `Atk_Prog`)**
Defaut TA	24 h	6 h
TA+25%	19.2 h	4.8 h
TA+50%	16 h	4 h
TA+100%	12 h	3 h

success. Furthermore, as in the previous illustrative examples, we also present Figure 7.9 with the reduction of probability of attack success due to MTD deployment.



(a) Default TA (b) TA+25% (c) TA+50% (d) TA+100%

Figure 7.8: CS#3 - Probability of attack success - different levels of *attacker's* technical abilities

The comparison between Figure 7.8(a) and Figure 7.8(d) reveals a non-negligible impact in the PAS due to increased chance of attack success. Figure 7.9 shows a similar reduction in PAS for *Default TA*, *TA+25%*, and *TA+50%*. However, the reduction in PAS for *TA+100%* presents steeper decay.

For a better understanding of the impact of increased attack success rates in MTD protection, we present Table 7.6. Instead of exhibiting *tolerance levels* and *effectiveness limit* results, as in the previous examples, maybe it is more useful to consider two other metrics: i) the probability of attack success at 720 hours

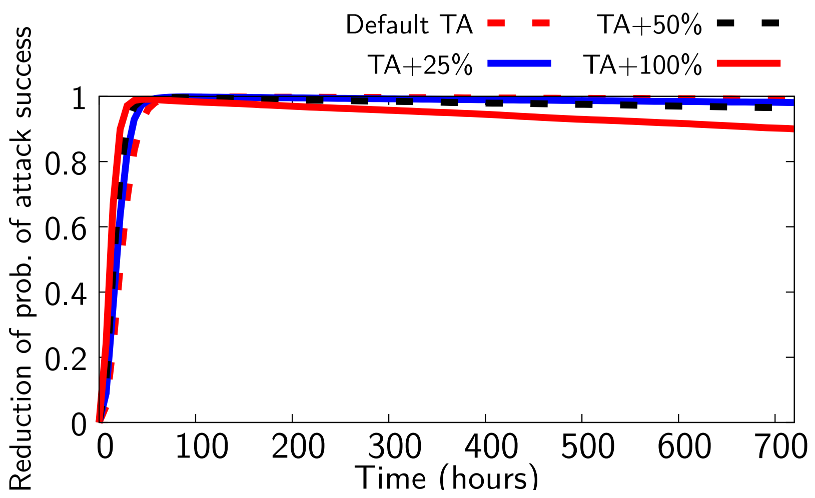


Figure 7.9: CS#3 - Reduction of probability of attack success - different levels of *attacker's* technical abilities

(**PAS 720h**), and ii) the reduction in the probability of attack success at 720 hours (**RPAS 720h**). *RPAS* is obtained by comparing the probability of attack success with and without MTD (*baseline*).

Table 7.6: Summary of CS#3 results

***Attacker* TA**	**PAS 720h**	**RPAS 720h**
Defaut TA	0.87%	99.13%
TA+25%	1.89%	98.11%
TA+50%	3.54%	96.46%
TA+100%	9.98%	90.01%

The results from Table 7.6 show that the MTD protection stays above 90% in all the considered attack success rates. Besides that, the proposed MTD maintains the probability of attack success under 10% during the first month of *attacker* presence. However, we emphasize that the policy considered in this example was *30 minutes*. Policy *30 minutes* achieved the best security results in the analysis presented in the first case study (Section 7.3.1). While the impact of an increased attack success rate is relatively low in this scenario, it may be more severe in situations with less frequent migrations (i.e., lower level of MTD protection).

7.4 Threats to Validity and Limitations

Lack of comparison with real testbed results

We put our best effort into designing models capable of representing realistic scenarios. However, our results were not compared with the results of real testbed experimentation. Therefore, this may raise a justified concern about the validity of our work. Unfortunately, security benchmarking is still an open challenge due to the complexity of the problem. Therefore, a consensual metric is not yet consolidated [Neto and Vieira, 2011; Knowles et al., 2015].

The same applies to the Moving Target Defense context, where we have diverse metrics to measure security (e.g., detection rate [Pacheco et al., 2016], impact on surface diversity [Pasupulati and Shropshire, 2016] or reconnaissance time [Jin et al., 2019]). Thus, finding a reliable approach to evaluate the probability of attack success is a rather difficult task. As an alternative, we decide to follow the modeling approach. The modeling approach appears as an alternative where the evaluation through experimentation is hard to conduct [Jain, 1990]. Finally, this research aims to deliver an evaluation method based on a consolidated paradigm SPNs for MTD based on VM migration.

Parameters retrieved from previous works instead of experimentation

The ideal situation is when we compute the parameters of the model from measurements in a real testbed. However, our research relies on previously published papers to retrieve those parameters. The reproduction of the threat model in a real virtualized environment is indeed one of our future works. Nevertheless, it is important to highlight that previous works on the same topic [Mendonça et al., 2020; Alavizadeh et al., 2018a] also retrieved parameters from the literature. Finally, the reproduction of the threat model in a real testbed may have a non-negligible cost. This reproduction needs a dedicated virtualized infrastructure and may require a specialized *red team* [Diogenes and Ozkaya, 2018] to conduct the attacks.

Lack of performance overhead evaluation

The evaluation of the performance overhead due to MTD deployment is important to support well-rounded decision-making. Although the performance evaluation is outside of the scope of this work, it is important to highlight that it plays a significant role in the selection of MTD policies. Note that, when taken into account, the performance impact may alter the suggestion of a VM migration scheduling. In Chapter 4, we show an approach to include performance in the evaluation. However, the proper inclusion in the security model is yet to be done. An alternative, in the current state of the research, is to adjust the model parameters to better reflect performance-wise scenarios. For example, increasing the VM migration failure probability to represent an overloaded network.

7.5 Summary

This chapter presented a SPN for the probability of attack success and availability evaluation of an MTD technique based on VM migration scheduling for non-persistent attacks. In three illustrative examples, we investigated MTD effectiveness under different VM migration scheduling, VM migration failure probability, and attack success rates. We delivered secondary metrics as *tolerance levels* and *effectiveness limit* to enhance our analysis. Our research may provide inputs on the designing of similar MTD policies.

Four research questions guided this research. The first one is about the evaluation of availability and probability of attack success in scenarios with MTD. Our results highlighted that there is a tradeoff between protection and availability when using the proposed MTD - therefore, system managers may decide, based on their requirements, which VM migration policy best suits their needs. The *effectiveness limit* results provide the answer for our second research question, which is about for how long the protective effect of MTD remains in the system. The third research question intends to include the effects of the VM migration failure probability into the evaluations. Specifically, we noticed that when the system is adopting a policy with 30 minutes between migrations, the probability of attack success stays under 10% even in scenarios with 40% of VM migration failure probability. Finally, our last research question adds the aspect of different attack success rate scenarios. For that one, we noticed that the policy *30 minutes* is also able to reduce the probability of attack success by more than 90% even in all scenarios of different attack success rates.

The selected input parameters are limited to a specific hypothetical scenario. In real-world situations, their values vary according to each IT system environment. Thus, the modification of the parameters is not an easy task, as it requires specific knowledge and a toolset to reproduce the proposed model. In the following chapter, we present *PyMTDEvaluator*, a tool derived from the model to ease the analysis and comparison of multiple MTD scenarios. *PyMTDEvaluator* acts as an interface for the proposed model.

Chapter 8

PyMTDEvaluator: A Tool for Time-based MTD against Non-persistent Attacks

This chapter presents *PyMTDEvaluator*, a tool for evaluating the effectiveness of time-based MTD policies against non-persistent threats. As there are many possible non-persistent attacks, in an attempt to bring a relevant context for the design of *PyMTDEvaluator*, we focus on *availability attacks* (e.g., Denial of Service - DoS, resource starvation attacks) as the specific threat of interest. However, we emphasize that the *PyMTDEvaluator* parameters can be adapted to represent other non-persistent threat scenarios. *PyMTDEvaluator* is based on simulation runs of the model presented in Chapter 7 and offers a user-friendly interface where it is possible to analyze and compare MTD policies with different parameters. *PyMTDEvaluator* provides results such as *probability of attack success*, *availability*, *system capacity*, *annual downtime*, among other relevant information for supporting MTD design decision making. The tool allows analyzing and comparing several scenarios in the same evaluation, thus enabling the study of the pros and cons of different MTD deployment alternatives. *PyMTDEvaluator* aims to be part of the toolset that cloud managers use when designing time-based MTD policies against *availability attacks*. It is also valuable for sensitivity analysis of MTD-enabled system parameters.

The implementation of *PyMTDEvaluator* consisted of three main steps: i) experimental investigation of time-based MTD effectiveness against *availability attacks* (presented in Chapter 3); ii) SPN model design based on the insights obtained from experimentation (see Chapter 7); and iii) development of a simulator for the SPN model. To demonstrate the tool, we present a use case where we analyze VM migration scheduling as MTD against resource starvation attacks. We also compared the *PyMTDEvaluator* results with the results obtained directly from the model computation. This comparison highlights that the *PyMTDEvaluator* results are similar to the outputs of the model. Therefore, *PyMTDEvaluator* tool could be considered as a interface to the proposed model *PyMTDEvaluator* comprehensive result set provides valuable insights for the MTD scheduling selection as, for example, which MTD alternative imposes less than annual downtime.

The *PyMTDEvaluator* tool covers only the MTD deployment against *non-persistent* attacks. Up to this moment, the empirical observation of the system behavior under a *persistent* attack is still pending. Therefore, in an attempt to deliver a more well-rounded tool, we focus on the aspects (i.e., non-persistent attacks) observed in the empirical study and in the analytical modeling. Finally, we highlight that the simulation scripts for the *persistent* attack scenario (used for model validation in Chapter 6) are in a refactoring stage aiming at enabling the inclusion of user parameters through a graphical interface.

Unlike previous works that propose automation frameworks for MTD [Enoch et al., 2020; Alavizadeh et al., 2019a], *PyMTDEvaluator* aims at providing means for MTD scheduling evaluation and analysis. To the best of our knowledge, this is the first tool to provide an evaluation of time-based MTD against *availability attacks*. Users can feed it with inputs to obtain information to answer questions, such as: *what MTD frequency can maintain the probability of attack success under 50% in the first 4 hours of attack? What MTD policy imposes less than two hours of annual downtime? Or, during an attack, is the system able to deliver at least 40% of its capacity to benign users?*

In summary, we highlight the following aspects regarding the current chapter, which is **adapted from Torquato et al. [2021b]**:

- *PyMTDEvaluator* appears in a relevant context of Moving Target Defense against non-persistent threats, where evaluation methods are needed.
- *PyMTDEvaluator* provides a broad set of results (e.g., probability of attack success, transient availability, steady-state availability, system capacity, among other metrics) through a user-friendly interface.
- *PyMTDEvaluator* eases the MTD policies comparison by presenting the plots for all the computed metrics and additional CSV and PDF files.
- *PyMTDEvaluator* is open-source: https://github.com/matheustor4/pymtdevaluator.
- Cybersecurity managers and researchers are the target audience of *PyMTDEvaluator*. However, the tool can be repurposed for other scenarios with time-based actions against a cumulative effect. An interesting example comes from reliability engineering with time-based software rejuvenation [Torquato et al., 2020a].

The rest of this chapter is organized as follows. Section 8.1 presents the details of *PyMTDEvaluator* implementation. Section 8.2 presents validation results. Section 8.3 describes a hypothetical use case for *PyMTDEvaluator*. Section 8.4 presents threats for validity and limitations. Finally, Section 8.5 presents the chapter conclusions.

8.1 *PyMTDEvaluator* Implementation

Figure 8.1 presents the component diagram of the *PyMTDEvaluator* tool. It has been developed in Python 3.7 and includes four modules: i) *User interface*, ii) *Steady-state evaluator*, iii) *Transient evaluator*, and iV) *Plot generator*.

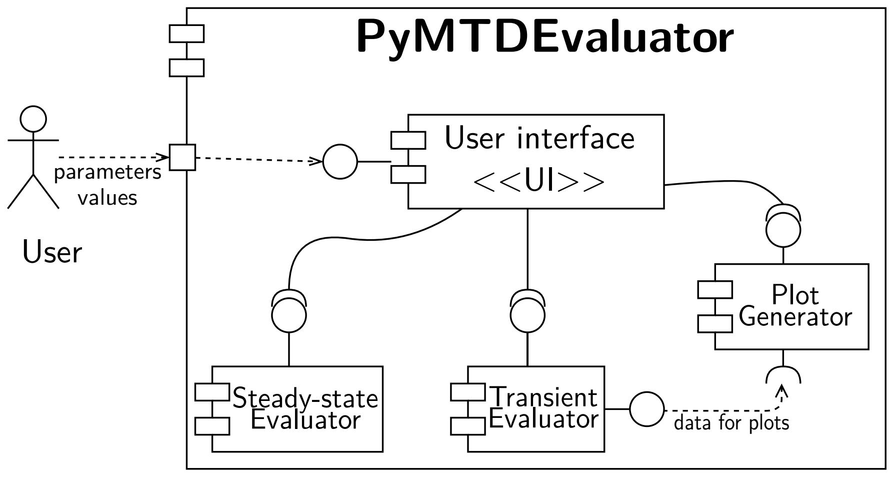


Figure 8.1: PyMTDEvaluator component diagram

8.1.1 User Interface

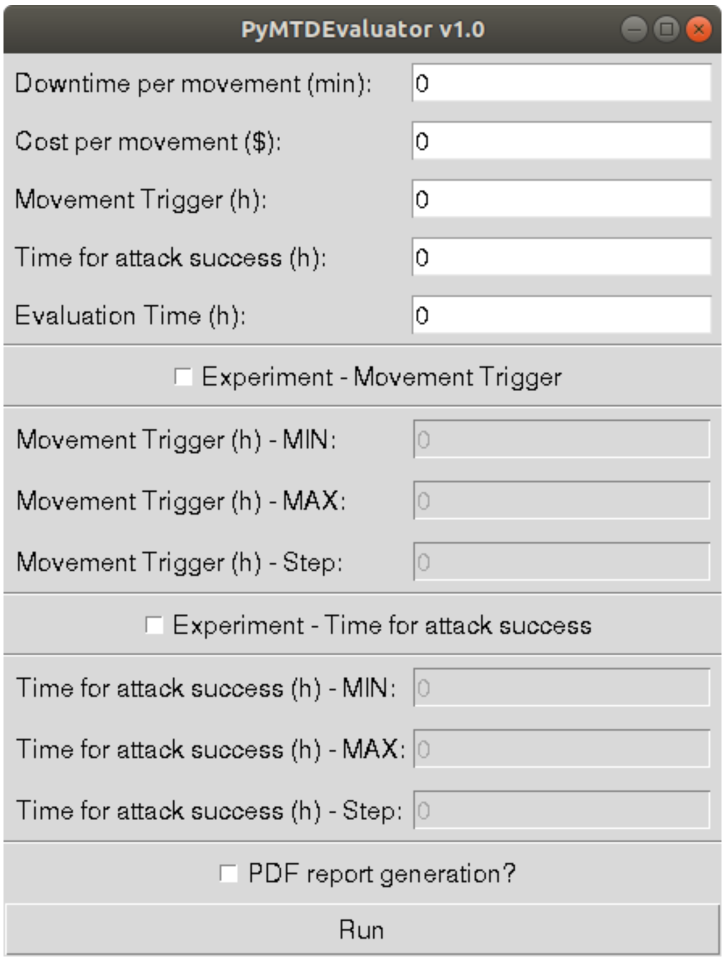
This module uses the Python tkinter library[1] to implement the *PyMTDEvaluator* graphical interface (see Figure 8.2) via which the users will input their parameters. The input parameters are `downtime per movement` - the expected system downtime during the MTD movement; `cost per movement` - the monetary cost (if any) related to each MTD movement action; `movement trigger` - the time between MTD movements; `time for attack success` - the expected time for attack success without MTD defense; and `evaluation time` - the time target (duration) for the simulation runs.

The *PyMTDEvaluator* interface allows setting the computation of multiple evaluations on a single run. Using the `Experiment` *checkboxes*, the user is able to set up the desired variation for the `movement trigger` and/or `time for attack success` parameters. For example, when selecting *checkbox* `Experiment - Movement Trigger`, the user can set minimum and maximum values (and the step) for a sensitivity analysis [Mainkar et al., 1993] of the `Movement Trigger` variable. This feature allows the comparison of multiple MTD deployment alternatives. The same idea applies to the `time for attack success` parameter.

In *PyMTDEvaluator*, a *scenario* is the set of results of a single run. Besides the computation of multiple evaluations in a single *scenario* (by varying the parameters mentioned above), the tool also allows the computation of multiple *scenarios*. In practice, to conduct a multiple *scenario* evaluation, the user must keep the *PyMTDEvaluator* main window open after the given *scenario* run. Then, the user will be able to re-fill the input parameters with the next scenario's parameters and run the simulation again. *PyMTDEvaluator* final output merges all the results from all the *scenarios*.

Using the Reportlab[2] library, *PyMTDEvaluator* compiles its output in a PDF file (by checking the *checkbox* `PDF report generation`). The generated PDF report

[1]https://docs.python.org/3/library/tkinter.html
[2]https://pypi.org/project/reportlab/



Downtime per movement (min): 0
Cost per movement ($): 0
Movement Trigger (h): 0
Time for attack success (h): 0
Evaluation Time (h): 0

☐ Experiment - Movement Trigger

Movement Trigger (h) - MIN: 0
Movement Trigger (h) - MAX: 0
Movement Trigger (h) - Step: 0

☐ Experiment - Time for attack success

Time for attack success (h) - MIN: 0
Time for attack success (h) - MAX: 0
Time for attack success (h) - Step: 0

☐ PDF report generation?

Figure 8.2: PyMTDEvaluator graphical interface

includes the results of the current and all the previous *scenarios* (See Appendix D).

8.1.2 Steady-state Evaluator

This module computes the steady-state availability considering the MTD movement schedule. In principle, the MTD deployment should not impose significant long-term system downtime. With the *Steady-state evaluator* output, the user can compare different MTD policies and select the one that provides the desired balance between security protection and system availability.

Steady-state evaluator module conducts a steady-state simulation of the SPN model using the SimPy framework[3]. For designing the *Steady-state evaluator* simulation and the *Transient evaluator* simulation, we have to implement all the SPN model transition firings as events. We use variables to keep the SPN state and then compute the *PyMTDEvaluator* metrics. NumPy[4] and Random[5] libraries are used to calculate the metrics and to generate the random variables,

[3]https://simpy.readthedocs.io/
[4]https://numpy.org/
[5]https://docs.python.org/3/library/random.html

respectively.

In the *Steady-state evaluator* module simulation, the goal is to achieve the SPN model steady-state. Therefore, contrarily to the *Transient evaluator* simulation, which has a specific evaluation time, the *Steady-state evaluator* keeps the evaluation running until the SPN model results stabilize. We use two variables to verify the SPN results stabilization: i) *warm-up time*: lower threshold time for the simulation - the *Steady-state evaluator* only starts to search for the steady-state after the *warm-up time*; and ii) *batch size*: number of consecutive events in which the results have to obey to the desired *precision*. The *precision* is the difference between the results before and after an event occurrence. After comparing the results with the TimeNET tool Zimmermann [2017], we defined the variables as: *warm-up time* $= 2000h$, *batch size* $= 90$, and *precision* $= 10e^{-4}$. This means that we consider that the SPN model results are steady-state if, from 2000 hours of simulation and after a continuing series of 90 events, the availability results do not vary beyond $10e^{-4}$. Extensive testing was required because using short values for *warm-up time* or *batch size* may lead to premature steady-state convergence, while using larger values may significantly prolong the search for the steady-state. These variable values are crucial for the *PyMTDEvaluator* development, as they support the correct execution of the evaluations. Nevertheless, they do not have a direct relation to the model evaluation parameters (as presented in Section 8.3).

8.1.3 Transient Evaluator

The simulation runs of the *Transient evaluator* have a specific time target defined by the `Evaluation time` user input. The goal is to compute the metrics curve from 0h (starting point) to the `Evaluation time`. The first step of the evaluation is to slice the `Evaluation time` using a one-hour step. To provide the necessary input for confidence interval calculation, the *Transient evaluator* runs a set of one thousand simulation runs for each slice. Besides storing the data in local NumPy arrays, the module also writes the results in CSV files.

The *Transient evaluator* computes the *PyMTDEvaluator* main metrics: *probability of attack success* - represents the probability of attack success in a specific point in time, *transient availability* - show the system expected status (i.e., running or down) in a specific point in time, *accumulated cost ($)* - the monetary cost expenditure due to MTD movements during the selected `Evaluation time`, and *system capacity* - measures how much service is delivered in a given point in time. As the considered attack here is an availability attack, the progress of attack phases impacts the capacity of delivering the correct service to benign users. Besides that, we also have a metric named *Expected threshold*, which is an MTD effectiveness metric that represents the first point in time where the probability of attack success reached its maximum value.

In some situations, the user may be interested in a more detailed view of the behavior of the system. To address this issue, we include a separated simulation run (henceforth, *example run*) in the *Transient evaluator* module. In this *example run*, the *Transient evaluator* creates a file with the trace of the simulated events and computes the availability, system capacity, and the survival time (i.e., time that the system remains available in the selected `Evaluation time` for the *example run*). Although this comes from a single evaluation, the *example run* results are useful to highlight a possible system behavior within the user's desired parameters.

8.1.4 Plot Generator

The *Plot generator* module is responsible for presenting the *PyMTDEvaluator* results to the user. The module receives the data from the *Transient evaluator* module and uses the Matplotlib[6] for data plotting. Besides that, the module presents a window with a summary of the results for each evaluation. The users can also plot the graphics on their preferred software using the CSV files generated. Examples of the *PyMTDEvaluator* output plots and window of results are presented in Section 8.3.

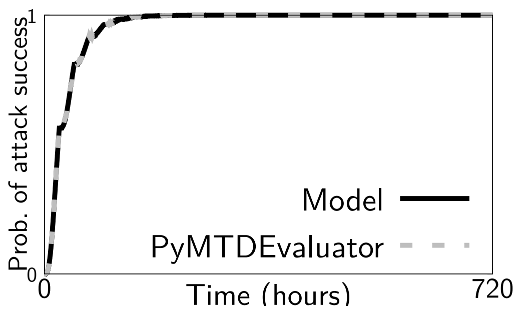
8.2 Validation against Model Results

To validate the *PyMTDEvaluator* results and assure its correctness, we compared the TimeNET tool analysis results with the *PyMTDEvaluator* simulation results (see Figure 8.3). We performed an extensive set of these validation experiments adopting a wide range of parameters. In all the observed results, *PyMTDEvaluator* output is very close to the model results. For the sake of illustration, we provide an example of this validation experiment below.

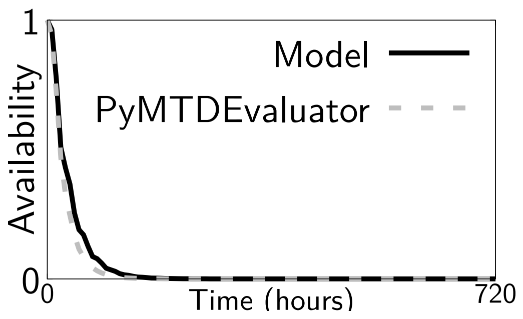
The main parameters for this validation experiment are: four seconds of downtime per movement, 24 hours between MTD movements, and 24 hours as the expected time for attack success. The results consider the first month of the system under attack (i.e., `Evaluation time` $= 720h$). Note that these parameters are just for validation and may not represent a real-scenario situation.

Figure 8.3 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 (Figure 8.3(a)) and system capacity (Figure 8.3(c)) are compatible with the *PyMTDEvaluator* results. The availability results are also very close, but they present a little variance in the curve. The difference in the availability curves may be due to the simulation stoppage method when the system reaches the *availability attack* success. Specifically, we include some break-point statements to interrupt the simulation run upon the triggering of the attack completion event. We had to implement this stoppage method in the *Transient evaluator* module to reduce *PyMTDEvaluator* computing time.

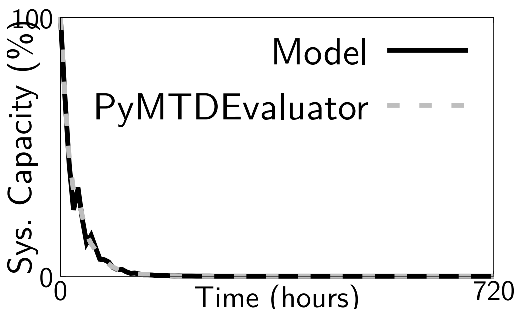
[6] https://matplotlib.org/



(a) Probability of attack success



(b) Availability



(c) System Capacity (%)

Figure 8.3: PyMTDEvaluator results validation against SPN model results

8.3 Use case

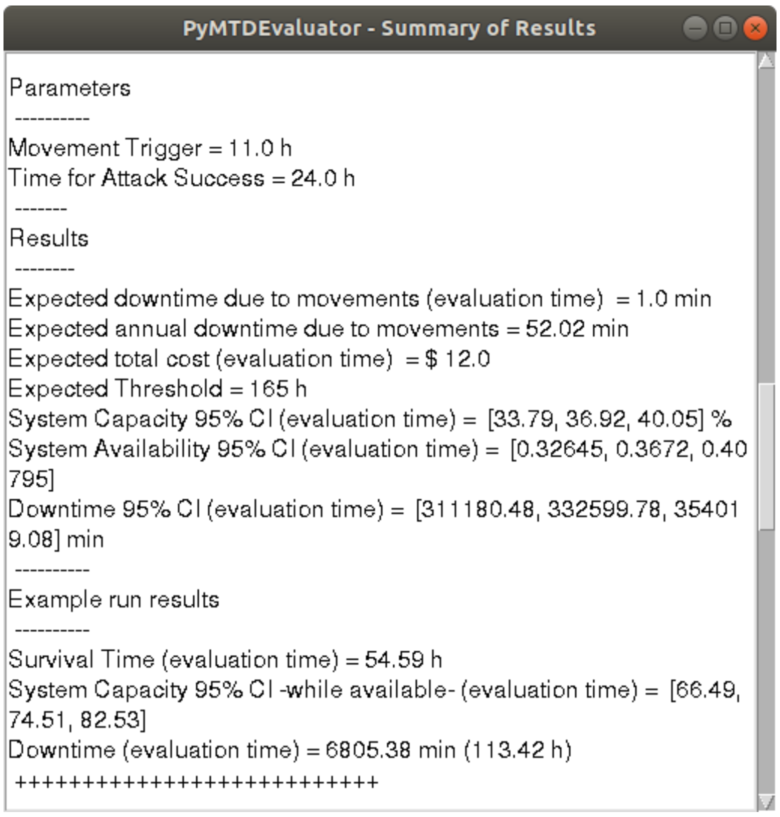
To illustrate the possible *PyMTDEvaluator* use cases, we provide an example considering an MTD-enabled virtualized system under a resource starvation attack. In this example, the attacker has the authorization to control a VM, like a legitimate client of a public cloud (the cloud provider cannot inspect the internal content of the VM of that client due to privacy rights). Then, the attacker tries to overload the underlying hypervisor. Note that, in this case, the cloud provider should accept the incoming attack because the client has permission to run any software upon the VM. The client may also be conducting an unintended (naive) attack by running specific workloads. If the resource starvation attack succeeds, the physical machine host and co-hosted VMs will become unavailable.

Once the attacker starts to overload the system, any resource monitoring tool can detect the attack source. Therefore, it is quite straightforward just to shut down the VM of the attacker. However, there are situations where the cloud provider cannot shut down the VMs of the clients (e.g., legit clients running a naive DoS, a user with privileged access and enhanced privacy rights). Moreover, there are subtle resource starvation attacks that may leverage, for example, hypervisor software aging bugs [Torquato et al., 2018a]. Indeed, we observed that the *unalignAttk* (i.e., memory DoS attack presented in Chapter 3) produces a negligible impact on the CPU and RAM utilization. However, the *unalignAttk* (i.e., an infinite loop of `LOCK` signals generation in the memory) causes a significant impact on the co-resident VMs. In these cases, cloud providers may apply heuristics to select what VMs should follow an MTD deployment.

We arbitrarily set the parameters for the evaluation as follows: `Time for attack success`: 24 hours; `downtime per movement` - four seconds (as suggested in Chapter 4); and `cost per movement` - 0.8$ (guesstimate based on the VM migration associated costs of downtime, performance and power consumption [Voorsluys et al., 2009; Huang et al., 2011]).

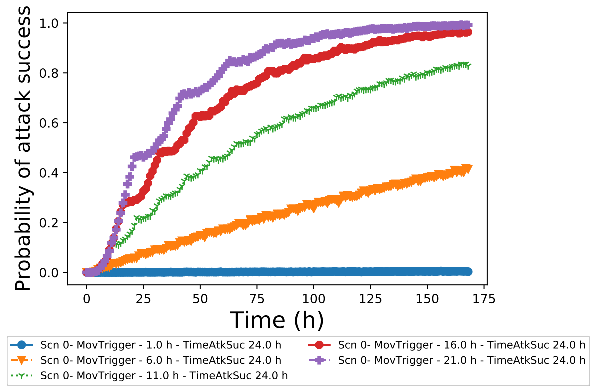
The evaluation aims to find a VM migration trigger that reduces the probability of attack success while maintaining the annual downtime due to VM migration below one hour. To find the intended VM migration trigger, we range the VM migration trigger from 1 to 21 hours with a step of 5 hours. We set up the `Evaluation time` as 168 hours (i.e., one week).

The *PyMTDEvaluator* results suggest that the VM migration trigger of 11 hours provides the best security results among the tested ones, maintaining annual VM migration accumulated downtime below one hour. Figure 8.4 presents the *PyMTDEvaluator* Summary of the Results window. It is worth highlighting that this window also presents the results for the other MTD policies tested.

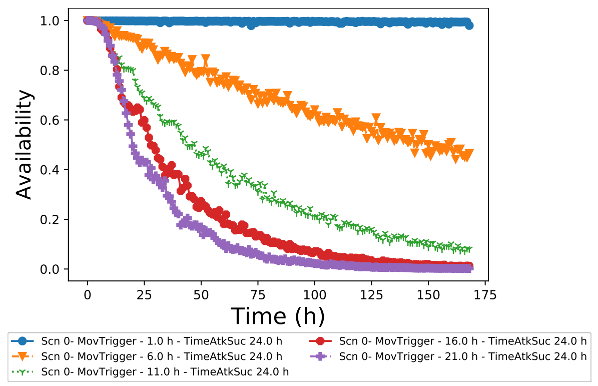


```
Parameters
----------
Movement Trigger = 11.0 h
Time for Attack Success = 24.0 h
-------
Results
--------
Expected downtime due to movements (evaluation time)  = 1.0 min
Expected annual downtime due to movements = 52.02 min
Expected total cost (evaluation time)  = $ 12.0
Expected Threshold = 165 h
System Capacity 95% CI (evaluation time) =  [33.79, 36.92, 40.05] %
System Availability 95% CI (evaluation time) =  [0.32645, 0.3672, 0.40
795]
Downtime 95% CI (evaluation time) =  [311180.48, 332599.78, 35401
9.08] min
----------
Example run results
----------
Survival Time (evaluation time) = 54.59 h
System Capacity 95% CI -while available- (evaluation time) =  [66.49,
74.51, 82.53]
Downtime (evaluation time) = 6805.38 min (113.42 h)
+++++++++++++++++++++++++++
```

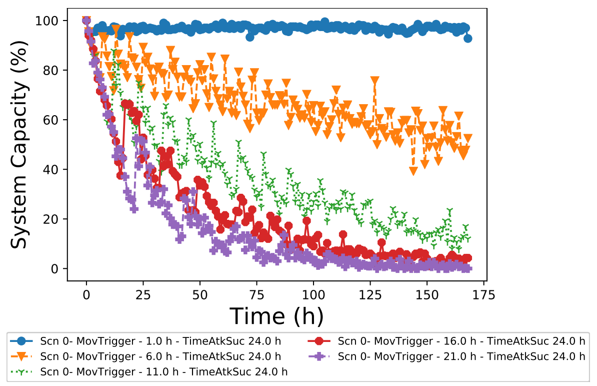
Figure 8.4: PyMTDEvaluator - Summary of results window



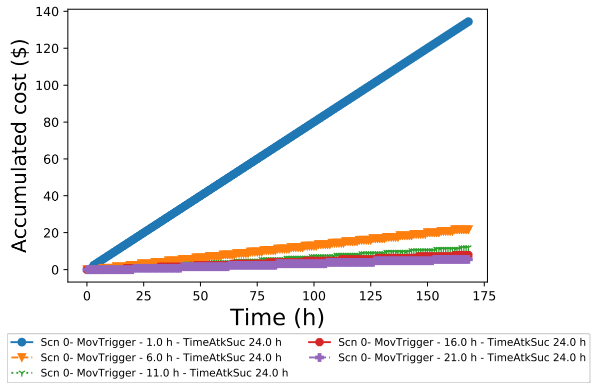
(a) Probability of attack success



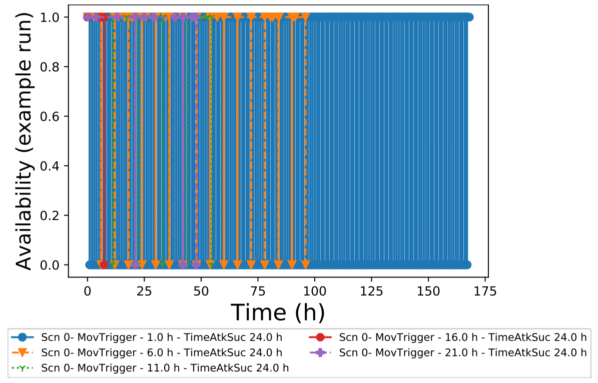
(b) Availability



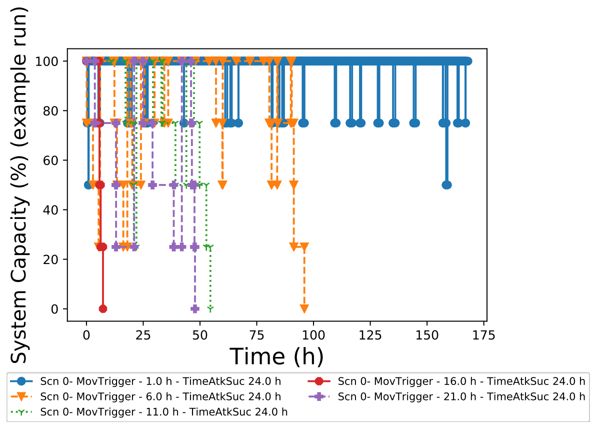
(c) System Capacity (%)



(d) Accumulated cost ($)



(e) Availability - example run



(f) System Capacity (%) - example run

Figure 8.5: PyMTDEvaluator results for the evaluation scenario

In the studied scenario, the VM migration scheduling imposes only about one minute of system downtime. VM migration-related annual downtime is about 52 minutes (see the lines below `Results` in the Figure 8.4). The accumulated monetary cost is 12 \$. The *expected threshold* shows that the MTD deployment becomes ineffective after 165 hours. Finally, we notice low system capacity and availability results. Due to resource consumption and the increasing probability of attack success, the MTD deployment can maintain only about 40% of system capacity during the `evaluation time`. Remember that the attack success turns the system unavailable, meaning 0% of capacity. System availability levels are also low, maintaining only about 36% of availability in the proposed scenario. The results of the *Example run* show that the system survived the attack for 54 hours, maintaining about 75% of its capacity, and that the system may experience more than four days of downtime.

Using the complete set of results provided by the *PyMTDEvaluator* (see Figure 8.5[7]), we can compare the metrics curves for each VM migration trigger. Hourly VM migrations MTD policy provides the best overall results for security, availability, and capacity. However, it imposes a higher monetary cost. As we are considering *availability attacks* in the threat model, we see a decreasing behavior in the availability curve when the probability of attack success increases. *Example run* results highlight that due to the frequent VM migrations, hourly VM migration MTD policy imposes severe system availability oscillation, as each migration produces system downtime.

The list of conclusions presented above is non-exhaustive, meaning that maybe there are other insights that could be extracted from the results. We hope that this use case may bring some input into the *PyMTDEvaluator* usefulness. In the context of this chapter, we applied *PyMTDEvaluator* to study a system under an *availability attack*. However, it is possible to extend its usage for evaluating systems in which the MTD action can clear the attack progress.

8.4 Threats to Validity and Limitations

Model threats to validity

As *PyMTDEvaluator* development relies on the model present in Section 7.2, it inherits the corresponding threats to validity and limitations. Namely, it is possible to highlight the following: i) lacking a complete model validation against experimental results and ii) neglecting the performance impact. These limitations are properly discussed in Section 7.4. However, *PyMTDEvaluator* goal is not to completely surpass these limitations. Instead, we aim to provide an easy-to-use tool. Therefore, even the audience without SPN background can benefit from the model computation.

[7] The complete confidence intervals are presented in the *PyMTDEvaluator* CSV output files

Lack of flexibility in the model design

PyMTDEvaluator acts as a interface for the proposed model. Therefore, in the current version, it lacks flexibility in the model design. This way, the implementation strictly executes the simulation of the model, and it lacks the feature of deactivating some transitions or applying redesign to the model layout. Furthermore, *PyMTDEvaluator* still adopts only deterministic and exponential transitions. Note that we decided to keep *PyMTDEvaluator* closely related to the model, as the model design already relies on an experimental background. Allowing flexibility of model design without proper verification was a path that we wanted to avoid. Finally, the inclusion of alternative distributions seems to be a matter of specific re-implementation of some of the methods of *PyMTDEvaluator*. This inclusion may be achievable for the next releases of the tool.

Performance issues

At the moment of this thesis writing, the performance evaluation of *PyMTDEvaluator* is yet to be done. We noticed that, as expected, scenarios with a large number of accumulated events tend to take longer to compute. *PyMTDEvaluator* features a progress bar (presented at the terminal), which indicates how much of the scenario is up to completion. Note that *PyMTDEvaluator* is intended to be a *design*-time tool, not a *runtime* tool. As these scenarios may be evaluated before MTD deployment, the system managers will be able to analyze the potential benefits and drawbacks of particular time-based VM migration policies.

8.5 Summary

This chapter presented *PyMTDEvaluator* that provides an easy-to-use and flexible approach for evaluating time-based MTD against non-persistent attacks. *PyMTDEvaluator* comprehensive results include relevant metrics to support the MTD design decision-making process. The user is thus able to analyze and compare different time-based MTD deployments verifying the levels of *probability of attack success*, *availability*, *system capacity*, *monetary cost of MTD deployment*, among other metrics.

PyMTDEvaluator design relies on the Petri Net-based model from Chapter 7, which is capable of representing the main events in the scenario as MTD action, system downtime due to MTD movement, and attack progress. The main assumptions of this Petri Net were drawn from empirical observations reported in Chapter 3. *PyMTDEvaluator* can cluster results from multiple evaluations, providing means for complex scenario analysis. The presented use case serves as an example of *PyMTDEvaluator* features. *PyMTDEvaluator* users may leverage the tool's flexibility to study scenarios beyond the one presented.

As a final contribution to our thesis, in the next chapter, we propose a study combining our previous models of Moving Target Defense and Software Rejuvenation based on VM migration. There, we intend to investigate the tradeoffs between security and availability while adopting such an approach.

Chapter 9

Modeling of Time-Based VM Migration as MTD and Rejuvenation

Software rejuvenation is particularly important in cloud computing environments, as these frequently suffer from software aging issues [Araujo et al., 2011; Machida et al., 2012; Matos et al., 2012a]. In this scenario, it is possible to use VM migration as support for software rejuvenation. The idea is to move VMs away from a host under software aging before taking rejuvenation actions. VM migration is also a MTD technique [Torquato and Vieira, 2020], Being quite effective against *host-based* attacks, where the attack targets the underlying physical host. To enhance the MTD protection, migrations can be done among hosts holding different versions of the *hypervisor*. This means that the system is using *diversity-based* MTD [Hong and Kim, 2015].

Due to the multipurpose applicability of VM migration (i.e., for software rejuvenation and MTD), its deployment could be analyzed from both the availability and security perspectives. However, the current literature lacks previous studies in this area. Most papers focus either on VM migration as support for rejuvenation [Machida et al., 2010] or MTD [Torquato et al., 2021a].

This chapter proposes a set of SPNs to analyze the availability and security of a system applying multipurpose time-based VM migration. The considered technique has the following characteristics: i) applies a regular time interval between VM migration triggering; ii) follows the *live migration* approach [Clark et al., 2005]; and iii) the migrations are made between different hypervisors to enforce *diversity-based* MTD deployment. The proposed models are nearly the same as in the previous chapters. However, we use a slightly different approach in the software rejuvenation modeling, where we adopt a hypoexponential distribution to represent software aging instead of an Erlang subnet. Note that the hypoexponential approach is widely used in the same context [Machida et al., 2013].

The following research questions guided this research:

RQ_1: What is the VM migration trigger interval that maximizes the steady-state system availability?

RQ_2: What is the MTD protection achieved while using a VM migration trigger interval for availability maximization?

RQ_3: Assuming time-based VM migration as support for rejuvenation and MTD, what are the tradeoffs between availability and security when selecting a specific VM migration trigger interval?

To address these questions, we consider three case studies, focusing on MTD protection, system availability, and availability vs. security tradeoffs, and considering four major *hypervisors* for server virtualization: Xen, KVM, ESXi, and Hyper-V. The last case study features an illustrative Multi-Criteria Decision Making analysis as an example of how to explore the results obtained. In general, our results show substantial availability improvement while using VM migration for software rejuvenation and highlight the need to ease lock-in platforms and for heterogeneous systems to enable *diversity-based* MTD techniques.

To the best of our knowledge, this is the first work evaluating VM migration as a multipurpose approach for MTD and rejuvenation. From an engineering perspective, we conclude that the heterogeneous *hypervisor* migration could provide substantial protection against specific attacks and believe that the deployment of *diversity* MTD should be encouraged to add a security layer to the virtualized environment. **This chapter is adapted from Torquato et al. [2022b]**.

The rest of this chapter is organized as follows. Section 9.1 presents the multipurpose VM migration workflow and evaluation approach. Section 9.2 shows the proposed model. Section 9.3 provides the results of the analysis in the form of case studies. Section 9.4 presents threats to the validity and limitations of our work. Conclusions are presented in Section 9.5.

9.1 Approach and Assumptions

We consider a live migration-enabled [Clark et al., 2005] virtualized system with its three classical main components: *Primary Host* (i.e., *Main Node* of the original architecture in Chapter 3) - physical machine that hosts the Virtual Machines, *Target Host*(i.e., *Standby Node* of the original architecture in Chapter 3) - physical machine selected to receive the VMs migration; and *VMs* - which run the client applications. It is possible to migrate the VMs from *Primary Host* to the *Target Host* and back. We consider this architecture appropriate for the scope of this chapter because it is the baseline of virtualized environments. It appears in various systems, from small-sized virtualized environments to large-scale cloud computing.

The environment is ready for VM migration between heterogeneous *hypervisors* [Ashino and Nakae, 2012; Kargatzis et al., 2017; Awasthi and Gupta, 2016; Raj et al., 2020]. For MTD purposes, the VMs always arrive in a different *hypervisor* variant after migration. The VM migration management adopts a *consolidation* approach, meaning all the VMs should be placed in a single physical host. The *consolidation* approach simplifies the software rejuvenation deployment, as the *Primary Host* is freed from the VMs execution after migration. The system triggers VM migration by observing a regular time interval (i.e., time-based VM migration).

We apply time-based VM migration to counteract *hypervisor* software aging on the *Primary Host* [Matos et al., 2012a; Machida et al., 2012]. The approach consists of moving VMs from the to the *Target Host*. After VMs migration completion, the *Target Host* assumes the VMs execution (i.e., assumes the role of *Primary Host*). Then, the previous *Primary Host* (i.e., VM migration source) passes through a *hypervisor* rejuvenation. Finally, it turns into the *Target Host* for the next migration.

As for the **attack and defense model**, we assume that the attacker is already in the system and starts to conduct the attack from the beginning of the evaluation. The attacker controls a set of *VMs* running in the *Primary Host*. The attack target is the underlying *hypervisor*. We assume that the attacker adopts a *persistent* tactic (as in Chapter 6). Thus, we consider that once the attacker identifies the *hypervisor* variant, it is possible to resume the attack regardless of the software rejuvenation performed in the system. The attack has two stages: i) *reconnaissance* - attacker tries to identify the specific *hypervisor* variant; and ii) *attack* - after hypervisor identification, the attacker runs specific malicious actions against it.

Our MTD approach consists of continuously moving the VM of the attacker among different *hypervisor* variants. Once the attacker arrives in a different *hypervisor* variant, the attack progress in the previous one is stopped. The attacker must conduct the *reconnaissance* phase every time the VMs arrive in a different *hypervisor*. However, once the VMs return to previously visited *hypervisor*, the attacker can resume the attack after the *reconnaissance* phase (i.e., the attacker keeps knowledge). The *hypervisor* variant modification (which occurs after every migration) follows a circular approach, in which the configuration goes from the current variant to the next one and returns to the first after passing through the last variant available.

Unless we have frequent VMs migration to keep the attacker in the *reconnaissance* phase in all *hypervisor* variants, the attacker will eventually compromise the system. Therefore, the proposed MTD approach cannot avoid attack success in the long run. Instead, the MTD goal is to extend the resistance against *host-based* attacks. Hopefully, this increased resistance may match the expected security levels and may allow the activation of additional defenses to enhance system security (e.g., provide more time for attack detection from Intrusion Detection Systems).

Figure 9.1 summarizes the proposed approach for VM migration as support for software rejuvenation and MTD. At the initial stage (S1), the system runs without aging accumulation, and the attacker starts (or continues) the attack. At the second stage (S2), the system starts to accumulate software aging effects, and the attacker passes through the *reconnaissance* phase. There are two possibilities from here: i) VM migration does not reach the triggering interval - then the system may fail due to software aging, or the attacker may reach the goal of compromising the *hypervisor*, or ii) VM migration reaches the triggering interval, which leads us to stage S3. In the stage S3, we apply *diversity* (i.e., change the *hypervisor* variant) in the *Target Host* before receiving the VMs migration. Then, after

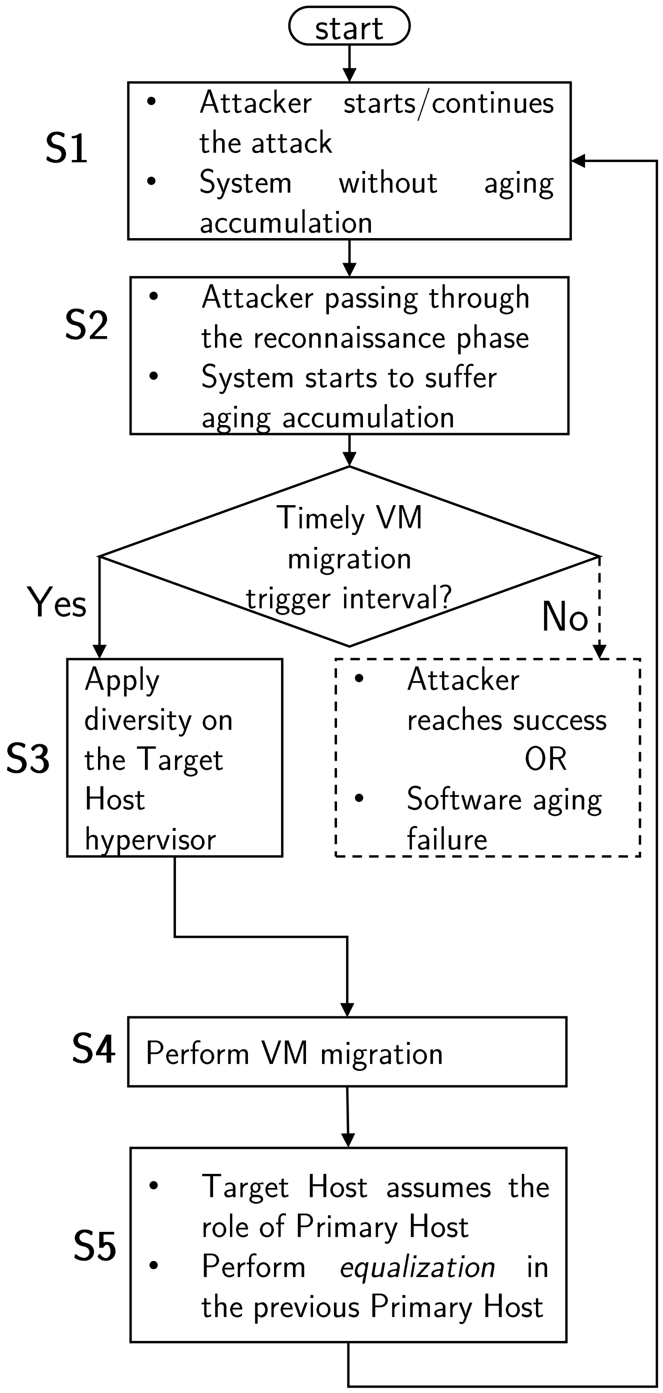

start
S1
Attacker starts/continues the attack
System without aging accumulation
S2
Attacker passing through the reconnaissance phase
System starts to suffer aging accumulation
Timely VM migration trigger interval?
Yes
No
S3
Apply diversity on the Target Host hypervisor
Attacker reaches success OR
Software aging failure
S4
Perform VM migration
S5
Target Host assumes the role of Primary Host
Perform equalization in the previous Primary Host

Figure 9.1: VM migration supporting software rejuvenation and diversity MTD - workflow

diversity completion, the migration moves the VMs from the *Primary Host* to the *Target Host* (stage S4). After VMs migration completion, we apply *equalization* in the *Primary Host*, and the *Target Host* assumes the role of executing the VMs. *Equalization* consists of software rejuvenation actions and homogenization of the *hypervisor* variants.

We decided to apply *equalization* after VM migration due to the costs associated with interoperability. From an engineering perspective, keeping the *diversity* throughout system execution is not trivial, as the cloud components are constantly communicating. Therefore, an interoperability issue may lead the entire system to fail or cause errors in the cloud management service. This way, we decided to maintain only sporadic interoperability actions (i.e., VM migration between different *hypervisors*) to avoid higher deployment and management costs.

9.2 Model

The proposed model, presented in Figure 9.2, has four sub-models: a) Clock Model - representing the behavior of the component responsible for triggering the VM migration in the environment; b) Control Model - used as control variables to guide the token flow through guard functions[1]; c) Availability Model - which captures the VM migration, non-aging failures and software aging; and d) Moving Target Defense Model - representing the attack progress and MTD effects. The presented Moving Target Defense Model corresponds to systems with only two variants. However, as will be seen later, we studied the MTD behavior while having up to four variants, as we have four major *hypervisors* for server virtualization: Xen, KVM, ESXi, and Hyper-V.

The sub-models interaction relies on guard functions (Table 9.1). Using guard functions, we avoid excessive transitions and control places, which may impair the model readability. We embed the guard functions only in immediate transitions[2] and assign different priorities to enforce the representation of the system workflow presented before.

Table 9.1: Guard functions

Transition	**Function**	**Description**
`startMig`	g1	`(#MigMTD==1) AND (#UP>0) AND (#TargetUP>0)`
`migFail`	g2	`(#UP==0) OR (#TargetUP==0)`
`triggerDiv`	g3	`(#ReadyToMigrate==1) AND (#Equal==1)`
`startDiv`	g4	`(#ReadyToMigrate==1) AND (#TargetWait==1)`
`completeDiv`	g5	`(#TargetUP==1)`
`finishMig`	g6	`(#MigDW==1)`
`restartCycle`	g7	`(#TargetWait==1)`
`v(1,2)v(1,2)Mig(2)`	g8	`(#UP==0)`
`v(1,2)v(1,2)MigFail(2)`	g9	`(#UP==1) AND (#Equal==1)`
`v(1,2)v(1,2)MigSucc(2)`	g10	`(#UP==1) AND (#Equal==0)`

[1]Guard functions express additional conditions for transition firing as boolean expressions
[2]Immediate transitions - thin black rectangles

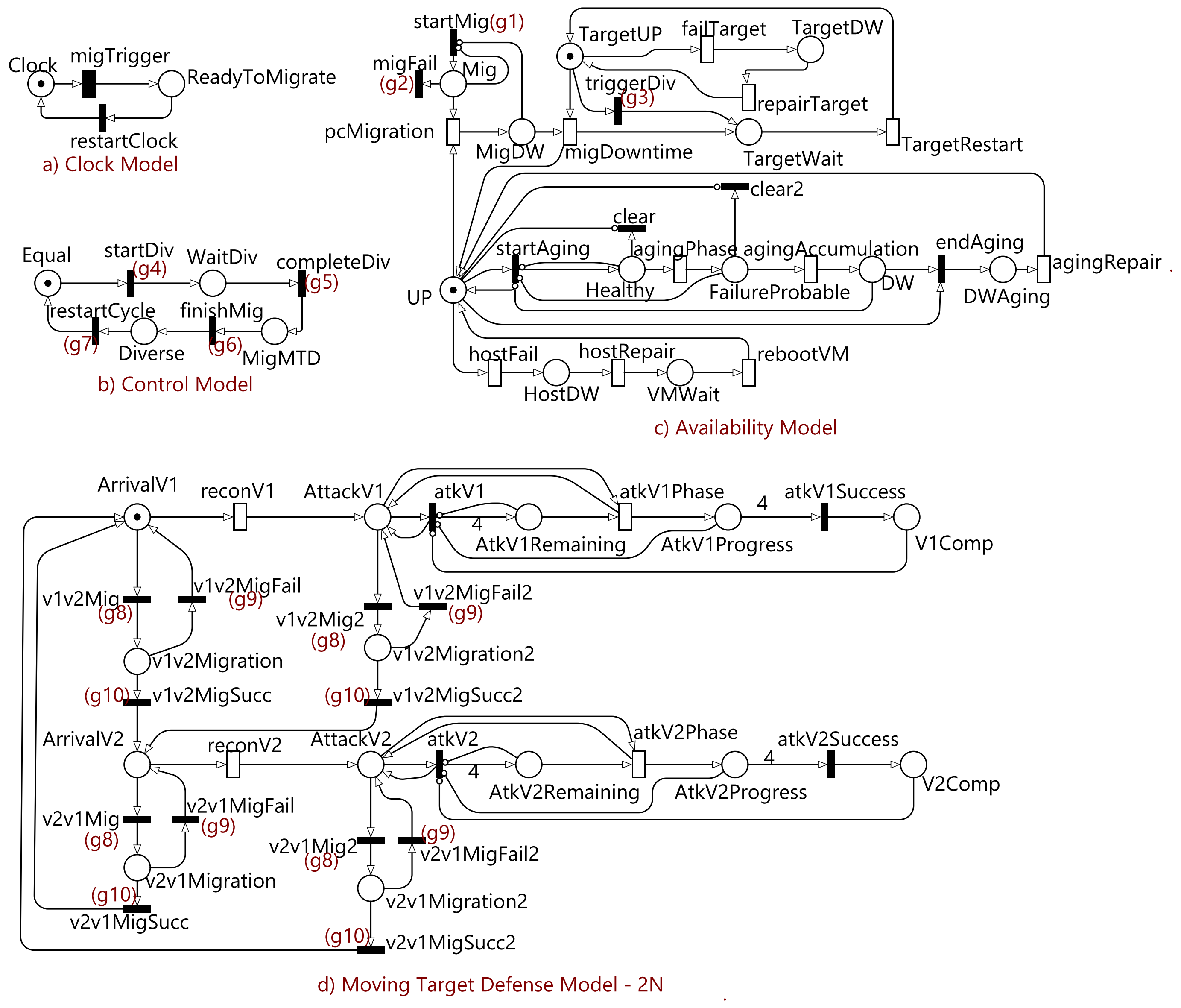


Figure 9.2: Proposed Models

9.2.1 Availability-related Model

The Clock Model has a deterministic transition[3], `migTrigger`, whose delay represents the interval between migrations. Transition[4] `migTrigger` firing moves the token from place[5] `Clock` to place `ReadyToMigrate`, representing that the clock reached the interval for VM migration. The transition `restartClock` firing moves the token back to the place `Clock`, thus restarting the time counting for VMs migration. Transition `restartClock` has low priority among other immediate transitions to assure token presence in place `ReadyToMigrate` for related guard functions `g3` and `g4` activation.

The Control Model guides the VM migration net execution through the associated guard functions. At the initial state, the place `Equal` has one token, representing that the *hypervisor* configuration is homogeneous among the hosts. In this scenario, assuming token presence in the place `ReadyToMigrate`, the net activates guard function `g3`. Let us also assume that the place `TargetUP` has tokens (i.e., the *Target Host* is running). The combination of these conditions triggers transition `triggerDiv`, which moves the token from place `TargetUP` to place `TargetWait`. In this case, the token arrival in place `TargetWait` represents the diversification of the *hypervisor* of *TargetHost*. Token presence in the place `TargetWait` also influences the transition `startDiv` firing.

Transition `TargetRestart` firing moves the token from place `TargetWait` to place `TargetUP`, representing the completion of the *Target Host hypervisor* diversification. The transition `completeDiv` also fires when the token returns to the place `TargetUP`. It deposits a token in the place `MigMTD`, representing that the system is ready to perform VM migration. Guard function `g1` embeds the necessary conditions for starting VM migration. Therefore, transition `startMig` only fires if the physical hosts are available (i.e., token presence in the places `UP` and `TargetUP`).

Transition `startMig` firing deposits a token in the place `Mig`, meaning that the system is under migration. At this point, any physical host failure cancels the migration (i.e., transition `migFail` firing removes the token from place `Mig`). Transition `pcMigration` represents the time for completion of the VM migration pre-copy phase. We adjust the duration of the pre-copy phase according to the current status of the system. Software aging may slow down the pre-copy phase as it affects system performance. We use marking-dependent firing rates[6] to represent this behavior. Note that the pre-copy phase duration may vary depending on the current status.

Transition `pcMigration` firing moves the token from place `Mig` to place `MigDW`. It also collects the token from place `UP`. In this case, the absence of tokens in the place `UP` represents that the system is under downtime related to VM migration. Token presence in the place `MigDW` also triggers transition `finishMig` of the Control

[3]Deterministic transition - thick black rectangle
[4]Transition (i.e., exponential transition) - white rectangle
[5]Place - White circle, Token - small black circle inside the Places.
[6]Marking-dependent firing rates - allow the modification of firing rates upon different tokens distributions in the places.

Model. After the expected downtime related to VM migration (i.e., the delay associated with transition `migDowntime`), the system is running again (i.e., the token returns to the place `UP`). In this case, place `TargetWait` also receives a token, meaning that the source of VM migration will pass through the *equalization* phase (state S5 of the workflow - Fig. 9.1). Finally, the Control Model restarts its cycle upon the firing of transition `restartCycle`.

It is possible to adapt the availability and security modeling of VM migration to alternative approaches (i.e., alternative migration methods). The VM migration-related transitions capture the two main steps of VM migration: copy of memory pages and downtime. Therefore, we can represent other migration techniques through parameter adjustment. In the case of performability (or performance) VM migration modeling, we may modify the model design as the different methods may have additional stages for the migration process.

The sub-net between the transitions `startAging` and `endAging` aims to represent the software aging and rejuvenation behavior. As in works Huang et al. [1995] and Machida et al. [2013], we use a two-phase (i.e., transitions `agingPhase` and `agingAccumulation`) hypoexponential net to represent the increasing failure rate due to software aging. The transitions `clear` and `clear2` remove the token from the places `Healthy` and `FailureProbable` (if any), respectively. These transitions represent the software aging removal due to software rejuvenation. We also consider that the repair from non-aging failures includes software rejuvenation actions. We use inhibitor arcs[7] to trigger transitions `clear` and `clear2` upon the absence of tokens in the place `UP`. In the case of lack of timely rejuvenation, the system faces an aging-related failure (transition `agingAccumulation` firing). Place `DWAging` receives a token upon transition `endAging` firing. Transition `agingRepair` firing, representing the system repair after aging, puts a token in place `UP`.

Transitions `startAging` and `endAging` along with place `DWAging` are just accessories in the modeling. However, we keep them because they isolate the model software aging and rejuvenation behavior. Therefore, it is possible to adjust software aging and rejuvenation modeling with minimal interference in the current model layout.

Primary Host repair after non-aging related failures has two steps: host repair and VMs reboot. We use the transitions `hostFail`, `hostRepair` and `rebootVM`; and the places `HostDW` and `VMWait`, to represent this behavior. The sub-net with the places `TargetUP` and `TargetDW` with transitions `failTarget` and `repairTarget` represent the behavior of non-aging failures and repairs in the *Target Host*.

Transition `hostRepair` refers to repair after a non-aging failure. Originally, this rate came from an RBD model comprising hardware and OS components Melo et al. [2013b]. We decided to turn VM reboot explicit after a non-aging failure because there are repairs after a non-aging failure that may not imply a system reboot. On the other hand, the repair after an aging failure should require a system reboot to ensure rejuvenation. Then, we decided to omit transition `VMreboot` after

[7]Arcs terminating in circles instead of arrowheads

transition `agingRepair`.

9.2.2 Security-related Model

The Moving Target Defense model represents the VM migration in the attack-defense flow. As mentioned Transition `reconV1` firing moves the token from place `ArrivalV1` to `AttackV1`. It means that the attacker passes through the *reconnaissance* phase of the attack and is now at the *attack* phase. As presented in Chapters 6 and 7, we also use a four-phase (transition `atkV1Phase`) Erlang sub-net to represent the increasing success rate of the *attack* phase. The Erlang sub-net has the places `AtkV1Remaining` and `AtkV1Progress` with the transition `atkV1Phase`. Once the token arrives in the place `AttackV1`, the immediate transition `atkV1` deposits four tokens in the place `AtkV1Remaining`. Then, the accumulation of tokens in the `AtkV1Progress` represents the progress of the *attack* phase. The transition `atkV1Success` immediately puts a token in the place `V1Comp` when the place `AtkV1Progress` stores the fourth token. The presence of tokens in the place `V1Comp` represents that the attacker succeeded in compromising the system security.

As mentioned in Section 9.1, the attacker follows a *persistence* tactic. This way, the VM migration can only interrupt the attack progress temporarily. Once the attacker identifies the *hypervisor* variant, it is possible to resume the attack based on the obtained knowledge. The net does not remove the tokens stored in the places related to the *attack phase* upon VM migration triggering. The attack is also interrupted in case of a host failure. Note that it is possible to adapt the models to cover *non-persistent* attacks (i.e., remove the attack progress after each migration) by adding cleaning transitions (like transitions `clear` and `clear2`) in the places of the four-phase Erlang sub-net. However, we emphasize that *non-persistent* attacks are out of the scope of this chapter.

The absence of tokens in place `UP` (meaning system execution interruption) enables the firing of transitions `v1v2Mig` and `v1v2Mig2`. These transitions move the token to the places `v1v2Migration` or `v1v2Migration2` depending on the current status of the attack. If the system interruption is due to VM migration, the transitions `v1v2MigSucc` or `v1v2MigSucc2` fire, depositing a token in the place `ArrivalV2`. Otherwise, depending on whether the system interruption is due to aging or non-aging failures, the transitions `v1v2MigFail` and `v1v2MigFail2` fire, returning the token to its previous place.

The attack-defense behavior in the second *hypervisor* variant follows the same dynamics as the first. We use identical sub-nets to represent attack-defense behavior in each *hypervisor* variant. The same applies to the MTD models for the environments with three and four *hypervisor* variants, which are omitted for simplicity.

9.2.3 Metrics Computation

Our primary metrics of interest are: **Steady-state availability** (A) and **Transient probability of attack success** (PAS). We obtain A by observing the steady-state probability of token presence in the place `UP`. For PAS, we observe the transient probability of token presence in any of the places related to successful attacks (`V1Comp` or `V2Comp`). Finally, we derive secondary metrics in the proposed case studies. *Pointwise Probability of Attack Success* ($PAS(t)$) computes the probability of attack success at a specific point in time (t). *Tolerance Level* (TL) is the estimated point in time where the system reaches a specific probability of attack success.

9.3 Case Studies

We use TimeNet tool [Zimmermann, 2017] for model design and analysis. We consider the parameters in Table 9.2 for the case studies presented below. These values come from the previous chapters, specifically the Chapters 4, 6, and 7. Note that, we needed to estimate some of these values since they are unavailable or are hard to obtain through experimentation. For the sake of simplicity, we assume the same *reconnaissance* and *attack* phase delays for all the available *hypervisor* variants. Moreover, in the lack of *hypervisor*-specific aging parameters, we also use the consolidated values from the works mentioned above in all the *hypervisor* variants.

This section presents three case studies. The first (Section 9.3.1) shows the steady-state availability results, where the evaluation searches for the VM migration trigger that maximizes system availability. The second (Section 9.3.2) provides the MTD evaluation using the VM migration schedule obtained in the first case study. The last (Section 9.3.3) considers the tradeoffs between availability and security while using VM migration as support for MTD and rejuvenation.

9.3.1 CS #1 - Availability-aware VM Migration Trigger

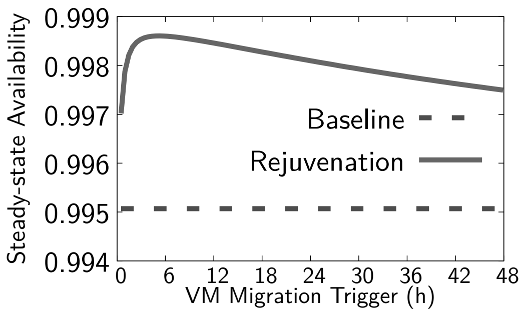
This case study aims to answer RQ_1: *What is the VM migration trigger interval that maximizes the steady-state system availability?* Frequent migrations may increase system downtime because of the interruptions associated with each migration. Alternatively, less frequent migrations increase the chance of software aging failure due to the lack of timely rejuvenation actions. Therefore, deploying time-based VM migration as support for rejuvenation requires searching for the *availability-aware VM migration trigger* (i.e., the interval between migrations that maximizes system availability). Compared to the current literature, this case study adopts a slightly different approach as it merges the hypoexponential two-phase for aging modeling and considers specific aspects such as VM migration pre-copy phase (a combination of Machida et al. [2013] and Torquato et al. [2022a]).

We search for the *availability-aware VM migration trigger* using sensitivity analysis of transition `migTrigger` delay on the system availability. We study the steady-state availability while varying VM migration triggers from 30 minutes to 48 hours using a 6-minute (0.1 hours) step. Our results also include the *baseline* scenario, in which the VM migration is disabled. Figure 9.3 shows the obtained results. Figure 9.3(a) shows the steady-state availability results, and Figure 9.3(b)

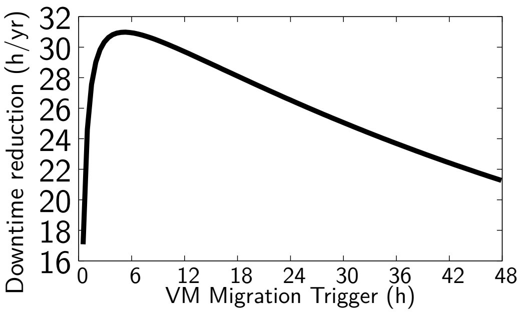
Table 9.2: Parameters used in the timed transitions

Transition	Description	Delay
`migTrigger`	Time for VM migration	Depends on the scenario
`pcMigration`	Pre-copy phase of VM migration	Varying from 72 to 96 seconds*1.
`migDowntime`	VM migration downtime	4 seconds
`failTarget`	Mean time to failure of the Target Host	51.53 days
`repairTarget`	Mean time to repair of the Target Host	1.09 hours
`TargetRestart`	Mean time to apply diversity or rejuvenation in the Target Host	2 minutes
`agingPhase`	First phase of software aging	1 week
`agingAccumulation`	Last phase of software aging	3 days
`agingRepair`	Mean time to repair after an aging failure	1 hour
`hostFail`	Mean time to failure of the Primary Host	51.53 days
`hostRepair`	Mean time to repair of the Primary Host	1.09 hours
`rebootVM`	Mean time to reboot VMs after non-aging failure	5 minutes
`reconV(1,2)`	*Reconnaissance* phase of the attack	30 minutes
`atkV(1,2)Phase`	*Attack* phase (Erlang)	6 hours*2
*1 We adopted the same idea presented in Chapter 4. `pcMigration` delay uses the following equation $PC + PC * (\#FailureProbable)/3$, where $PC = 72$ seconds, and $\#FailureProbable$ is the probability of token presence in the place `FailureProbable`.		
*2 As we have four Erlang phases, *attack* phase total delay is of 24 hours		

present the downtime reduction results. Downtime reduction is the difference between the expected downtime results with and without software rejuvenation. We measure the downtime reduction in hours per year.



(a) Steady-state availability



(b) Downtime reduction

Figure 9.3: CS#1 Results

The availability improvement is noticeable among the entire sensitivity analysis range. Even the worst rejuvenation scenario (i.e., VM migration trigger of 30 min) produces more than 16 hours of annual downtime reduction compared to the baseline. Our results suggest that the *availability-aware migration trigger* is of 5.1 hours. We summarize the results of the *availability-aware migration trigger* in Table 9.3, showing that it reduces the annual downtime by more than 70% when compared to the baseline scenario.

Table 9.3: Comparison between baseline and availability-aware migration trigger results (5.1 hr)

Baseline availability	**Baseline downtime (h/yr)**	**Rej. enabled availability**	**Rej. enabled downtime (h/yr)**	**Downtime reduction due to rejuvenation**
0.995068	43.20	0.99861	12.19	31.01

9.3.2 CS #2 - MTD Protection using Availability-aware Trigger

This case study aims to answer RQ_2: *What is the MTD protection while using VM migration trigger interval for availability maximization?* We conduct a transient evaluation[8] of the probability of attack success while using up to four *hypervisor* variants. The evaluation results show the probability of attack success curve in the first week (168 hours) of the system runtime. The results are in Figure 9.4. The **1N** curve corresponds to the probability of attack success with disabled MTD. **2N, 3N** and **4N** curves correspond to the system with two, three, and four *hypervisor* variants available in the physical hosts, respectively. Note that we consider that

[8] It is worth noting that the steady-state *PAS* is equal to 100% because of the considered *persistence* tactic adopted in the attack

the attack is in place from the initial moment of the evaluation (i.e., the attack starts at time $t = 0$).

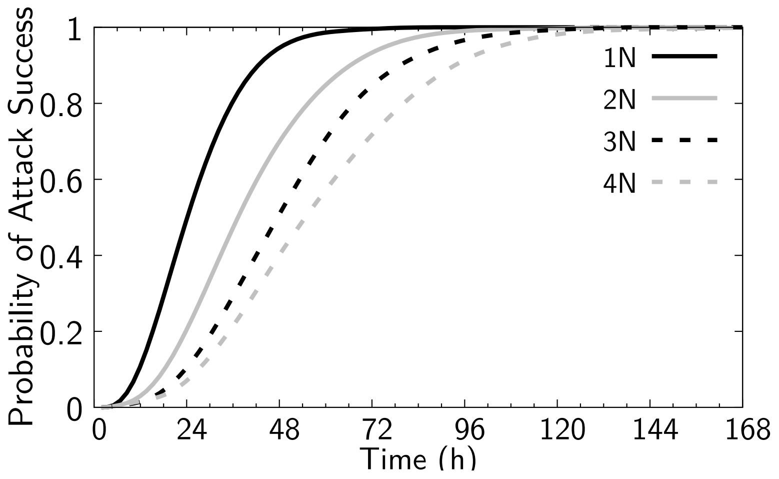


Figure 9.4: CS#2 Results

We notice a flattening effect on the curve while increasing the number of *hypervisor* variants. We also see that the MTD has a time window of effectiveness, which means that there are intercepts of PAS curves at the early or late stages of the evaluation. There is no MTD effect on reducing the probability of attack success compared to the system without MTD in these stages.

We compute our proposed secondary metric, Pointwise Probability of Attack Success ($PAS(t)$), to investigate the probability of attack success at a specific point in time (t). $PAS(t)$ results may help establish the MTD time window of effectiveness. Besides that, $PAS(t)$ results are relevant for systems with job completion deadlines. In such cases, it is possible to verify whether the proposed VM migration schedule can maintain the MTD protection at an acceptable level during job execution. We exercise $PAS(t)$ evaluation in different scenarios as presented in Table 9.4.

Table 9.4: $PAS(t)$ while using $migTrigger = 5.1$ hrs

t in hours	**1N**	**2N**	**3N**	**4N**
1	0.000	0.000	0.000	0.000
6	0.009	0.009	0.009	0.009
24	0.545	0.220	0.119	0.078
72	0.999	0.953	0.877	0.748
144	1.000	1.000	1.000	0.997

The results show that, up to 6 hours, the use of MTD provides no reduction in the probability of attack success. We notice that the difference of PAS results with $t > 144$ hours is negligible. However, the results with $t = 24$ hours suggest a substantial reduction in attack success probability in scenarios with enabled

MTD. For example, in the case of the $4N$ architecture, the reduction is 47%. Note that, in our evaluations, the VM migration trigger is 5.1 hours. Therefore, at the early stages of the evaluation ($t < 5.1$ hours), the MTD effect is yet to take place as the system is still waiting for the first migration.

The time (t) is the independent variable in the $PAS(t)$ calculation, and PAS is the dependent variable. However, there are other systems where we need to swap these variables. For example, systems where it is necessary to keep the probability of attack success under a certain threshold. TL results help answer questions like: "For how long this specific time-based VM migration policy can maintain the probability of attack success under 10%?". TL is particularly useful when setting up security SLAs because it highlights the expected system resistance time observing a PAS tolerance level. We computed TL metrics with PAS arbitrary values. The obtained results are in Table 9.5.

Table 9.5: TL results while using *migTrigger* = 5.1 hrs

PAS	**1N**	**2N**	**3N**	**4N**
1%	7 hrs	7 hrs	7 hrs	7 hrs
20%	15 hrs	23 hrs	30 hrs	34 hrs
70%	29 hrs	46 hrs	57 hrs	68 hrs
100%	76 hrs	126 hrs	133 hrs	>168hrs

The results suggest that assuming $TL = 1\%$, it makes no difference in applying MTD or not. PAS of 1% is reached about 7 hours in all studied scenarios. Alternatively, when $PAS = 70\%$, the use of a $4N$ architecture more than doubles the TL when compared to the $1N$ architecture. As in the $PAS(t)$ evaluation, the TL results also consider the VM migration trigger of 5.1 hours. This way, the MTD effects only become noticeable after 5.1 hours. As the system is under attack in the entire evaluation, the proposed VM migration trigger is insufficient to increase the system resistance while considering $PAS = 1\%$. On the other hand, the use of *diversity*-based MTD can delay the moment in time when the system reaches PAS of 70%. This delay is due to the accumulation of attack progress interruptions during the MTD deployment.

It is worth presenting a hypothetical example to emphasize the convenience of the proposed metrics for practical situations. Let us consider the results presented in the case studies and a system that tolerates up to $PAS = 20\%$. Aiming to maximize the system availability, we deploy a time-based VM migration interval of 5.1 hours. Table 9.5 reveals that, without MTD, the system may resist for the first 15 hours before reaching the desired PAS tolerance level. If 15 hours is not enough to meet our requirements, it is possible to increase TL through the MTD deployment. TL increase is directly related to the number of *hypervisor* variants available in the physical hosts. Therefore, we should select a $4N$ architecture to achieve the best security levels.

9.3.3 CS #3 - Availability and MTD Protection Evaluation

The previous case study evaluates MTD under the fixed *availability-aware migration trigger*. *Availability-aware migration trigger* is neither too short to avoid excessive downtime due to VM migration nor too long to allow software aging failures. However, from a purely MTD perspective, the objective is to have more frequent VMs migration. This way, the attacker will face more interruptions during the attack course.

This last case study combines the availability and security evaluation to answer RQ_3: *What are the tradeoffs between availability and security when selecting a specific VM migration schedule?* We compute the system *Increased Resistance* (IR) as the security perspective. IR is the TL difference between the system with and without MTD. For this hypothetical example, we assume $PAS = 10\%$ for the TL computation. We use the downtime reduction results for the availability perspective.

Figure 9.5 presents the results. We use a plot with a double Y-axis. The left Y-axis represents the IR results. IR results curves are in black. The right Y-axis represents the downtime reduction results. The downtime reduction curve is in gray. In this plot, the X-axis corresponds to the VM migration trigger intervals. We limit the X-axis range up to the *availability-aware migration trigger* (i.e., 5.1 hours). We adopt this approach because longer VM migration trigger intervals provide worse results for both availability and security.

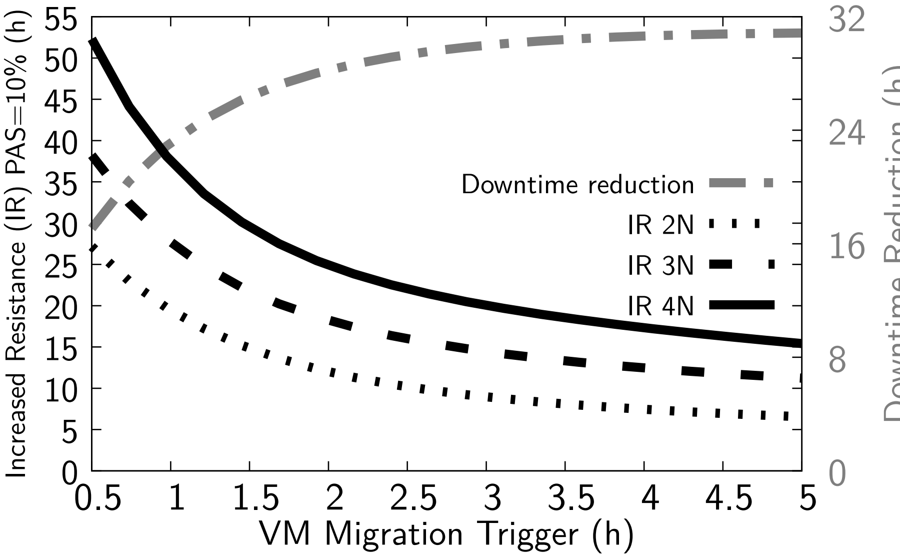


Figure 9.5: CS#4 Results

The plot in Figure 9.5 highlights the availability and security tradeoffs in the VM migration trigger interval selection. For example, while the best policy for maximizing system availability is on the rightmost side of the plot, the best policy for security is on the leftmost side. It is noticeable that deciding on one of the metrics (availability or security) will compromise the other. It is hard to define the best policy as the availability and security requirements depend on the specific business running in the virtualized environment.

One of the possible approaches to selecting the VM migration trigger interval in this availability vs. security scenario is through MCDM [Triantaphyllou, 2000]. For illustration, we conduct MCDM considering three criteria: *availability* - in the form of downtime reduction, *security* - by means of Increased Resistance (IR); and *cost*. We consider that the addition of *hypervisor* variants increases the *cost* due to the interoperability actions. For this illustrative example, we consider that $2N$ costs 100, $3N$ costs 200, and $4N$ costs 300. As presented in Araujo et al. [2018], we used TOPSIS method [Pavić and Novoselac, 2013] for the analysis. In summary, TOPSIS method aims to select the alternative in the shortest distance from the ideal solution and in the farthest distance of the worst scenario (i.e., negative ideal solution) [Jahanshahloo et al., 2006].

The results are in Table 9.6. The columns w_{ir}, w_{dwt} and w_{cost} represent the assigned weight for IR, downtime reduction and cost, respectively. The remaining columns provide the best solution (BS) and its associated results. Those results are illustrative and highlight how the VM migration schedule should be adapted accordingly to the system requirements. The MCDM results suggest a preference for security-aware VM migration triggers (i.e., triggers focusing on maximizing security) over availability-aware triggers. We see this behavior because the range of values in the IR curves is wider than the range of values in the downtime reduction curve. We only see a more *availability-aware* VM migration (i.e., closer to 5.1 hours between migrations) when we reduce the security weight to 15% and boost the availability weight to 85%.

Table 9.6: Multi-criteria decision making - VM migration trigger alternatives

w_{ir}	w_{dwt}	w_{cost}	BS *mig Trigger*	BS arch.	**IR**	**Downt. reduction**
33.33%	33.33%	33.33%	0.5 h	4N	52.31 h	17.08 h
50%	25%	25%	0.5 h	4N	52.31 h	17.08 h
25%	50%	25%	0.5 h	4N	52.31 h	17.08 h
25%	25%	50%	0.5 h	2N	27.16 h	17.08 h
15%	85%	0%	2.5 h	4N	22.13 h	30.06 h

9.4 Threats to Validity and Limitations

Lack of comparison with real testbed results

The presented model results were not compared against experimental results from a testbed. Thus, it is possible to argue about the validity of the model. The selection of model-based evaluation comes from the necessity of testing and comparing *hard* scenarios. By *hard*, we mean that it is difficult or onerous to properly reproduce or achieve in a real testbed. Software aging accumulation requires the submission of specific workloads to activate the *aging*-related bugs [Escheikh et al., 2016; Bovenzi et al., 2011]. The submission of such workload along with a *attack-defense* workload may lead the system to a non-representative state. Therefore, the non-representative state evaluation will produce potentially innocuous results.

Parameters retrieved from previous works instead of experimentation

The ideal situation is when we collect parameters from measurements in a real testbed. However, our research relies on previously published papers to retrieve the parameters of the models. Nevertheless, it is important to highlight that previous papers on the same topic [Mendonça et al., 2020; Alavizadeh et al., 2018a] also retrieved parameters from the literature. Finally, the reproduction of the threat model in a real testbed may have a non-negligible cost. This reproduction needs a dedicated virtualized infrastructure and may require a specialized *red team* [Diogenes and Ozkaya, 2018] to conduct the attacks.

Proposed MTD and software rejuvenation are focused on host-only problems

Although the use of VM migration as software rejuvenation and MTD is beneficial for the final clients in the cloud, the entire strategy is focused on host (i.e., physical host) protection. Then, the applied approach serves only to propose a partial security posture for a cloud environment. In a comprehensive setting, it is necessary to take others threats into account. For example, application-level threats and software aging are potential problems that may jeopardize the validity of the proposed approach. Nevertheless, the modeling provides enough flexibility to study and analyze the system before the MTD deployment. System managers and cybersecurity experts may leverage the proposed approach as support for dependability-security policy decision-making.

9.5 Summary

This chapter presented a set of SPN models for availability and security evaluation of multipurpose time-based VM migration. In our study, time-based VM migration simultaneously serves as software rejuvenation and MTD. Results show that time-based VM migration produces substantial availability and security improvement compared to the baseline conditions. The proposed secondary metrics highlight important practical aspects of MTD deployment. At the time of this chapter writing, we could not find a previous study covering this specific multipurpose time-based VM migration.

Three research questions guided the development of this chapter. The first one is about the VM migration interval that maximizes the system steady-state availability. In the proposed scenarios, we found out that the interval of 5 hours and 6 minutes between migrations produces the best availability result. It reduces the system expected downtime to more than 31 hours per year. The second research question explores the system protection while using MTD based on VM migration. Results showed a reduction of 47% in the probability of attack success while adopting the *4N* (i.e., most capable deployment) architecture. The last research question is about the tradeoffs between availability and security of using VM migration as MTD. The results indeed suggested a tradeoff between these metrics. However, in the balanced scenarios (i.e., scenarios with equal priority for the metrics), the best deployment was 30 minutes between migrations in a *4N* architecture.

Our findings show that to boost one attribute (i.e., availability or security), we should compromise the other. Using Multi-Criteria Decision Making (MCDM) methods may aid the user in selecting appropriate VM migration trigger intervals. The model analysis also suggests that it is advantageous to enable *hypervisor* variants interoperability to enhance system security.

Chapter 10

Conclusion and Future Work

Cybersecurity is a top concern for cloud computing users. Regardless of private or public deployments, the need for proactive and adaptive defensive mechanisms is extremely high due to the emergence of novel threats. MTD appears as an alternative, providing flexibility to mitigate complex and advanced security threats. Nevertheless, a comprehensive evaluation framework for a proper MTD deployment is still an open problem. There are, in fact, many questions that need to be responded to, such as *once the attacker is in the environment, for how long can my system resist the attack? how often do I need to apply MTD to reach a desired level of security against a specific threat?* and *what is the impact of a MTD deployment on other system metrics?*

The answers to the questions mentioned above depend on the development or design of suitable evaluation methods for MTD-enabled cloud environments. A proper evaluation scheme should also consider the other relevant aspects of the virtualized domain. One of the interesting perspectives in this context is the system availability, which is a double-sided metric [Avizienis et al., 2004], as it may decrease due to security issues (e.g., a successful DoS attack) or dependability problems (e.g., failure in a system component). Indeed, availability is one of the major concerns for cloud computing managers and providers and is usually among the fundamental SLA (e.g., an IaaS provider offers availability of 99.999% for its clients). Therefore, the evaluation presented in this thesis not only takes into account the security metric but also the system availability. One of the subtle issues in this field, already reported as a problem in cloud computing, is software aging. Our approach also takes these problems into account to deliver a more well-rounded evaluation solution.

The work presented in this thesis provided evaluation approaches for MTD in virtualized environments. Based mainly on SPN models, this thesis delivers a series of models ranging from baseline models focusing on availability evaluation of software rejuvenation-enabled environments to models with MTD against *host-based* attacks. Specifically, the scope comprises different aspects of a VM migration-enabled system. Firstly, we developed a series of baseline models for evaluating system availability while using VM migration to support software rejuvenation. Then, we extend the proposed model to cover the security aspect by proposing a security risk evaluation metric (RiskScore). Finally, we adapted the baseline models to cover the MTD aspect under different circumstances. Specifically, adapted models cover the aspects of using VM migration as MTD against *persistent* and *non-persistent* attacks. This approach of considering both scenarios provided a comprehensive overview of VM migration MTD strengths and limitations.

Our research path resulted in a set of contributions for the fields of dependability and security evaluation of virtualized systems with VM migration. Each step towards the main goal resulted in an individual contribution (as highlighted in Chapter 1). In summary, we first devoted attention to the practical aspects of VM migration and put it to test in a real-world environment. The results obtained show the feasibility of using it for the desired purposes. Then, we design the first model to cover the baseline aspects of a cloud with VM migration. The baseline model extends the availability evaluation and also includes performability metrics. After, we exercised the security evaluation in the same scenario and subsequently proposed a secondary metric to evaluate the security from a *system-oriented* perspective instead of a *attack* perspective. Finally, we proposed models for security and availability evaluation of a system using VM migration as MTD. This step was further extended by the proposal of the *PyMTDEvaluator* tool, which provides an automated way to perform analysis and comparison of different MTD migration schedules. The tool is open-source and publicly available through a Docker container. Closing our set of contributions, we analyzed multipurpose VM migration considering scenarios where the VM migration serves both as MTD and software rejuvenation.

10.1 Key Takeaways

We learned several lessons during the development of this work. Below, we highlight some of the main ones. The discussion sections in each of the chapters do add some details to this list.

Our empirical results show that, as expected, **VM migration can defend against *host-based* attacks**. The use of appropriated time-based VM migration prevents Memory DoS (i.e., a *non-persistent* attack) attack success. Nevertheless, the VM migration frequency plays a key role in the defense effectiveness. In fact, results suggest that there is an optimal VM migration frequency that minimizes the accumulated impact while preventing attack success.

The proposed baseline models show that **in scenarios with higher probability and intensity of bursty workloads, the overall performability improvement due to VM migration is minor.** The quantification of the performability improvement is helpful to guide the design of SLAs. Furthermore, the results also show significant improvement in the system reliability while applying VM migration for software rejuvenation support.

Our modeling results show that, **in scenarios under software aging conditions and with multiple threats as DoS and MITM, it is preferred to apply more frequent VM migrations, as they mitigate software aging accumulation**. Despite the VM migration increasing the risk of MITM attack success (because we are transferring the memory pages), in cases where both threats (i.e., DoS and MITM) have equal importance (i.e., have the same weight for the evaluation) frequent migrations tend to improve the system overall security. This is due to the accumulated time that the system stays in a *risky* state. Applying frequent migrations increases the "in migration" state, thus, the risk of a MITM attack success also increases. However, the frequent migrations also come with the benefit of increased resistance to software aging effects, which, in our scenarios, displayed better security levels when considering both threats (i.e., DoS and MITM) with equal importance in the evaluation.

The MTD evaluation models suggest that **time-based VM migration can extend the system resistance to *host-based* attacks, but it is unable to prevent attack success**. Hypothetically, time-based VM migration MTD will be able to prevent attack success in scenarios with very frequent MTD actions. Nevertheless, these scenarios seem unrealistic as such a VM migration accumulated overhead may impair the other system metrics. Furthermore, the models enable the finding of the *effectiveness limit* of the deployed MTD. This metric is interesting to evaluate the point in time where the MTD deployment becomes innocuous.

The model parameters are limited to hypothetical environments. Therefore, more than such generic and illustrative parameters may be required to reflect specific situations. **Despite the illustrative aspect of generic parameters, they are useful to understand system behavior under similar circumstances. For a more specific and representative evaluation, we need to adjust the parameter accordingly**. *PyMTDEvaluator* supports this flexible evaluation through an easy-to-use interface. In an attempt to facilitate the *PyMTDEvaluator* usage, it is available in a Docker container.

As a final takeaway, we highlight that **the adoption of multipurpose (i.e., software rejuvenation and MTD) time-based VM migration provides significant availability and security improvement when compared to the baseline conditions**. The proposed evaluation also emphasizes enabling VM migration between heterogeneous hypervisors benefits system security. These results might be useful for system managers to understand the availability and security improvements for an already in-place VM migration policy.

10.2 Future Work

This thesis sheds light on new research directions. Below, we suggest possible future works.

- **Find the optimal schedule for VM migration.** The VM migration frequency will surely vary depending on the scenario of MTD deployment. However, working into finding the optimal schedule for the VM migration may provide insights into what factors influence its effectiveness. Furthermore, through a series of tests, it will be possible to present more statistically significant results, including mean time to attack success and its standard deviation. The first step in this direction is to reproduce the empirical observations varying the time between migrations. The main goal is finding a specific schedule that reduces the VM migration frequency and yet is capable of preventing attack success. After this, it is necessary to repeat the experiment to generate a statistically significant sample of data.

- **Adoption of hierarchical compositions or interacting SPN models to extend the proposed MTD evaluation**. More powerful models are needed to investigate the security, availability, and performability tradeoffs when applying MTD based on VM migration scheduling. The idea is to develop a separate model to evaluate only performance that should communicate with the models presented in this thesis. This communication will enable a combined evaluation of performance, availability, and probability of attack success. It is also interesting to adapt the model to cover the economic aspects of the considered scenarios. Economic metrics may highlight how much the attacker needs to invest to reach the attack success. An interesting approach is presented in [Alavizadeh et al., 2018a], which uses a *return on attack* economic metric.
- **Extend the architecture to larger environments.** To represent larger deployments, it is necessary to adapt the models to represent such architectures. The approach to reach this goal is similar to the previous one: use hierarchical composition. However, instead of using interacting models, one possible path is to encapsulate a minimal infrastructure model in a RBD. RBD models are easier to scale as they can be easily reduced to closed formulas. We apply a similar idea in our previous work [Torquato et al., 2019a].
- **Reproduce alternative threat models to check VM migration as MTD effectiveness.** The idea is to reproduce the attack on the proposed threat model in a real testbed. The main goal is to measure the time for attack completion. Besides that, research is needed to understand the performance overhead impact due to the MTD deployment. A possible start in this direction is through the exposure of a VM migration based MTD against *persistent* attacks (e.g., VM escape attack).
- **Include more probability distributions to represent attack progress behavior in the *PyMTDEvaluator* implementation.** The inclusion of diverse probability distributions will enable the evaluation of different scenarios of MTD deployment. The first step is to conduct experiments of attack and defense in a real-world setup (as described before) to gather data about the behavior of the system. Once the probability distribution of the events is retrieved, it is possible to adapt the *PyMTDEvaluator* code as it is open-source. Although the experiments are the first recommended step, it is possible to adjust the *PyMTDEvaluator* implementation beforehand to evaluate hypothetical scenarios.

Besides these future works, based on the acquired knowledge on MTD, we also suggest the following long-term research directions in the field:

- **Adoption of Artificial Intelligence (AI) and ML techniques to empower MTD evaluation.** The use of AI and ML methods could be a way to enhance the model- and simulation-based evaluations. For example, it is possible to use genetic algorithms and solution search approaches to find specific VM migration schedules to reach specific levels of security or availability. We started this research path in Torquato et al. [2019c]. In that work, we proposed a *genetic algorithm* implementation to exercise a RBD model. The goal is to search for architectures that satisfy some target availability levels. The proposed *genetic algorithm* helps in answering a question as *"Which architecture deployment provides availability of 99.999%?"* The next step is to adapt this genetic algorithm to interact with *PyMTDEvaluator*. Therefore, the possible resulting implementation might be able to answer questions such as *"What migration schedule is able to keep the probability of attack success below 40% in the first month while achieving a maximum of five minutes of downtime?"*

- **Inclusion of alternative MTD techniques to provide a comprehensive cloud security posture.** As stated earlier in this thesis, the proposed VM migration-based MTD is only able to defend against a specific set of threats. For a more comprehensive MTD deployment, it is necessary to expand the evaluation to additional MTD actions. For example, it is possible also to consider SDN-based MTD deployment in combination with the proposed MTD.

- **Measurement-based MTD evaluation to guide model design.** Usually, the model-based evaluation proposals neglect experimental background. The main goal of the measurement evaluation is to provide data to support the model design. The system experimentation and testing provide relevant insights to design and calibrate the models. Here, the guideline to keep in mind is the goal of knowledge acquisition of the system behavior. Note that this research direction does not nullify the path into modeling systems from a *knowledge-only* perspective. As mentioned previously in this thesis, there are scenarios where the measurements are hard to obtain. In those, the proposal of analytical evaluation models could be useful to understand the system expected behavior without interacting with it. Nevertheless, whenever possible, the empirical observation and testing of a system may bring valuable insights into designing better models.

- **Combined evaluation for VM migration deployments.** VM migration may be useful for various purposes. In this thesis, we observed the feature from the perspective of dependability and security. Nevertheless, VM migration is also useful to increase system maintainability, as it facilitates the preventive maintenance moving VMs to selected PMs. Besides that, the same technique is useful to improve system sustainability (e.g., the migration feature simplifies *packing* the VMs in fewer PMs) and may also be applied to improve system performance through load balancing. Proposing combined evaluations for these scenarios may be useful to provide a holistic view of time-based VM migration deployment.

Bibliography

Adili, M. T., Mohammadi, A., Manshaei, M. H., and Rahman, M. A. (2017). A cost-effective security management for clouds: A game-theoretic deception mechanism. In *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*, page 98–106. IEEE.

Agarwal, A. and Duong, T. N. B. (2019). Secure virtual machine placement in cloud data centers. *Future Generation Computer Systems*, 100:210–222.

Ahmad, R. W., Gani, A., Hamid, S. H. A., Shiraz, M., Yousafzai, A., and Xia, F. (2015). A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of network and computer applications*, 52:11–25.

Ahmed, N. O. and Bhargava, B. (2016). Mayflies: A moving target defense framework for distributed systems. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, page 59–64. ACM.

Akoush, S., Sohan, R., Rice, A., Moore, A. W., and Hopper, A. (2010). Predicting the performance of virtual machine migration. In *2010 IEEE international symposium on modeling, analysis and simulation of computer and telecommunication systems*, page 37–46. IEEE.

Alavizadeh, H. (2020). *Effective Security Analysis for Combinations of MTD Techniques on Cloud Computing.* PhD thesis, Massey University.

Alavizadeh, H., Alavizadeh, H., Kim, D. S., Jang-Jaccard, J., and Torshiz, M. N. (2019a). An automated security analysis framework and implementation for mtd techniques on cloud. In *International Conference on Information Security and Cryptology*, page 150–164. Springer.

Alavizadeh, H., Hong, J. B., Jang-Jaccard, J., and Kim, D. S. (2018a). Comprehensive security assessment of combined mtd techniques for the cloud. In *Proceedings of the 5th ACM Workshop on Moving Target Defense*, page 11–20. ACM.

Alavizadeh, H., Hong, J. B., Kim, D. S., and Jang-Jaccard, J. (2020). Evaluating the effectiveness of shuffle and redundancy mtd techniques in the cloud. *Computers & Security*, page 102091.

Alavizadeh, H., Hong, J. B., Kim, D. S., and Jang-Jaccard, J. (2021). Evaluating the effectiveness of shuffle and redundancy mtd techniques in the cloud. *Computers & Security*, 102:102091.

Alavizadeh, H., Jang-Jaccard, J., and Kim, D. S. (2018b). Evaluation for combination of shuffle and diversity on moving target defense strategy for cloud computing. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, page 573–578. IEEE.

Alavizadeh, H., Kim, D. S., Hong, J. B., and Jang-Jaccard, J. (2017). Effective security analysis for combinations of mtd techniques on cloud computing (short paper). In *International Conference on Information Security Practice and Experience*, page 539–548. Springer.

Alavizadeh, H., Kim, D. S., and Jang-Jaccard, J. (2019b). Model-based evaluation of combinations of shuffle and diversity mtd techniques on the cloud. *Future Generation Computer Systems.*

Alhozaimy, S. and Menascé, D. A. (2022). A formal analysis of performance-security tradeoffs under frequent task reconfigurations. *Future Generation Computer Systems*, 127:252–262.

Alonso, J. and Trivedi, K. (2015). Software rejuvenation and its application in distributed systems. *Quantitative Assessments of Distributed Systems: Methodologies and Techniques*, pages 301–325.

Ammar, H., Huang, Y., and Liu, R. (1987). Hierarchical models for systems reliability, maintainability, and availability. *IEEE Transactions on Circuits and Systems*, 34(6):629–638.

Anderson, N., Mitchell, R., and Chen, R. (2016). Parameterizing moving target defenses. In *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, page 1–6. IEEE.

Andrade, E., Pietrantuono, R., Machida, F., and Cotroneo, D. (2021). A comparative analysis of software aging in image classifiers on cloud and edge. *IEEE Transactions on Dependable and Secure Computing.*

Araujo, J., Maciel, P., Andrade, E., Callou, G., Alves, V., and Cunha, P. (2018). Decision making in cloud environments: an approach based on multiple-criteria decision analysis and stochastic models. *Journal of Cloud Computing*, 7(1):1–19.

Araujo, J., Matos, R., Alves, V., Maciel, P., Souza, F. V. d., Jr, R. M., and Trivedi, K. S. (2014). Software aging in the eucalyptus cloud computing infrastructure: characterization and rejuvenation. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 10(1):1–22.

Araujo, J., Matos, R., Maciel, P., Matias, R., and Beicker, I. (2011). Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure. In *Proceedings of the Middleware 2011 Industry Track Workshop*, page 4. ACM.

Asadabadi, M. R., Chang, E., and Saberi, M. (2019). Are mcdm methods useful? a critical review of analytic hierarchy process (ahp) and analytic network process (anp). *Cogent Engineering*, 6(1):1623153.

Ashino, Y. and Nakae, M. (2012). Virtual machine migration method between different hypervisor implementations and its evaluation. In *2012 26th International Conference on Advanced Information Networking and Applications Workshops*, page 1089–1094. IEEE.

Avizienis, A., Laprie, J.-C., and Randell, B. (2001). Fundamental concepts of computer system dependability. In *Workshop on Robot Dependability: Technological Challenge of Dependable Robots in Human Environments*, page 1–16. University of Newcastle upon Tyne, Computing Science.

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33.

Awasthi, A. and Gupta, R. (2016). Multiple hypervisor based open stack cloud and vm migration. In *2016 6th International Conference-Cloud System and Big Data Engineering (Confluence)*, page 130–134. IEEE.

Azab, M. and Eltoweissy, M. (2011). Defense as a service cloud for cyber-physical systems. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, page 392–401. IEEE.

Azab, M., Eltoweissy, M., Attiya, G., et al. (2017). Towards online smart disguise: Real-time diversification evading co-residency based cloud attacks. In *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)*, pages 235–242. IEEE.

Azab, M., Mokhtar, B. M., Abed, A. S., and Eltoweissy, M. (2016). Smart moving target defense for linux container resiliency. In *Collaboration and Internet Computing (CIC), 2016 IEEE 2nd International Conference on*, page 122–130. IEEE.

Bahrami, P. N., Dehghantanha, A., Dargahi, T., Parizi, R. M., Choo, K.-K. R., and Javadi, H. H. (2019). Cyber kill chain-based taxonomy of advanced persistent threat actors: Analogy of tactics, techniques, and procedures. *Journal of information processing systems*, 15(4):865–889.

Bai, J., Chang, X., Machida, F., Trivedi, K. S., and Han, Z. (2020). Analyzing software rejuvenation techniques in a virtualized system: Service provider and user views. *IEEE Access*, 8:6448–6459.

Bai, J., Chang, X., Machida, F., Trivedi, K. S., and Li, Y. (2023a). Model-driven dependability assessment of microservice chains in mec-enabled iot. *IEEE Transactions on Services Computing.*

Bai, J., Li, Y., Chang, X., Machida, F., and Trivedi, K. S. (2023b). Understanding nfv-enabled vehicle platooning application: A dependability view. *IEEE Transactions on Cloud Computing.*

Bause, F. (1993). Queueing petri nets-a formalism for the combined qualitative and quantitative analysis of systems. In *Proceedings of 5th international workshop on Petri nets and performance models*, page 14–23. IEEE.

Bazm, M.-M., Lacoste, M., Südholt, M., and Menaud, J.-M. (2017). Side-channels beyond the cloud edge: New isolation threats and solutions. In *Cyber Security in Networking Conference (CSNet), 2017 1st*, page 1–8. IEEE.

Bobbio, A. (1990). System modelling with petri nets. In *Systems Reliability Assessment: Proceedings of the Ispra Course held at the Escuela Tecnica Superior de Ingenieros Navales, Madrid, Spain, September 19–23, 1988 in collaboration with Universidad Politecnica de Madrid*, pages 103–143. Springer.

Bolch, G., Greiner, S., De Meer, H., and Trivedi, K. S. (2006). *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications.* John Wiley & Sons.

Boubour, R., Jard, C., Aghasaryan, A., Fabre, E., and Benveniste, A. (1997). A petri net approach to fault detection and diagnosis in distributed systems. i. application to telecommunication networks, motivations, and modelling. In *Proceedings of the 36th IEEE Conference on Decision and Control*, volume 1, pages 720–725. IEEE.

Bovenzi, A., Cotroneo, D., Pietrantuono, R., and Russo, S. (2011). Workload characterization for software aging analysis. In *Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on*, page 240–249. IEEE.

Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., Gelenbe, E., Javadi, B., Vaquero, L. M., Netto, M. A., et al. (2018). A manifesto for future generation cloud computing: Research directions for the next decade. *ACM Computing Surveys (CSUR)*, 51(5):105.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616.

Cai, G., Wang, B., Luo, Y., and Hu, W. (2016a). A model for evaluating and comparing moving target defense techniques based on generalized stochastic petri net. In *Conference on Advanced Computer Architecture*, page 184–197. Springer.

Cai, G., Wang, B., Luo, Y., Li, S., and Wang, X. (2016b). Characterizing the running patterns of moving target defense mechanisms. In *Advanced Communication Technology (ICACT), 2016 18th International Conference on*, pages 191–196. IEEE.

Cai, G.-l., Wang, B.-s., Hu, W., and Wang, T.-z. (2016c). Moving target defense: state of the art and characteristics. *Frontiers of Information Technology & Electronic Engineering*, 17(11):1122–1153.

Candea, G., Kawamoto, S., Fujiki, Y., Friedman, G., and Fox, A. (2004). Microreboot–a technique for cheap recovery. *arXiv preprint cs/0406005*.

CAPEC (2023). Capec-94: Man in the middle attack.

Casola, V., De Benedictis, A., Rak, M., and Villano, U. (2018). A security sla-driven moving target defense framework to secure cloud applications. In *Proceedings of the 5th ACM Workshop on Moving Target Defense*, page 48–56. ACM.

Castellanos Contreras, J. U. and Rodríguez Urrego, L. (2023). Technological developments in control models using petri nets for smart grids: A review. *Energies*, 16(8):3541.

Chang, X., Shi, Y., Zhang, Z., Xu, Z., and Trivedi, K. S. (2020). Job completion time under migration-based dynamic platform technique. *IEEE Transactions on Services Computing*, page 1–1.

Chen, S.-J. and Hwang, C.-L. (1992). Fuzzy multiple attribute decision making methods. In *Fuzzy multiple attribute decision making: Methods and applications*, pages 289–486. Springer.

Chen, Z., Chang, X., Han, Z., and Yang, Y. (2020). Numerical evaluation of job finish time under mtd environment. *IEEE Access*, 8:11437–11446.

Ching, W.-K. and Ng, M. K. (2006). Markov chains. *Models, algorithms and applications*.

Cho, J.-H., Sharma, D. P., Alavizadeh, H., Yoon, S., Ben-Asher, N., Moore, T. J., Kim, D. S., Lim, H., and Nelson, F. F. (2020). Toward proactive, adaptive defense: A survey on moving target defense. *IEEE Communications Surveys & Tutorials*, 22(1):709–745.

Chong, F., Lee, R., Acquisti, A., Horne, W., Palmer, C., Ghosh, A., Pendarakis, D., Sanders, W., Fleischman, E., Teufel III, H., et al. (2009). National cyber leap year summit 2009: Co-chairs' report - moving target defense (chapter 4). *NITRD Program*.

Chowdhary, A., Alshamrani, A., Huang, D., and Liang, H. (2018). Mtd analysis and evaluation framework in software defined network (mason). In *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, page 43–48. ACM.

Christopher Frey, H. and Patil, S. R. (2002). Identification and review of sensitivity analysis methods. *Risk analysis*, 22(3):553–578.

Chung, C.-J., Xing, T., Huang, D., Medhi, D., and Trivedi, K. (2015). Serene: on establishing secure and resilient networking services for an sdn-based multi-tenant datacenter environment. In *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, page 4–11. IEEE.

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. (2005). Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, NSDI'05, page 273–286, Berkeley, CA, USA. USENIX Association, USENIX Association.

Clemente, D., Pereira, P., Dantas, J., and Maciel, P. (2022). Availability evaluation of system service hosted in private cloud computing through hierarchical modeling process. *The Journal of Supercomputing*, 78(7):9985–10024.

Connell, W., Menasce, D. A., and Albanese, M. (2018). Performance modeling of moving target defenses with reconfiguration limits. *IEEE Transactions on Dependable and Secure Computing*.

Connell, W., Menascé, D. A., and Albanese, M. (2017). Performance modeling of moving target defenses. In *Proceedings of the 2017 Workshop on Moving Target Defense*, page 53–63.

Conti, M., Dragoni, N., and Lesyk, V. (2016). A survey of man in the middle attacks. *IEEE Communications Surveys & Tutorials*, 18(3):2027–2051.

Corporation, T. M. (2023). Persistence - tactic - mitre att&ck.

Costa, J., Matos, R., Araujo, J., Li, J., Choi, E., Nguyen, T. A., Lee, J.-W., and Min, D. (2023). Software aging effects on kubernetes in container orchestration systems for digital twin cloud infrastructures of urban air mobility. *Drones*, 7(1):35.

Cotroneo, D., De Simone, L., Natella, R., Pietrantuono, R., and Russo, S. (2022). Software micro-rejuvenation for android mobile systems. *Journal of Systems and Software*, 186:111181.

Cotroneo, D., Natella, R., Pietrantuono, R., and Russo, S. (2010). Software aging analysis of the linux operating system. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*, page 71–80. IEEE.

Cotroneo, D., Natella, R., Pietrantuono, R., and Russo, S. (2014). A survey of software aging and rejuvenation studies. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 10(1):8.

Dantas, J., Matos, R., Araujo, J., and Maciel, P. (2012). An availability model for eucalyptus platform: An analysis of warm-standy replication mechanism. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, page 1664–1669. IEEE.

Debroy, S., Calyam, P., Nguyen, M., Stage, A., and Georgiev, V. (2016). Frequency-minimal moving target defense using software-defined networking. In *Computing, Networking and Communications (ICNC), 2016 International Conference on*, page 1–6. IEEE.

Diogenes, Y. and Ozkaya, E. (2018). *Cybersecurity??? Attack and Defense Strategies: Infrastructure security with Red Team and Blue Team tactics*. Packt Publishing Ltd.

Distefano, S., Scarpa, M., Chang, X., and Bobbio, A. (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.*

Dohi, T., Goševa-Popstojanova, K., and Trivedi, K. (2001). Estimating software rejuvenation schedules in high-assurance systems. *The Computer Journal*, 44(6):473–485.

Dohi, T., Trivedi, K. S., and Avritzer, A. (2020). *Handbook of Software Aging and Rejuvenation: Fundamentals, Methods, Applications, and Future Directions*. World Scientific.

Dsouza, G., Hariri, S., Al-Nashif, Y., and Rodriguez, G. (2013). Resilient dynamic data driven application systems (rdddas). *Procedia Computer Science*, 18:1929–1938.

Dumitraş, T. and Shou, D. (2011). Toward a standard benchmark for computer security research: The worldwide intelligence network environment (wine). In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, page 89–96. ACM.

El Mir, I., Haqiq, A., and Kim, D. S. (2017). A game theoretic approach for cloud computing security assessment using moving target defense mechanisms. In *Proceedings of the Mediterranean Symposium on Smart City Applications*, page 242–254. Springer.

Enoch, S. Y., Huang, Z., Moon, C. Y., Lee, D., Ahn, M. K., and Kim, D. S. (2020). Harmer: Cyber-attacks automation and evaluation. *IEEE Access*, 8:129397–129414.

Enoch, S. Y., Mendonça, J., Hong, J. B., Ge, M., and Kim, D. S. (2022). An integrated security hardening optimization for dynamic networks using security and availability modeling with multi-objective algorithm. *Computer Networks*, 208:108864.

Escheikh, M., Tayachi, Z., and Barkaoui, K. (2016). Workload-dependent software aging impact on performance and energy consumption in server virtualized systems. In *Software Reliability Engineering Workshops (ISSREW), 2016 IEEE International Symposium on*, page 111–118. IEEE.

Fakhrolmobasheri, S., Ataie, E., and Movaghar, A. (2018). Modeling and evaluation of power-aware software rejuvenation in cloud systems. *Algorithms*, 11(10):160.

Fernandes, D. A., Soares, L. F., Gomes, J. V., Freire, M. M., and Inácio, P. R. (2014). Security issues in cloud environments: a survey. *International Journal of Information Security*, 13(2):113–170.

Fitch, D. F. and Xu, H. (2012). A petri net model for secure and fault-tolerant cloud-based information storage. In *SEKE*, page 333–339. Citeseer.

Fleck, D., Stavrou, A., Kesidis, G., Nasiriani, N., Shan, Y., and Konstantopoulos, T. (2018). Moving-target defense against botnet reconnaissance and an adversarial coupon-collection model. In *2018 IEEE Conference on Dependable and Secure Computing (DSC)*, page 1–8. IEEE.

Flexera (2023). 2023 state of the cloud report.

Fonseca, J., Vieira, M., and Madeira, H. (2013). Evaluation of web security mechanisms using vulnerability & attack injection. *IEEE Transactions on dependable and secure computing*, 11(5):440–453.

Francis, T. (2018). A comparison of cloud execution mechanisms fog, edge, and clone cloud computing. *International Journal of Electrical & Computer Engineering (2088-8708)*, 8(6).

Franklin, B. (2023). 40 cloud computing stats and trends to know in 2023 | google cloud blog.

Garg, S., Puliafito, A., Telek, M., and Trivedi, K. S. (1995). Analysis of software rejuvenation using markov regenerative stochastic petri net. In *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on*, page 180–187.

German, R. (2000). *Performance analysis of communication systems with non-Markovian stochastic Petri nets.* John Wiley & Sons, Inc.

Ghribi, C., Hadji, M., and Zeghlache, D. (2013). Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, page 671–678. IEEE.

Gonçalves, C. F., Antunes, N., and Vieira, M. (2023). Intrusion injection for virtualized systems: Concepts and approach. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 417–430. IEEE.

Goyal, S. (2014). Public vs private vs hybrid vs community-cloud computing: a critical review. *International Journal of Computer Network and Information Security*, 6(3):20–29.

Gray, J. and Reuter, A. (1993). *Transaction processing.* Morgan Kaufíann Publishers.

Groat, S., Moore, R., Marchany, R., and Tront, J. (2013). Securing static nodes in mobile-enabled systems using a network-layer moving target defense. In *2013 1st International Workshop on the Engineering of Mobile-Enabled Systems (MOBS)*, page 42–47. IEEE.

Grottke, M., Matias, R., and Trivedi, K. S. (2008). The fundamentals of software aging. In *2008 IEEE International conference on software reliability engineering workshops (ISSRE Wksp)*, pages 1–6. Ieee.

Grottke, M. and Trivedi, K. (2005). A classification of software faults. *Journal of Reliability Engineering Association of Japan*, 27(7):425–438.

Grottke, M. and Trivedi, K. S. (2007). Fighting bugs: Remove, retry, replicate, and rejuvenate. *Computer*, 40(2):107–109.

Guedes, E. A. C. (2019). *Availability and capacity modeling for virtual network functions based on redundancy and rejuvenation supported through live migration.* PhD thesis, Universidade Federal de Pernambuco.

Gupta, A. K., Zeng, W.-B., and Wu, Y. (2010). *Probability and statistical models: foundations for problems in reliability and financial mathematics.* Springer Science & Business Media.

Heiner, M., Lehrack, S., Gilbert, D., and Marwan, W. (2009). Extended stochastic petri nets for model-based design of wetlab experiments. In *Transactions on Computational Systems Biology XI*, pages 138–163. Springer.

Hill, M. D., Masters, J., Ranganathan, P., Turner, P., and Hennessy, J. L. (2019). On the spectre and meltdown processor security vulnerabilities. *IEEE Micro*, 39(2):9–19.

Hong, J. B. and Kim, D. S. (2015). Assessing the effectiveness of moving target defenses using security models. *IEEE Transactions on Dependable and Secure Computing*, 13(2):163–177.

Hosseinzadeh, S., Laurén, S., Rauti, S., Hyrynsalmi, S., Conti, M., and Leppänen, V. (2015). Obfuscation and diversification for securing cloud computing. In *International Workshop on Enterprise Security*, page 179–202. Springer.

Hsueh, M.-C., Tsai, T., and Iyer, R. (1997). Fault injection techniques and tools. *Computer*, 30(4):75–82.

Hu, J., Gu, J., Sun, G., and Zhao, T. (2010). A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In *2010 3rd International symposium on parallel architectures, algorithms and programming*, page 89–96. IEEE.

Huang, D. and Wu, H. (2017). *Mobile cloud computing: foundations and service models.* Morgan Kaufmann.

Huang, Q., Gao, F., Wang, R., and Qi, Z. (2011). Power consumption of virtual machine live migration in clouds. In *2011 Third International Conference on Communications and Mobile Computing*, page 122–125. IEEE.

Huang, Y., Kintala, C., Kolettis, N., and Fulton, N. D. (1995). Software rejuvenation: Analysis, module and applications. In *ftcs*, page 0381. IEEE.

Iooss, B. and Lemaître, P. (2015). A review on global sensitivity analysis methods. *Uncertainty management in simulation-optimization of complex systems: algorithms and applications*, pages 101–122.

Ishizaka, A. and Nemery, P. (2013). *Multi-criteria decision analysis: methods and software*. John Wiley & Sons.

Jahanshahloo, G. R., Lotfi, F. H., and Izadikhah, M. (2006). Extension of the topsis method for decision-making problems with fuzzy data. *Applied mathematics and computation*, 181(2):1544–1551.

Jain, R. (1990). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. John Wiley & Sons.

Jajodia, S., Ghosh, A. K., Subrahmanian, V., Swarup, V., Wang, C., and Wang, X. S. (2012). *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, volume 100. Springer.

Jajodia, S., Ghosh, A. K., Swarup, V., Wang, C., and Wang, X. S. (2011). *Moving target defense: creating asymmetric uncertainty for cyber threats*, volume 54. Springer Science & Business Media.

Jia, Q., Wang, H., Fleck, D., Li, F., Stavrou, A., and Powell, W. (2014). Catch me if you can: A cloud-enabled ddos defense. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, page 264–275. IEEE.

Jin, H., Li, Z., Zou, D., and Yuan, B. (2019). Dseom: A framework for dynamic security evaluation and optimization of mtd in container-based cloud. *IEEE Transactions on Dependable and Secure Computing*.

Jyotinagar, V. and Meshram, B. B. (2023). Developing an openstack environment: An exploration of experimentation and the diagnostic research. In *2023 2nd International Conference on Edge Computing and Applications (ICECAA)*, pages 42–49. IEEE.

Kahla, M., Azab, M., and Mansour, A. (2018). Secure, resilient, and self-configuring fog architecture for untrustworthy iot environments. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, page 49–54. IEEE.

Kargatzis, D., Sotiriadis, S., and Petrakis, E. G. (2017). Virtual machine migration in heterogeneous clouds: from openstack to vmware. In *2017 IEEE 38th Sarnoff Symposium*, page 1–6. IEEE.

Kashkoush, M. S., Azab, M., Attiya, G., and Abed, A. S. (2018). Online smart disguise: real-time diversification evading coresidency-based cloud attacks. *Cluster Computing*, page 1–16.

Kielland, S., Esmaeily, A., Kralevska, K., and Gligoroski, D. (2022). Secure service implementation with slice isolation and wireguard. In *2022 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, pages 148–153. IEEE.

Kim, D. S., Machida, F., and Trivedi, K. S. (2009). Availability modeling and analysis of a virtualized system. In *Dependable Computing, 2009. PRDC '09. 15th IEEE Pacific Rim International Symposium on*, page 365–371. IEEE.

Kleinrock, L. (1975). *Queueing systems. Volume I: theory.* wiley New York.

Knowles, W., Prince, D., Hutchison, D., Disso, J. F. P., and Jones, K. (2015). A survey of cyber security management in industrial control systems. *International journal of critical infrastructure protection*, 9:52–80.

Kot, M. (2003). The state explosion problem. *Retrieved May*, 18:2015.

Kounev, S. (2006). Performance modeling and evaluation of distributed component-based systems using queueing petri nets. *IEEE Transactions on Software Engineering*, 32(7):486–502.

Kourai, K. and Chiba, S. (2010). Fast software rejuvenation of virtual machine monitors. *IEEE Transactions on Dependable and Secure Computing*, 8(6):839–851.

Kuchárik, M. and Balogh, Z. (2019). Modeling of uncertainty with petri nets. In *Asian Conference on Intelligent Information and Database Systems*, page 499–509. Springer.

Kukrál, T., Kozák, M., Hégr, T., and Boháč, L. (2015). Vm migration measurement and failure detection. In *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*, page 285–288. IEEE.

Küngas, P. (2005). Petri net reachability checking is polynomial with optimal abstraction hierarchies. In *International Symposium on Abstraction, Reformulation, and Approximation*, pages 149–164. Springer.

Kurra, H., Al-Nashif, Y., and Hariri, S. (2013). Resilient cloud data storage services. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, page 17. ACM.

KVM (n.d.). Kvm. `https://www.linux-kvm.org/page/Main_Page`. Accessed 2023-01-04.

Le Goues, C., Nguyen-Tuong, A., Chen, H., Davidson, J. W., Forrest, S., Hiser, J. D., Knight, J. C., and Van Gundy, M. (2013). Moving target defenses in the helix self-regenerative architecture. In *Moving target defense II*, page 117–149. Springer.

Lei, C., Zhang, H.-Q., Wan, L.-M., Liu, L., and Ma, D.-h. (2018). Incomplete information markov game theoretic approach to strategy generation for moving target defense. *Computer Communications*, 116:184–199.

Leslie, D., Sherfield, C., and Smart, N. P. (2015). Threshold flipthem: When the winner does not need to take all. In *International Conference on Decision and Game Theory for Security*, page 74–92. Springer.

Leutenegger, S. T. and Dias, D. (1993). A modeling study of the tpc-c benchmark. *ACM Sigmod Record*, 22(2):22–31.

Levitin, G., Xing, L., and Ben-Haim, H. (2018). Optimizing software rejuvenation policy for real time tasks. *Reliability Engineering & System Safety*, 176:202–208.

Li, Z., Sen, T., Shen, H., and Chuah, M. C. (2020). Impact of memory dos attacks on cloud applications and real-time detection schemes. In *49th International Conference on Parallel Processing-ICPP*, page 1–11.

Liu, H., Xu, C.-Z., Jin, H., Gong, J., and Liao, X. (2011). Performance and energy modeling for live migration of virtual machines. In *Proceedings of the 20th international symposium on High performance distributed computing*, page 171–182. ACM.

Liu, J., Zhou, J., and Buyya, R. (2015). Software rejuvenation based fault tolerance scheme for cloud applications. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 1115–1118. IEEE.

Liu, L., Wang, A., Zang, W., Yu, M., Xiao, M., and Chen, S. (2018). Shuffler: Mitigate cross-vm side-channel attacks via hypervisor scheduling. In *International Conference on Security and Privacy in Communication Systems*, page 491–511. Springer.

Liu, P., Yang, Z., Song, X., Zhou, Y., Chen, H., and Zang, B. (2008). Heterogeneous live migration of virtual machines. In *International Workshop on Virtualization Technology (IWVT'08).*

Longo, F., Ghosh, R., Naik, V. K., and Trivedi, K. S. (2011). A scalable availability model for infrastructure-as-a-service cloud. In *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, page 335–346. IEEE.

Luo, Y.-B., Wang, B.-S., Cai, G.-L., Wang, X.-F., and Zhang, B.-F. (2016). High performance low latency network address and port hopping mechanism based on netfilter. In *International Conference on Intelligent and Interactive Systems and Applications*, page 239–244. Springer.

Lysenko, S., Savenko, O., Bobrovnikova, K., and Kryshchuk, A. (2018). Self-adaptive system for the corporate area network resilience in the presence of botnet cyberattacks. In *International Conference on Computer Networks*, page 385–401. Springer.

Ma, D., Lei, C., Wang, L., Zhang, H., Xu, Z., and Li, M. (2016). A self-adaptive hopping approach of moving target defense to thwart scanning attacks. In *International Conference on Information and Communications Security*, page 39–53. Springer.

Machida, F., Kim, D. S., and Trivedi, K. S. (2010). Modeling and analysis of software rejuvenation in a server virtualized system. In *2010 IEEE Second International Workshop on Software Aging and Rejuvenation*, pages 1–6. IEEE.

Machida, F., Kim, D. S., and Trivedi, K. S. (2013). Modeling and analysis of software rejuvenation in a server virtualized system with live vm migration. *Performance Evaluation*, 70(3):212–230. Special Issue on Software Aging and Rejuvenation.

Machida, F., Xiang, J., Tadano, K., and Maeno, Y. (2012). Aging-related bugs in cloud computing software. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops*, page 287–292. IEEE.

Maciel, P., Dantas, J., Melo, C., Pereira, P., Oliveira, F., Araujo, J., and Matos, R. (2021). A survey on reliability and availability modeling of edge, fog, and cloud computing. *Journal of Reliable Intelligent Environments*, pages 1–19.

Maciel, P., Matos, R., Silva, B., Figueiredo, J., Oliveira, D., Fé, I., Maciel, R., and Dantas, J. (2017). Mercury: Performance and dependability evaluation of systems with exponential, expolynomial, and general distributions. In *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, page 50–57. IEEE.

Maciel, P., Trivedi, K., and Kim, D. (2010). Dependability modeling in: Performance and dependability in service computing: Concepts, techniques and research directions. *Hershey: IGI Global, Pennsylvania, USA*, 13.

Maciel, P. R. M. (2023). *Performance, reliability, and availability evaluation of computational systems, Volume 2: Reliability, availability modeling, measuring, and data analysis.* CRC Press.

Maciel, R., Araujo, J., Melo, C., Dantas, J., and Maciel, P. (2018). Impact assessment of multi-threats in computer systems using attack tree modeling. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, page 2448–2453. IEEE.

Mainkar, V., Choi, H., and Trivedi, K. (1993). Sensitivity analysis of markov regenerative stochastic petri nets. In *Proceedings of 5th International Workshop on Petri Nets and Performance Models*, pages 180–181. IEEE.

Manadhata, P. K. and Wing, J. M. (2010). An attack surface metric. *IEEE Transactions on Software Engineering*, 37(3):371–386.

Marsan, M. A. (1988). Stochastic petri nets: an elementary introduction. In *European Workshop on Applications and Theory in Petri Nets*, page 1–29. Springer.

Marsan, M. A., Balbo, G., Conte, G., Donatelli, S., and Franceschinis, G. (1998). *Modelling with generalized stochastic Petri nets.* ACM New York, NY, USA.

Matias, R., Beicker, I., Leitão, B., and Maciel, P. R. (2010). Measuring software aging effects through os kernel instrumentation. In *2010 IEEE Second International Workshop on Software Aging and Rejuvenation*, pages 1–6. IEEE.

Matos, R., Araujo, J., Alves, V., and Maciel, P. (2012a). Characterization of software aging effects in elastic storage mechanisms for private clouds. In *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on*, page 293–298. IEEE Computer Society.

Matos, R., Araujo, J., Alves, V., and Maciel, P. (2012b). Experimental evaluation of software aging effects in the eucalyptus elastic block storage. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, page 1103–1108. IEEE.

Matos, R. d. S., Maciel, P. R., Machida, F., Kim, D. S., and Trivedi, K. S. (2012c). Sensitivity analysis of server virtualized system availability. *IEEE Transactions on Reliability*, 61(4):994–1006.

Maziku, H. and Shetty, S. (2014). Towards a network aware vm migration: Evaluating the cost of vm migration in cloud data centers. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, page 114–119. IEEE.

Melo, M., Araujo, J., Matos, R., Menezes, J., and Maciel, P. (2013a). Comparative analysis of migration-based rejuvenation schedules on cloud availability. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, page 4110–4115. IEEE.

Melo, M., Maciel, P., Araujo, J., Matos, R., and Araújo, C. (2013b). Availability study on cloud computing environments: Live migration as a rejuvenation mechanism. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, page 1–6. IEEE.

Mendonça, J., Cho, J.-H., Moore, T. J., Nelson, F. F., Lim, H., Zimmermann, A., and Kim, D. S. (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*, page 1180–1189.

Meyer, J. F. (1992). Performability: a retrospective and some pointers to the future. *Performance evaluation*, 14(3-4):139–156.

MITRE (2023). Defense evasion.

Moody, W. C., Hu, H., and Apon, A. (2014). Defensive maneuver cyber platform modeling with stochastic petri nets. In *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, page 531–538. IEEE.

Moon, S.-J., Sekar, V., and Reiter, M. K. (2015). Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. In *Proceedings of the 22nd acm sigsac conference on computer and communications security*, page 1595–1606. ACM.

Mosberger, D. and Jin, T. (1998). Httperf - a tool for measuring web server performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):31–37.

Muppala, J. K. and Lin, C. (1996). Dependability analysis of large-scale distributed systems using stochastic petri nets. In *1996 IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and Systems (Cat. No. 96CH35929)*, volume 4, pages 3033–3038. IEEE.

Muppala, J. K. and Trivedi, K. S. (1991). Composite performance and availability analysis using a hierarchy of stochastic reward nets. *Computer Performance Evaluation, Modelling Techniques and Tools*, page 335–350.

Nelson, W. (1980). Accelerated life testing-step-stress models and data analyses. *IEEE transactions on reliability*, 29(2):103–108.

Neto, A. A. and Vieira, M. (2011). To benchmark or not to benchmark security: That is the question. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, page 182–187. IEEE.

Nguyen, M., Pal, A., and Debroy, S. (2018). Whack-a-mole: Software-defined networking driven multi-level ddos defense for cloud environments. In *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, page 493–501. IEEE.

Nguyen, T. A., Min, D., and Choi, E. (2020). A hierarchical modeling and analysis framework for availability and security quantification of iot infrastructures. *Electronics*, 9(1):155.

Nguyen, T. A., Min, D., Choi, E., and Tran, T. D. (2019). Reliability and availability evaluation for cloud data center networks using hierarchical models. *IEEE Access*, 7:9273–9313.

Nicol, D. M., Sanders, W. H., and Trivedi, K. S. (2004). Model-based evaluation: from dependability to security. *IEEE Transactions on dependable and secure computing*, 1(1):48–65.

Oberheide, J., Cooke, E., and Jahanian, F. (2008). Exploiting live virtual machine migration. *BlackHat DC Briefings*.

Okamura, H., Yamamoto, K., and Dohi, T. (2014). Transient analysis of software rejuvenation policies in virtualized system: Phase-type expansion approach. *Quality Technology & Quantitative Management*, 11(3):335–351.

Okhravi, H., Rabe, M., Mayberry, T., Leonard, W., Hobson, T., Bigelow, D., and Streilein, W. (2013). Survey of cyber moving target techniques. Technical report, MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB.

Oliveira, F., Araujo, J., Matos, R., Lins, L., Rodrigues, A., and Maciel, P. (2020). Experimental evaluation of software aging effects in a container-based virtualization platform. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 414–419. IEEE.

Pacheco, J., Tunc, C., and Hariri, S. (2016). Design and evaluation of resilient infrastructures systems for smart cities. In *2016 IEEE International Smart Cities Conference (ISC2)*, page 1–6. IEEE.

Paing, A. M. M. and Thein, N. L. (2012). Stochastic reward nets model for time based software rejuvenation in virtualized environment. *International Journal of Computer Science and Telecommunications*, 3(1):1–10.

Pan, J. (1999). Software testing. *Dependable Embedded Systems*, 5(2006):1.

Paolieri, M., Biagi, M., Carnevali, L., and Vicario, E. (2019). The oris tool: quantitative evaluation of non-markovian systems. *IEEE Transactions on Software Engineering*, 47(6):1211–1225.

Parrend, P., Navarro, J., Guigou, F., Deruyver, A., and Collet, P. (2018). Foundations and applications of artificial intelligence for zero-day and multi-step attack detection. *EURASIP Journal on Information Security*, 2018:1–21.

Pasupulati, R. P. and Shropshire, J. (2016). A diversity defense for cloud computing systems. In *SoutheastCon, 2016*, page 1–7. IEEE.

Patil, R. and Modi, C. (2019). An exhaustive survey on security concerns and solutions at different components of virtualization. *ACM Computing Surveys (CSUR)*, 52(1):12.

Pavić, Z. and Novoselac, V. (2013). Notes on topsis method. *International Journal of Research in Engineering and Science*, 1(2):5–12.

Pawlewski, P. (2012). *Petri nets: manufacturing and computer science*. BoD–Books on Demand.

Peng, W., Li, F., Huang, C.-T., and Zou, X. (2014). A moving-target defense strategy for cloud-based services with heterogeneous and dynamic attack surfaces. In *Communications (ICC), 2014 IEEE International Conference on*, page 804–809. IEEE.

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

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic mapping studies in software engineering. In *EASE*, volume 8, page 68–77.

Petersen, K., Vakkalanka, S., and Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18.

Petri, C. A. (1962). *Kommunikation mit Automaten.* PhD thesis, Universitat Hamburg.

PICUS (2023).

Pietrantuono, R. and Russo, S. (2019). A survey on software aging and rejuvenation in the cloud. *Software Quality Journal*, page 1–32.

Pingree, L. (2023).

Pluralsight (2023).

Portnoy, M. (2012). *Virtualization essentials*, volume 19. John Wiley & Sons.

Raj, S., Mangal, N., Savitha, S., and Salvi, S. (2020). Virtual machine migration in heterogeneous clouds-a practical approach. In *2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, page 1–6. IEEE.

Ross, R., McEvilley, M., and Oren, J. C. (2016). Systems security engineering: Considerations for a multidisciplinary approach in the engineering of trustworthy secure systems. In *NIST Special Publication 800-160.*

Ross, R. S., Winstead, M., and McEvilley, M. (2022). Engineering trustworthy secure systems.

Royce, W. W. (1970). Managing the development of large software systems. In *proceedings of IEEE WESCON*, volume 26. Los Angeles.

Sadiku, M. N., Musa, S. M., and Momoh, O. D. (2014). Cloud computing: opportunities and challenges. *IEEE potentials*, 33(1):34–36.

Salfner, F., Tröger, P., and Polze, A. (2011). Downtime analysis of virtual machine live migration. In *The Fourth International Conference on Dependability (DEPEND 2011). IARIA*, page 100–105.

Sarwar, M. M. S., Rivera, J. J. D., Muhammad, A., and Song, W.-C. (2022). Geneve@ tein: A sophisticated tunneling technique for communication between openstack-based multiple clouds at tein. In *2022 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–4. IEEE.

Schatz, B. (2007). Bodysnatcher: Towards reliable volatile memory acquisition by software. *digital investigation*, 4:126–134.

Schroeder, B. and Gibson, G. A. (2007). Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you? In *FAST*, volume 7, page 1–16.

Sengupta, S., Chowdhary, A., Sabur, A., Alshamrani, A., Huang, D., and Kambhampati, S. (2020). A survey of moving target defenses for network security. *IEEE Communications Surveys & Tutorials.*

Sianipar, J., Sukmana, M., and Meinel, C. (2018). Moving sensitive data against live memory dumping, spectre and meltdown attacks. In *2018 26th International Conference on Systems Engineering (ICSEng)*, page 1–8. IEEE.

Siddiqui, S., Darbari, M., Yagyasen, D., et al. (2020). Modelling and simulation of queuing models through the concept of petri nets. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal.*

Song, F., Zhou, Y.-T., Wang, Y., Zhao, T.-M., You, I., and Zhang, H.-K. (2019). Smart collaborative distribution for privacy enhancement in moving target defense. *Information Sciences*, 479:593–606.

Soussi, W., Christopoulou, M., Xilouris, G., and Gür, G. (2021). Moving target defense as a proactive defense element for beyond 5g. *IEEE Communications Standards Magazine*, 5(3):72–79.

Stafford, V. (2020). Zero trust architecture. *NIST special publication*, 800:207.

Strunk, A. (2012). Costs of virtual machine live migration: A survey. In *2012 IEEE Eighth World Congress on Services*, page 323–329. IEEE.

Subashini, S. and Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, 34(1):1–11.

Tatam, M., Shanmugam, B., Azam, S., and Kannoorpatti, K. (2021). A review of threat modelling approaches for apt-style attacks. *Heliyon*, 7(1).

Thebeau II, D., Reidy, B., Valerdi, R., Gudagi, A., Kurra, H., Al-Nashif, Y., Hariri, S., and Sheldon, F. (2014). Improving cyber resiliency of cloud application services by applying software behavior encryption (sbe). *Procedia Computer Science*, 28:62–70.

Thein, T. and Park, J. S. (2009). Availability analysis of application servers using software rejuvenation and virtualization. *Journal of computer science and technology*, 24(2):339–346.

Theisen, C., Munaiah, N., Al-Zyoud, M., Carver, J. C., Meneely, A., and Williams, L. (2018). Attack surface definitions: A systematic literature review. *Information and Software Technology*, 104:94–103.

Torquato, M., Araujo, J., Umesh, I., and Maciel, P. (2018a). Sware: A methodology for software aging and rejuvenation experiments. *Journal of Information Systems Engineering & Management*, 3(2):15.

Torquato, M., Guedes, E., Maciel, P., and Vieira, M. (2019a). A hierarchical model for virtualized data center availability evaluation. In *15th European Dependable Computing Conference.*

Torquato, M., Maciel, P., Araujo, J., and Umesh, I. (2017). An approach to investigate aging symptoms and rejuvenation effectiveness on software systems. In *Information Systems and Technologies (CISTI), 2017 12th Iberian Conference on*, page 1–6. IEEE.

Torquato, M., Maciel, P., and Vieira, M. (2019b). A model for availability and security risk evaluation for systems with vmm rejuvenation enabled by vm migration scheduling. *IEEE Access*, 7:138315–138326.

Torquato, M., Maciel, P., and Vieira, M. (2020a). Availability and reliability modeling of vm migration as rejuvenation on a system under varying workload. *Software Quality Journal*, 28:59–83.

Torquato, M., Maciel, P., and Vieira, M. (2020b). Security and availability modeling of vm migration as moving target defense. In *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*, page 50–59. IEEE.

Torquato, M., Maciel, P., and Vieira, M. (2021a). Analysis of vm migration scheduling as moving target defense against insider attacks. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, page 194–202.

Torquato, M., Maciel, P., and Vieira, M. (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)*, page 357–366. IEEE.

Torquato, M., Maciel, P., and Vieira, M. (2022a). Model-based performability and dependability evaluation of a system with vm migration as rejuvenation in the presence of bursty workloads. *Journal of Network and Systems Management*, 30(1):1–33.

Torquato, M., Maciel, P., and Vieira, M. (2022b). 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)*, pages 205–216. IEEE.

Torquato, M., Torquato, L., Maciel, P., and Vieira, M. (2019c). Iaas cloud availability planning using models and genetic algorithms. In *2019 9th Latin-American Symposium on Dependable Computing (LADC)*, page 1–10. IEEE.

Torquato, M., Umesh, I., and Maciel, P. (2018b). Models for availability and power consumption evaluation of a private cloud with vmm rejuvenation enabled by vm live migration. *The Journal of Supercomputing*, 74(9):4817–4841.

Torquato, M. and Vieira, M. (2018). Interacting srn models for availability evaluation of vm migration as rejuvenation on a system under varying workload. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, page 300–307. IEEE.

Torquato, M. and Vieira, M. (2019). An experimental study of software aging and rejuvenation in dockerd. In *15th European Dependable Computing Conference*, page 1–6. IEEE.

Torquato, M. and Vieira, M. (2020). Moving target defense in cloud computing: A systematic mapping study. *Computers & Security*, 92:101742.

Torquato, M. and Vieira, M. (2021). Vm migration scheduling as moving target defense against memory dos attacks: An empirical study. In *2021 IEEE Symposium on Computers and Communications (ISCC)*, page 1–6. IEEE.

Triantaphyllou, E. (2000). Multi-criteria decision making methods. In *Multi-criteria decision making methods: A comparative study*, page 5–21. Springer.

Trivedi, K. S. (1982). *Probability and statistics with reliability, queuing, and computer science applications*, volume 13. Wiley Online Library.

Trivedi, K. S. (2008). *Probability & statistics with reliability, queuing and computer science applications*. John Wiley & Sons.

Trivedi, K. S. and Bobbio, A. (2017). *Reliability and Availability Engineering: Modeling, Analysis, and Applications*. Cambridge University Press.

Trivedi, K. S., Kim, D. S., Roy, A., and Medhi, D. (2009). Dependability and security models. In *Design of Reliable Communication Networks, 2009. DRCN 2009. 7th International Workshop on*, page 11–20. IEEE.

Trivedi, K. S., Vaidyanathan, K., and Goseva-Popstojanova, K. (2000). Modeling and analysis of software aging and rejuvenation. In *Proceedings 33rd Annual Simulation Symposium (SS 2000)*, page 270–279. IEEE.

Umesh, I., Srinivasan, G., and Torquato, M. (2017). Software aging forecasting using time series model. *Indonesian Journal of Electrical Engineering and Computer Science*, 7(3):839–845.

U.S. Department Homeland of Security (2020), U. D. H. o. S. . (2020). Moving target defense. `https://www.dhs.gov/science-and-technology/csd-mtd`. Accessed: 2018-12-09.

Vaidyanathan, K. and Trivedi, K. S. (2001). Extended classification of software faults based on aging. In *Fast Abstract, Int. Symp. Software Reliability Eng., Hong Kong*.

Vaidyanathan, K. and Trivedi, K. S. (2005). A comprehensive model for software rejuvenation. *IEEE Transactions on Dependable and Secure Computing*, 2(2):124–137.

Valmari, A. (1996). The state explosion problem. In *Advanced Course on Petri Nets*, pages 429–528. Springer.

Vieira, M. and Madeira, H. (2005). Towards a security benchmark for database management systems. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, page 592–601. IEEE.

Villarreal-Vasquez, M., Bhargava, B., Angin, P., Ahmed, N., Goodwin, D., Brin, K., and Kobes, J. (2017). An mtd-based self-adaptive resilience approach for cloud systems. In *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*, page 723–726. IEEE.

Vinícius, L., Rodrigues, L., Torquato, M., and Silva, F. A. (2022). Docker platform aging: a systematic performance evaluation and prediction of resource consumption. *The Journal of Supercomputing*, page 1–31.

von Kistowski, J., Eismann, S., Schmitt, N., Bauer, A., Grohmann, J., and Kounev, S. (2018). Teastore: A micro-service reference application for benchmarking, modeling and resource management research. In *Proceedings of the 26th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS '18.

Voorsluys, W., Broberg, J., Venugopal, S., and Buyya, R. (2009). Cost of virtual machine live migration in clouds: A performance evaluation. In *IEEE International Conference on Cloud Computing*, page 254–265. Springer.

Wahab, O. A., Bentahar, J., Otrok, H., and Mourad, A. (2019). Resource-aware detection and defense system against multi-type attacks in the cloud: Repeated bayesian stackelberg game. *IEEE Transactions on Dependable and Secure Computing.*

Wang, D., Xie, W., and Trivedi, K. S. (2007). Performability analysis of clustered systems with rejuvenation under varying workload. *Performance Evaluation*, 64(3):247–265.

Wang, H., Jia, Q., Fleck, D., Powell, W., Li, F., and Stavrou, A. (2014). A moving target ddos defense mechanism. *Computer Communications*, 46:10–21.

Wang, H., Li, F., and Chen, S. (2016). Towards cost-effective moving target defense against ddos and covert channel attacks. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, page 15–25. ACM.

Wang, Y., Li, J., Meng, K., Lin, C., and Cheng, X. (2013). Modeling and security analysis of enterprise network using attack–defense stochastic game petri nets. *Security and Communication Networks*, 6(1):89–99.

Wang, Y., Yu, M., Li, J., Meng, K., Lin, C., and Cheng, X. (2012). Stochastic game net and applications in security analysis for enterprise network. *International Journal of Information Security*, 11(1):41–52.

Xen (n.d.). Xen project. `https://xenproject.org/`. Accessed 2023-01-04.

Yadav, T. and Rao, A. M. (2015). Technical aspects of cyber kill chain. In *Security in Computing and Communications: Third International Symposium, SSCC 2015, Kochi, India, August 10-13, 2015. Proceedings 3*, pages 438–452. Springer.

Yang, C., Guo, Y.-f., Hu, H.-c., Wang, Y.-w., Tong, Q., and Li, L.-s. (2019). Driftor: mitigating cloud-based side-channel attacks by switching and migrating multi-executor virtual machines. *Frontiers of Information Technology & Electronic Engineering*, 20(5):731–748.

Yang, Y. and Cheng, L. (2018). An sdn-based mtd model. *Concurrency and Computation: Practice and Experience*, page e4897.

Zhang, M., Wang, L., Jajodia, S., Singhal, A., and Albanese, M. (2016a). Network diversity: a security metric for evaluating the resilience of networks against zero-day attacks. *IEEE Transactions on Information Forensics and Security*, 11(5):1071–1086.

Zhang, S. (2012). Deep-diving into an easily-overlooked threat: Inter-vm attacks. Technical report, Technical Report). Manhattan, Kansas: Kansas State University.

Zhang, T. and Lee, R. B. (2017). Host-based dos attacks and defense in the cloud. In *Proceedings of the Hardware and Architectural Support for Security and Privacy*, pages 1–8.

Zhang, T., Zhang, Y., and Lee, R. B. (2016b). Memory dos attacks in multi-tenant clouds: Severity and mitigation. *arXiv preprint arXiv:1603.03404*.

Zhang, T., Zhang, Y., and Lee, R. B. (2017). Dos attacks on your memory in cloud. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, page 253–265.

Zhang, Y., Li, M., Bai, K., Yu, M., and Zang, W. (2012). Incentive compatible moving target defense against vm-colocation attacks in clouds. In *IFIP International Information Security Conference*, page 388–399. Springer.

Zhuang, R., DeLoach, S. A., and Ou, X. (2014). Towards a theory of moving target defense. In *Proceedings of the First ACM Workshop on Moving Target Defense*, page 31–40. ACM.

Zimmermann, A. (2017). Modelling and performance evaluation with timenet 4.4. In *International Conference on Quantitative Evaluation of Systems*, page 300–303. Springer.

Appendixes

Appendix A: MTD in the Cloud: A Systematic Mapping Study

In the following pages, we present the full content of our systematic map study of Moving Target Defense in cloud computing.

Computers & Security 92 (2020) 101742

Contents lists available at ScienceDirect

Computers & Security

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

Moving target defense in cloud computing: A systematic mapping study

Matheus Torquato[a,b,*], Marco Vieira[a]

[a] Department of Informatics Engineering, University of Coimbra, Coimbra, Portugal
[b] Federal Institute of Alagoas, Campus Arapiraca, Arapiraca, Brazil

ARTICLE INFO

Article history:
Received 30 October 2019
Revised 25 January 2020
Accepted 1 February 2020
Available online 3 February 2020

Keywords:
Moving target defense
Cloud computing
Systematic mapping
Cyber security
Network security

ABSTRACT

Moving Target Defense (MTD) consists of applying system reconfiguration (e.g., VM migration, IP shuffling) to dynamically change the available attack surface. MTD makes use of reconfiguration to confuse attackers and nullify their knowledge about the system state. It also can be used as an attack reaction (e.g., using Virtual Machine (VM) migration to move VMs away from a compromised host). Thus, MTD seems to be a promising technique to tackle some of the cloud computing security challenges. In this systematic mapping study, we aim to investigate the current research state of Moving Target Defense in the cloud computing context, and to identify potential research gaps in the literature. Considering five major scientific databases in the computer science domain, we collected 224 papers related to the area. After disambiguation and filtering, we selected 95 papers for analysis. The outcome of such analysis offers a comprehensive overview of the current research. We can highlight some relevant research opportunities. First, only a few works present advances in the theoretical field of Moving Target Defense in cloud computing. Second, the proposal and evaluation of multi-layer Moving Target Defense mechanisms is still an open problem. Thirdly, there is a need for frameworks to support MTD evaluation, which may include a benchmark for comparing alternative MTD strategies. Finally, the study of potential impacts of Moving Target Defense in context-oriented clouds is a barely explored topic.

© 2020 Elsevier Ltd. All rights reserved.

1. Introduction

Cloud computing is a computing paradigm that enables ubiquitous, on-demand network access to a configurable set of resources (e.g., computing, storage, network, and services) (Mell et al., 2011). Cloud computing reduces the up-front cost for its users, allowing a gradual increase or decrease of resources allocation, adapting the available computing power to the existing needs (Armbrust et al., 2010). Due to its characteristics, many companies and organizations rely on cloud computing to run their applications.

Existing surveys show that cloud computing security is at the top of users' concerns (RightScale, 2018). Besides that, cloud computing security and privacy persist as significant research challenges (Krutz and Vines, 2010; Ren et al., 2012).

In this context, Moving Target Defense (MTD) has emerged as a low-cost technique to improve cloud computing resiliency and security. The United States Department of Homeland Security defines MTD as *the concept of controlling change across multiple system dimensions in order to increase uncertainty and apparent complexity for attackers, reduce their window of opportunity and increase the costs of their probing and attack efforts. MTD assumes that perfect security is unattainable. Given that starting point, and the assumption that all systems are compromised, research in MTD focuses on enabling the continued safe operation in a compromised environment and to have systems that are defensible rather than perfectly secure* (hls, 2018).

In this paper, we aim to investigate the current research state of MTD in cloud computing. To achieve this goal, we adopted a systematic mapping approach (Petersen et al., 2008; 2015). From systematic maps, we can understand the focus of community research efforts and also perceive what areas are barely explored. The systematic mapping process aims to reduce bias in papers classification by applying a well-defined methodology. Mapping studies provide a good overview of a research topic and are useful before starting more deep research works (Kitchenham et al., 2010).

The current literature provides comprehensive surveys and review papers on Moving Target Defense. Lei et al. (2018a) focus on the characteristics of moving target defense techniques. Cai et al. (2016b) presents a comprehensive survey on Moving Target Defense. Besides these two works, two relevant surveys in the

* Corresponding author.
E-mail addresses: mdmelo@dei.uc.pt, matheus.torquato@ifal.edu.br (M. Torquato), mvieira@dei.uc.pt (M. Vieira).

https://doi.org/10.1016/j.cose.2020.101742
0167-4048/© 2020 Elsevier Ltd. All rights reserved.

area were recently published (Sengupta et al., 2019b; Zheng and Namin, 2019). Two books from Jajodia et al. in this same topic can also be found in the literature (Jajodia et al., 2012; 2011). Different from all these works, our paper is focused on the cloud computing context. Besides that, instead of surveying the papers, our goal is to use a structured approach (i.e., systematic mapping) to provide a comprehensive overview of MTD in the cloud.

Our analysis shows a growing interest in MTD in the cloud computing context. Current research is focused on proposing and evaluating MTD techniques based on environment reshuffle (like VM migration or dynamic network reconfiguration). There are few papers regarding MTD theory, and there is a need for unified methodologies to support MTD evaluation and comparison. Also, the combination of MTD strategies requires further research. In practice, this paper presents a comprehensive overview of MTD in cloud computing research. The diagrams and charts presented intend to provide useful information for the community on the current state of the research in the area.

The rest of this paper is organized as follows. Section 2 presents some background about cloud computing security. Section 3 presents motivations for conducting this research. Section 4 introduces the methodology adopted for map construction. Section 5 presents the results in terms of the papers collected from the scientific databases. Section 6 presents the maps obtained. Section 7 discusses the maps and the main observations. Finally, Section 8 concludes the paper.

2. Cloud computing security

Cloud computing architecture and intrinsic characteristics raise several security concerns. For example, Popović and Hocenski (2010) show that the cloud security concerns that range from the *location of the encryption and decryption keys* to the *auditability of VMs*. Due to the relevance of this question, cloud computing security usually appear as a significant research challenge (Buyya et al., 2018; Ren et al., 2012).

One of the challenging problems for cloud security is the asymmetric advantage of attackers over defenders. Attackers can perform a series of actions (e.g., repeated attacks, vulnerability analysis) until they achieve their goal. So, the attackers can try to exploit a specific system vulnerability while the defenders have to protect all the possible attack venues (Cai et al., 2016b). Besides that, the generally static nature of data centers facilitates the attacker to obtain enough information to improve the chance of attack success.

Moving Target Defense applies dynamic environment reconfiguration capable of confusing attackers or reacting to an attack in progress. For example, to minimize the attackers' asymmetric advantage, we can apply a dynamic network address shuffle (Fleck et al., 2018). Moreover, as an attack reaction, we can use VM migration to save benign clients from the security attack (Jia et al., 2014).

3. Motivation

Fig. 1 shows the Google search trends for the term "Moving Target Defense" in the last ten years. It is possible to notice a growing interest in the field in the last ten years. As we will show in this paper, the same occurs in the number of papers published in the last years. Due to the urgency in the development of innovative techniques to protect cloud computing systems, the MTD attracted significant attention because of its flexibility. As mentioned earlier, it is possible to deploy MTD to confuse attackers and also to react to attacks in progress.

However, compared to the established defense mechanisms as firewalls and Intrusion Detection Systems (IDS) (Bonguet and Bellaiche, 2017), MTD technique is a newer technology and the related

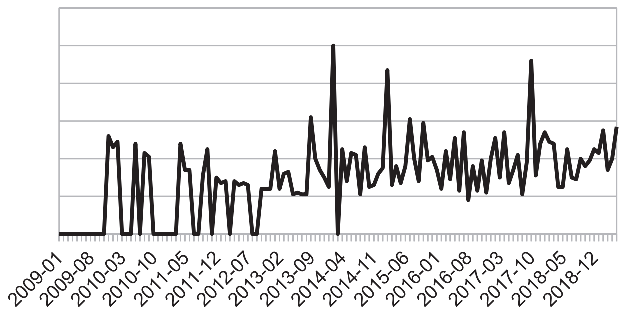


Fig. 1. Google trends - "Moving Target Defense" - Jan 1, 2009 to July 5, 2019.

research in the area is incipient. MTD draws attention because of the idea of accepting imperfect security (hls, 2018) and applying dynamic changes in the environment to protect it. Nevertheless, there are challenges in the MTD deployment as how to apply the dynamic changes, when to apply them, and how to evaluate their effectiveness (more details of MTD in the cloud research opportunities in Section 7).

Recent papers tackled such challenges. For example, the papers (Alavizadeh et al., 2019; Hong et al., 2018) presented evaluation mechanisms for MTD. Sengupta et al. (2019a) leveraged from a Markov Game to provide optimal strategies for security resources placement. Das et al. (2019) used a process to obfuscate VM migration in the cloud environment, reducing the attacker's chance to recognize it. Peng et al. (2014b) propose MTD techniques based on polymorphism, rapid provisioning of defenses, and defensive mechanisms to facilitate unauthorized access detection.

The usual first step of research projects (as Ph.D. studies) is to understand the state-of-the-art in the related field. The systematic mapping aims to bring an overview of such state-of-the-art, usually focusing on specific aspects of the literature. Different from systematic reviews, which discuss the related papers in detail, the systematic mappings focus on specific aspects and perform a more concise analysis of the papers. The main advantage of the systematic mappings is to usually provide faster results, as they focus on the desired papers' aspects. Besides that, the visual data (maps) facilitates the understanding of the state of the literature.

The current literature of MTD in the cloud lacks a systematic mapping. This work intends to fill this gap proposing a mapping from the last ten years of the research in the area. The scientific may leverage our work for understanding the current state of the MTD in cloud research, as well as on the definition of future research lines.

4. Methods

The systematic mapping process is based on the work by Petersen et al. (2008). Fig. 2 presents the process steps and outcomes, which will be described in the following sections.

4.1. Research questions

The main goal of this systematic mapping study is to provide an overview of recent research on Moving Target Defense mechanisms in cloud computing environments. To reach this goal, we propose three generic research questions:

RQ1: How has the frequency of publication on moving target defense in cloud computing changed in the last ten years?
RQ2: In which forums have research on moving target defense on cloud computing been published?
RQ3: What are the most researched techniques for moving target defense on cloud computing?

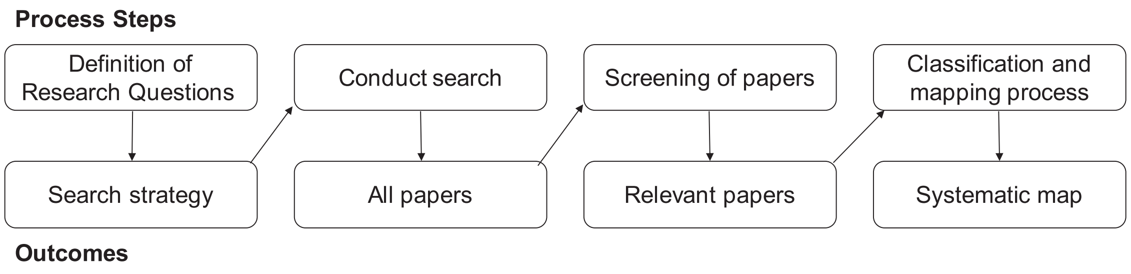


Fig. 2. Systematic mapping process.

The first two questions aim to give an overview of publication frequency and relevant forums of publication. The third question, which is the most important, seeks to offer a bigger picture of the relations of MTD techniques and cloud computing.

We also analyzed the considered papers to find the most prominent authors from the field (i.e., authors with more publications) and to verify the existence of related research of the same set of authors (i.e., identifying papers and their potential extensions).

4.2. Scientific databases and search strategy

We selected five relevant online computer science computer databases to find the papers related to MTD in the cloud. We decided to use them instead of generic databases, such as *Scopus* and *Web of Science*, because some papers may be missing from these generic platforms (e.g., early access papers). We neglected the *Google Scholar* database because it indexes non-peer reviewed papers. The list of the selected databases is the following:

- ACM Digital Library;
- IEEE Xplore Digital Library;
- ScienceDirect;
- SpringerLink;
- Online Wiley Library.

The search string is the following:

("moving target defense") AND ("cloud")

The first part of the search string is related to moving target defense, and the second is related to cloud computing. We decided to neglect the use of the acronym "MTD" in the search process because it is an ambiguous acronym. MTD can be related to Moving Target Defense, Managing Technical Debt, Mixture Transition Distribution, Mean Texture Depth, or Machine-Type Device. Thus, we assume that relevant works have at least one direct mention of the term "moving target defense".

In the second part of the search string, we decided to use only the word "cloud" instead of its derivatives (e.g., IaaS, Cloud Computing, Cloud environment). With this, we assume that the papers within our research scope mention the word "cloud" directly.

4.3. Screening of papers

We determined the inclusion and exclusion criteria to filter the search results. Our goal is to select the relevant research of MTD in the cloud in the last ten years (i.e., as in *RQ1*). Thus, our paper selection process intends to cover the peer-reviewed papers about the subject. The research on MTD in the cloud is relatively new when compared to consolidated areas (as performance evaluation, for example), meaning that incipient research may have been published in small conferences or workshops. Therefore, we do not apply filters related to the rank/quality of the considered venues of publication. Finally, we also removed the survey papers from the search results as we intend to analyze the individual contributions from the papers instead of a compilation of papers. The adopted filtering strategy is summarized below.

- *Inclusion criteria*
  - Research papers about moving target defense techniques applied in cloud computing.
    - ∗ Papers with direct reference to cloud computing and moving target defense in their titles, abstracts, keywords or introduction.
  - Papers published in journals, magazines or conference proceedings.
- *Exclusion criteria*
  - Papers published before 2009.
  - Papers not written in the English language.
  - Surveys.

We applied these criteria in a step-by-step process using the available filters on the webpage of each scientific database considered. We decided to include only papers from journals, magazines, and conference proceedings because these are usually peer-reviewed. However, we highlight that some considered papers from the Springer database are also published as book chapters.

4.4. Classification

Following the classification presented in Cai et al. (2016a) and Okhravi et al. (2013), we propose a classification with four categories: (i) MTD research area; (ii) MTD strategy; (iii) evaluation metrics; and (iv) platform considered. These categories aim to cover the meaningful aspects of each paper, considering our research questions. Our classification approach is transverse in all the proposed categories, meaning that a paper classification can comprise more than one group of each proposed category. The details of each category are discussed in the following paragraphs.

4.4.1. MTD research area

In this category, we aim to classify the type of research published observing the classification proposed in Cai et al. (2016a). The category has three groups, Theory, Strategy, and Evaluation, as follows:

- **Theory** - Find answers to fundamental questions regarding MTD techniques.
  - How to create effective MTD system?
  - What capabilities and features are essential to MTD systems?
- **Strategy** - Propose a technique for MTD.
- **Evaluation** - Measure the effectiveness of existing (or proposed) strategies.

As mentioned before, we consider that a paper can be classified in more than one category. For example, some papers propose and evaluate a mechanism for MTD on the cloud. Therefore, these papers are classified as *Strategy + Evaluation (S+E)*.

4.4.2. MTD strategies

While the research area category is quite generic, we assume that MTD strategies are more focused on cloud computing environments. We organize the papers using the groups proposed by Okhravi et al. (2013). We have thus three groups:

- ***Dynamic application*** - which comprises dynamic *data* (change data format, syntax or encoding), dynamic *software* (dynamic changes on the application code), and dynamic *runtime environment* (address space randomization or instruction set randomization).
- ***Dynamic platform*** - dynamic changes on the platform configurations, including Operating System (OS) version, CPU architecture or OS instance. We can deploy a *Dynamic Platform* MTD in the cloud using VM migration or VM placement techniques.
- ***Dynamic Network*** - change network properties (e.g. IP address or network protocols) dynamically.

4.4.3. Evaluation metrics

The deployment of an MTD mechanism implies costs for system performance while improving the system security. Therefore, we intend to understand which evaluation metrics are currently used in MTD in cloud research. We propose only two groups for this category:

- ***Performance***, evaluation comprises performance metrics, such as response time, system overhead, *etc*;
- ***Security***, evaluation covers security aspects, such as attack success rate, survivability, *etc*.

4.4.4. Platform considered

This category has no predefined groups. We aim to fill the category using a bottom-up approach. Thus, we propose the category groups after the analysis of the papers. This category can reveal the correlation between cloud computing and other platforms (e.g., Software Defined Networks or Virtualized Containers).

4.5. Threats to validity and limitations

There are threats to validity in systematic mapping research. It is important to highlight the adopted approaches to avoid or minimize them. In the scope of our work, we can highlight the following threats to validity: (i) *weak search string*; (ii) *scientific database limitation*; and (iii) *classification bias*.

About threat (i), weak search strings may result in a reduced search result lacking relevant papers of the considered area. We decided to use a generic search string instead of the composition of specific search strings (as presented in Section 4.2). Therefore, we collected a reasonable amount of papers from scientific databases. The drawback of applying generic search strings is the increased subjective classification effort. When using the generic search string, we ended up finding many papers about MTD that also mention the word “cloud” as a concept or related work. We carefully classified these papers to avoid mistakes in the paper selection process.

About threat (ii), we decide to include five relevant scientific databases in the computer science area, as presented in previous systematic mapping studies (Fernandez et al., 2011; Roberto et al., 2016).

Finally, about threat (iii), as the analysis of paper inclusion/exclusion was made by the authors, the classification may be biased. However, we did put a reasonable effort to reduce classification bias by analyzing more paper content than just the abstract. Besides that, to improve our classification process, we performed two separate rounds of the full systematic mapping process, one in November 2018 and a confirmation round in July 2019. In the confirmation round, we analyzed all the papers from the previous round (to confirm the classification). Besides that, we added the papers published between the rounds. The results presented here are from the confirmation round.

Besides the threats to validity, we emphasize the following limitations in our research approach. Firstly, our research focuses on the deployment of MTD techniques in the cloud computing environment. As we mentioned earlier, we used a generic search string to find the relevant papers in the context. However, there may be other papers which do not mention ”cloud” directly but apply to the virtualized environment. These papers will require a more in-depth analysis, which is out of the scope of this paper.

Moreover, considering the number of collected papers, we performed the second step of papers filtering manually (exclusion of surveys, editorials, keynote, and duplicate entries). Thus, our approach has some scalability issues. A procedure to overcome this limitation is the development of a software (script) capable of conducting automated paper collection and filtering.

5. Results

The search was made between 5 and 16 of July 2019 and resulted in 224 papers. The first step was to apply the selection criteria using the automated filters provided in each database webpage. Following the proposed criteria, the automated filters help to exclude papers published before 2009 and papers that were not published in journals, magazines, or conference proceedings. This step resulted in a set of 163 papers. The second step of the process was performed manually and consisted of the exclusion of editorial papers, keynote papers, surveys, or duplicated entries. The manual work is as follows. We downloaded all the papers and performed a quick analysis of them to notice whether they are editorial, keynote, or survey papers. We removed all the papers in these categories, along with the duplicated entries. Unfortunately, we have to conduct this work by hand because the automated search bases do not offer the filtering for these types of papers. This step reduced the to 95 the set of papers eligible for analysis. The step-by-step process for each paper is presented in Fig. 3.

Fig. 3 shows that the most relevant database is the IEEEXplore, which provided 40 papers for the analysis (42.1% of the total). The rest of the databases provided: SpringerLink - 29 papers (30.52%), ScienceDirect - 14 papers (14.73%), ACM Digital Library - 10 papers (10.52%), and Wiley Online Library - 2 papers (2.1%).

RQ1 is related to the frequency of publication of papers on MTD in Cloud computing. Fig. 4 shows the annual trend since 2011. We did not find any published in 2009 and 2010. It is possible to observe a growing research interest in MTD in cloud computing. Note that the data for 2019 is still incomplete (because of the date of the search), so the number of publications is expected to increase.

Table 1 presents the most relevant publication forums. It provides the answer to *RQ2*. *ACM Workshop on Moving Target Defense* is the flagship forum with five papers. We noticed that the publication forums are diverse, as more than 80% of the papers are from forums that provided less than three papers for our analysis. Some papers from the *Procedia Computer Science* journal are extended versions of previous conference papers.

We also analyzed the distribution of papers according to the type of publication forum. Fig. 5 presents the proportion of papers published in the three types of forums. As expected, papers in conferences and workshops represent the majority of the published papers. Journals and magazine papers usually pass through a rigorous review process, which can lead to fewer papers published in such forums.

Each paper was classified using the scheme presented in Section 4. The complete list of the selected works is available on-

Table 1
List of the most relevant publication forums.

Forum	Type of forum	Number of papers	Percentage of the total
ACM Workshop on Moving Target Defense	Workshop	5	5.26%
Procedia Computer Science	Journal	4	4.21%
IEEE International Conference on Cloud Computing	Conference	3	3.16%
Frontiers of Information Technology & Electronic Engineering	Journal	3	3.16%
Future Generation Computer Systems	Journal	3	3.16%
Other 64 forums	-	77	81.05%

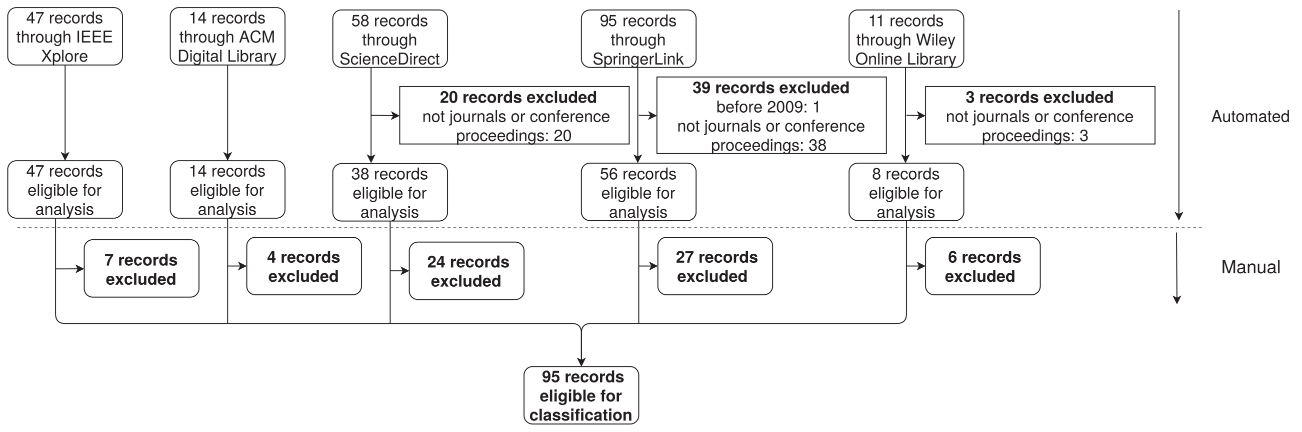

47 records through IEEE Xplore
14 records through ACM Digital Library
58 records through ScienceDirect
95 records through SpringerLink
11 records through Wiley Online Library
20 records excluded not journals or conference proceedings: 20
39 records excluded before 2009: 1 not journals or conference proceedings: 38
3 records excluded not journals or conference proceedings: 3
Automated
47 records eligible for analysis
14 records eligible for analysis
38 records eligible for analysis
56 records eligible for analysis
8 records eligible for analysis
7 records excluded
4 records excluded
24 records excluded
27 records excluded
6 records excluded
Manual
95 records eligible for classification

Fig. 3. Paper selection process.

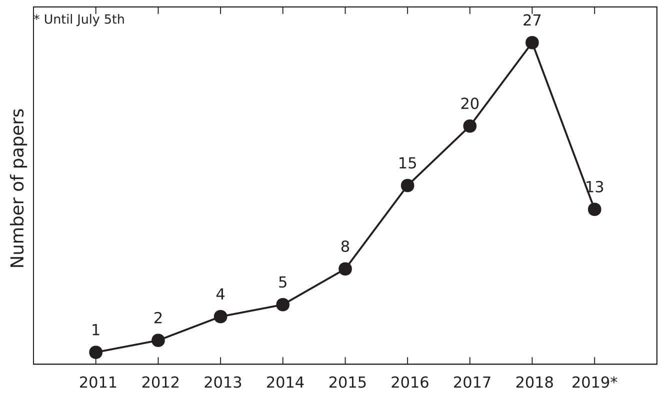


Fig. 4. Publications over time with annual trend.

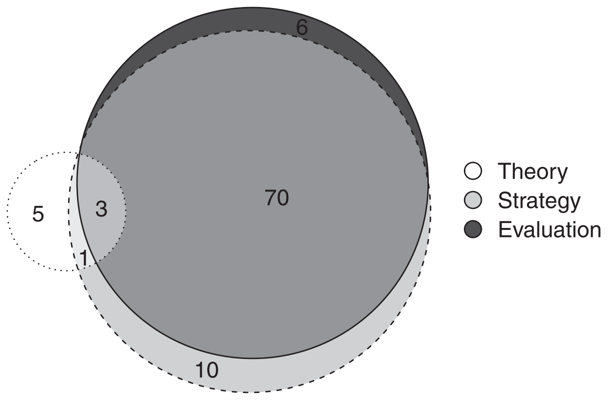


Fig. 6. Research area classification.

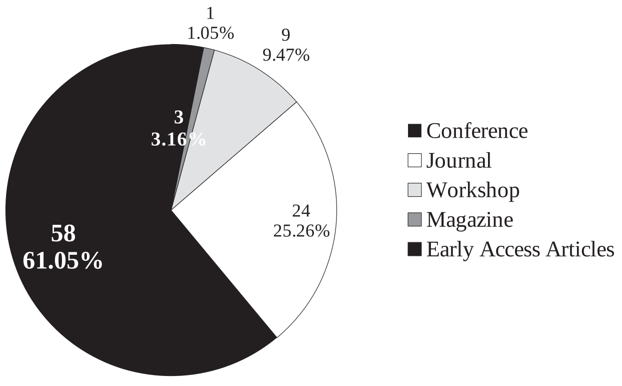


Fig. 5. Publications classification - type of forum.

line[1]. The research community can send new entries or suggestions for improving the classification using a link on the same web page.

[1] https://www.matheustorquato.com/publications/systematic-map-of-moving-target-defense-on-cloud-computing.

6. Mapping

This section provides an overview of Moving Target Defense in cloud computing research. We present charts and diagrams with the distribution of publications regarding the classification mentioned earlier.

6.1. Research area - papers distribution

The first map is a Venn Diagram of the distribution of papers in terms of theory, strategy, and evaluation (see Fig. 6). We noticed that most papers propose an MTD strategy and present its evaluation. As we have books and seminal papers to support MTD theory (Jajodia et al., 2012; 2011; Zhuang et al., 2014), the scientific community seeks to offer more approaches to enhance the available set of MTD mechanisms. However, theoretical papers usually provide more generic contributions on the use of MTD in other scenarios (not only cloud computing). For example, Leslie et al. (2015) propose a model based on game theory to support resources configuration to reduce the likelihood of a secu-

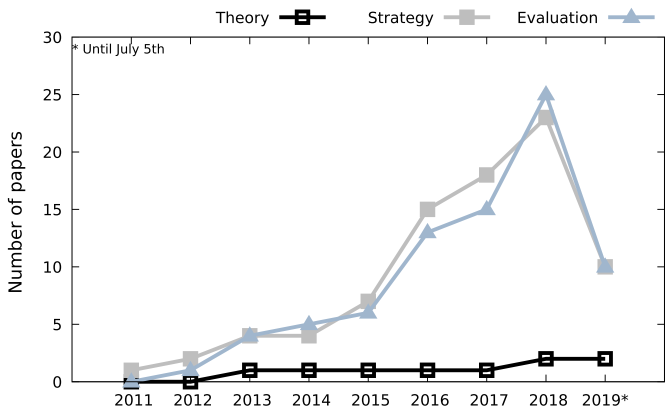


Fig. 7. Research area classification - publications over time.

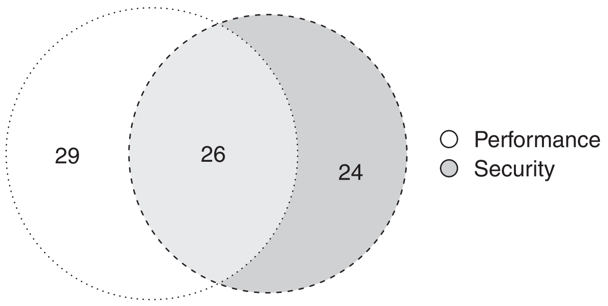


Fig. 8. Evaluation approaches - papers distribution.

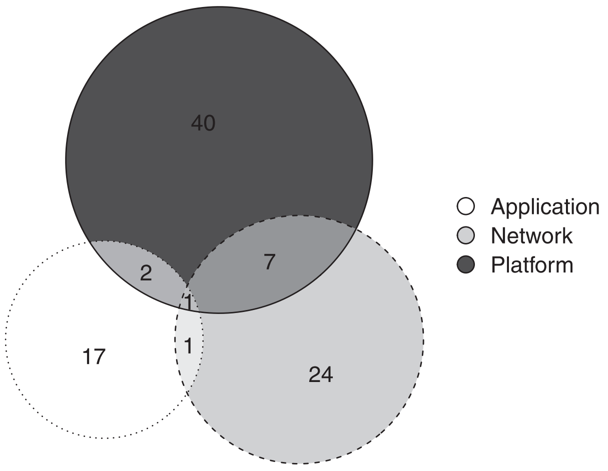


Fig. 9. Strategy classification.

rity attack. Lei et al. (2018b) also propose an approach based on game theory. Their proposal consists of an incomplete information Markov game theory comprising a moving attack surface and optimal strategy selection. The papers (Peng et al., 2014a; Song et al., 2019; Wang et al., 2016) are in the intersection between theory, strategy, and evaluation. Besides proposing an MTD strategy and its evaluation, they present a robust theoretical framework with models and algorithms.

Papers that propose generic evaluation methods are useful to support the comparison of MTD techniques. The papers from Alavizadeh et al. (2018a, 2018b, 2017) provide a modeling framework for the evaluation of MTD in cloud environments. The authors cover relevant security aspects as return on the attack, attack cost, and the probability of attack success. Their results also comprise a comparison between a system with and without MTD deployments.

We also studied the evolution in terms of the number of papers published over time. The plot in Fig. 7 is inclusive, meaning that the paper is counted in each research area that it resides. For example, if a paper is about an MTD strategy and its evaluation, we count this paper in the Strategy category and also in the Evaluation category. It is noticeable that papers on theory received less attention from the research community in the last years. Theoretical papers focused on the aspects of cloud computing and how to deploy effective MTD in the cloud are an exciting aspect for future research.

6.2. Evaluation metrics - papers distribution

Fig. 8 presents a Venn diagram with the distribution of the metrics found in the selected papers. We noticed a balanced distribution of papers in the proposed classification. Papers that include a performance evaluation usually focus on the overhead caused by the proposed (or evaluated) MTD strategy. For example, Yang and Cheng (2018) present an MTD based on Software Defined Network (SDN). Among their results, the authors compared the response time of the application when using their proposal with the traditional MTD strategies. Wang et al. (2016) propose a cost-effective MTD against Distributed Denial of Service (DDoS) and Covert Channel Attacks. Their performance evaluation covers the cost per minute of using different variations of MTD algorithms.

The specific assumptions of each research impose barriers for the comparison of different MTD strategies. The security evaluation metrics considered in the works analyzed tend to be related to specific aspects to characterize the proposed MTD effectiveness. The metrics are usually defined by the authors and applied in their specific context. For example, the study from Sianipar et al. (2018) is focused on the Meltdown and Spectre vulnerabilities. Therefore, their results are based on Spectre and Meltdown effectiveness while applying their approach. Wang et al. (2014) propose an MTD solution as a DDoS defense. The evaluation comprises the percentage of clients saved based on the number of shuffles. Wahab et al. (2019) propose a comprehensive framework for MTD deployment in the cloud. Their framework comprises several techniques and methods, including a risk assessment methodology and a machine learning approach to collect information from malicious activities. In the evaluation, the authors use two primary metrics: percentage of attack detection and survived services.

The most recurrent security metric is related to the MTD impact in the attack success rate (Debroy et al., 2016; Ma et al., 2016; Nguyen et al., 2018; Zhang et al., 2016b). However, it is still challenging to set up a direct comparison between the papers due to their particular assumptions. The development of a unified approach for MTD evaluation is an open problem.

6.3. Strategy - papers distribution

This section presents an overview of the type of MTD strategies applied in cloud computing. The results presented here provide answers to *RQ3*. Fig. 9 presents a Venn diagram with the distribution of the proposed MTD strategies. As mentioned in Section 4, each set in the Venn diagram corresponds to MTD strategies related to the dynamic application, dynamic network, and dynamic platform.

We noticed that most of the MTD techniques leverage cloud computing inherent features. For example, MTD based on the dynamic platform usually relies on Virtual Machine (VM) migration for the environment reconfiguration. In this context, VM migration is usually used to defend against side-channel attacks (Adili et al., 2017; Azab et al., 2017; Kashkoush et al., 2018; Moon et al., 2015; Yang et al., 2019; Zhang et al., 2012).

Liu et al. (2018) present an MTD approach against side-channel attacks based on dynamically scheduling VM computing resources. Agarwal and Duong (2019) propose a different MTD solution to defend against side-channel attacks using a VM placement technique. They propose an algorithm to reduce the probability of malicious VM co-location. Also using VM placement techniques, Ahmed and Bhargava (2016) propose a MTD framework based on the creation and deletion (*reincarnation*) of VMs. To improve security, the authors dynamically change the OS instance on the VM in each *reincarnation* round. Jia et al. (2014) present an MTD mechanism to isolate attacked servers from benign clients during DDoS attacks. Their approach consists of turning victim servers into moving targets. Penner et al. work (Penner and Guirguis, 2017) leverage on both VM migration and VM placement techniques to provide a comprehensive MTD mechanism for cloud computing.

Dynamic network approaches are usually based on network address hopping techniques (El Mir et al., 2017; Groat et al., 2013; Luo et al., 2016). Kurra et al. (2013) present an MTD mechanism based on data partitioning and key hopping. Using key hopping mechanisms, the authors can reduce the length of keys to improve system performance while maintaining system security levels. Fleck et al. (2018) propose dynamic changes on the IP addresses of proxies to thwart the reconnaissance phase of attacks. Lysenko et al. (2018) also propose dynamic network configurations to protect a Corporate Area Network.

Regarding MTD techniques related to the dynamic application approach, we highlight that the most used technique is *Software Behavior Encryption* (SBE). SBE is usually applied using a dynamic selection of functionally-equivalent software variants at runtime (Dsouza et al., 2013; Hosseinzadeh et al., 2015; Le Goues et al., 2013). The oldest paper in our classification (Azab and Eltoweissy, 2011) also applies SBE in the context of Cyber-Physical Systems (CPS). The authors used the *ChameleonSoft*, a biological-inspired MTD framework that provides software diversity at runtime.

Finally, we highlight that just one paper (Chung et al., 2015) propose a framework (SeReNe) comprising all the three layers (application, network, and platform). However, SeReNe is still in a conceptual phase and lacks practical implementation and evaluation.

6.4. Platforms - papers distribution

In this category, we aim to understand whether the papers are only focused on cloud computing or are also considering other platforms. We find this category useful to identify cross-platform MTD strategies or to perceive how cloud may support the application MTD in other scenarios. We found out that the majority of papers are focused only on cloud computing instead of considering the use of cloud computing in conjunction with other platforms (e.g., Software Defined Networking or Virtualized Containers). However, there is a non-negligible set of papers considering these other platforms. An interesting example is a paper from Kahla et al. (2018), which proposes a technique for Fog computing and the Internet of Things (IoT). Some papers leverage the flexibility of Software Defined Networks (SDN) to propose more robust MTD solutions (Chowdhary et al., 2016; Urias et al., 2015; Villarreal-Vasquez et al., 2017). Due to lightweight virtualization overhead, some research works aim to deploy MTD using virtualized containers (Jin et al., 2019; Torkura et al., 2018). The overall results are presented in Fig. 10.

We highlight the paper from Pacheco et al. (2016), which applies MTD on top of the cloud but focusing on the smart city context. The authors compared the performance levels with and without using MTD. Lei et al. paper (Lei et al., 2018b) has a generic context. Thus, it is hard to classify it in a single category of this classification. Therefore, we decided to exclude this paper from the classification of this specific subsection of our systematic mapping.

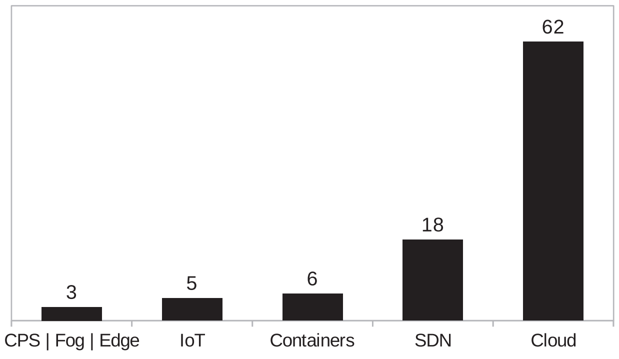


Fig. 10. Considered platform category.

6.5. Research area and strategy - classification relationship

The bubble chart in Fig. 11 presents the relationship between the research area and strategy categories of the selected papers. Bubble charts simplify the identification of research gaps and the areas that received the most attention from the research community.

As mentioned earlier, most papers propose an MTD strategy and its evaluation (*S+E* category). Moreover, among these papers, the majority use dynamic platform strategies. There are three theoretical papers (Bazm et al., 2017; Lei et al., 2018b; Leslie et al., 2015) that study generic MTD theory without focusing on specific strategies. Some papers propose MTD strategies but lack the evaluation of their effectiveness. MTD Theory receives less attention than the other areas from the research community. The paper from Casola et al. (2018) was classified as theory and strategy because, besides presenting a security SLA-driven MTD framework, it presents a strong theory about cloud applications and security SLAs.

6.6. Platform and strategy - classification relationship

Fig. 12 presents the relationship between platforms and strategies. We noticed that the papers from the cloud category usually apply dynamic platform techniques, as presented in Section 6.3. Besides that, we noticed that MTD approaches for virtualized containers also tend to leverage from dynamic platform techniques. For example, Azab et al. (2016a,b) present an MTD based on the live migration of virtualized containers to avoid security attacks.

The majority of the papers that use dynamic network MTD approaches leverages the flexibility of the SDN paradigm. The strategies vary from usual IP mutation (Chang et al., 2018; Zhang et al., 2016a) and port hopping (Chowdhary et al., 2018) to route randomization (Aydeger et al., 2019; Karim et al., 2019).

6.7. Authors analysis

Some of the considered papers in our systematic mapping are closely related. For example, there are papers from the same authors published in conferences and journals, which have similar goals (e.g., Kashkoush et al., 2017; Kashkoush et al., 2018). In such cases, we consider both papers in our analysis (i.e., conference and journal versions). As our *RQ1* is related to the frequency of paper publication, we decided to maintain both versions in our analysis to provide a more precise overview of the literature in the last ten years.

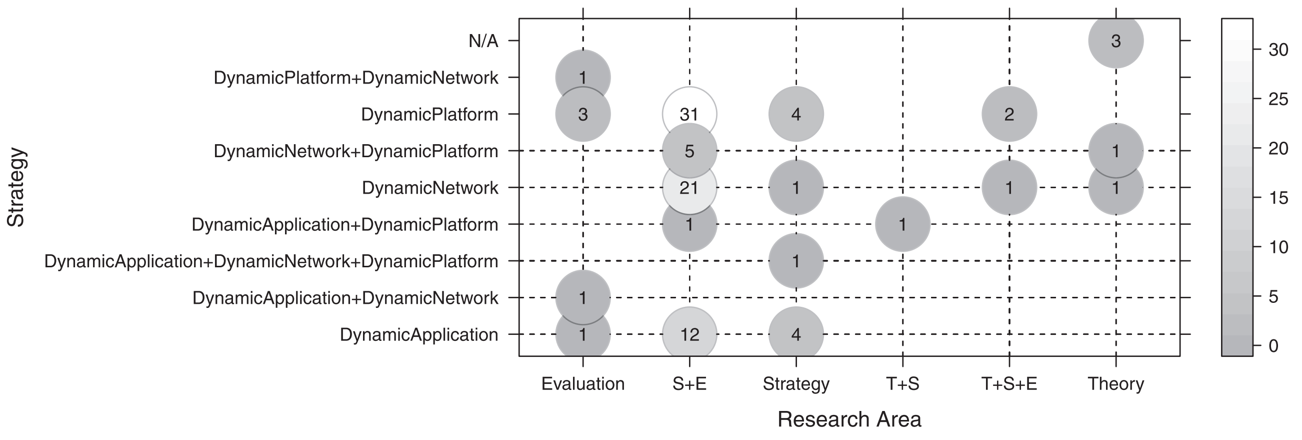


Fig. 11. Relationship between Research Area and Strategy categories.

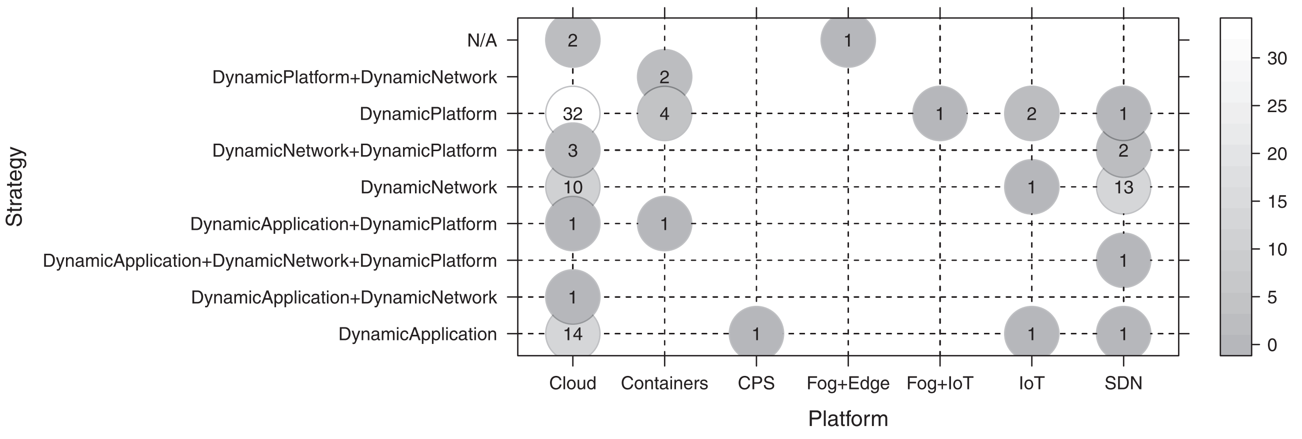


Fig. 12. Relationship between considered platforms and strategy categories.

However, to highlight the possible relationship between the publications from the same authors, we conducted an author analysis on the considered papers. This analysis aims to find the papers from the same set of authors and to verify possible relationships between them. Besides that, the analysis output also provides the most prominent authors on the field (i.e., authors with most publications).

Mir et al. published two related papers with evaluation approaches for MTD deployments in the cloud (El Mir et al., 2016; 2017). Wang et al. published papers about MTD deployments for defending cloud computing from Denial of Service attacks (Jia et al., 2014; Wang et al., 2014; 2016). As mentioned earlier, Kashkoush et al. published the related papers (Kashkoush et al., 2017; 2018) about Moving Target Defense to avoid co-residency attacks. Hooman Alavizadeh is one of the most active authors in the MTD evaluation research area. He and his co-authors published papers applying modeling in the MTD deployments evaluation (Alavizadeh et al., 2018a; 2018b; 2017). Jajodia et al. provide interesting insights into the deployment of MTD on cloud databases (Jajodia et al., 2015; 2016).

Fig. 13 presents a *wordcloud* with the names of all the authors of the selected papers. In this *wordcloud*, the font size of the author's name varies according to the number of publications from the author (i.e., big-sized font names represent that the author has more publications than the authors with small-sized font names). Thus, this *wordcloud* is useful to highlight the most prominent authors on the scope of our systematic mapping.

Finally, we highlight the following authors as the most prominent authors on MTD in the cloud in from 2009 to July 2019: **Mohamed Azab** (Virginia Military Institution, USA and Informatics Research Institute of Alexandria, Egypt.) published eight of the considered papers; **Dijiang Huang** (Arizona State University, USA) published seven of the considered papers; and, **Ankur Chowdhary** (Arizona State University, USA), **Dong Seong Kim** (The University of Queensland, Australia), and **Salim Hariri** (The University of Arizona, USA) with six published papers.

7. Discussion

Moving target defense attracted the attention of the cloud security research community in the last years, e.g., the number of publications per year increased 35% from 2017 to 2018. The number of publications on July 5th, 2019, is already about 50% of the publications in 2018. Considering that relevant venues, such as the *2019 ACM Workshop on Moving Target Defense* and the *2019 IEEE International Conference on Cloud Computing*, are still not indexed till the date of this paper writing, the number of papers published in 2019 may surpass the number of publications in 2018. Actually, in the same period of 2018 (January 1st to July 5th), only seven papers were published, while in 2019, 13 papers were published. The publication forums in the last ten years are diverse, meaning that the research community is still consolidating the primary forums of interest.

While applying or proposing MTD mechanisms for cloud computing, the researchers leverage on cloud and virtualization features, including VM placement and migration techniques. The problem with this approach is that these MTD techniques rely only on platform modification. Therefore, some more well-prepared at-

Fig. 13. Word cloud with the selected papers' authors.

tackers may develop security attacks targeting higher layers, such as application confidentiality or user privacy. However, the use of cloud and virtualization capabilities reduces the cost of MTD implementation due to the use of cloud embedded features. Besides that, the mappings show that authors have a strong interest in proposing and evaluating MTD mechanisms.

There is a significant research effort in expanding MTD from cloud computing to other platforms. The relationship between cloud and those platforms is mutual. Some works use the cloud to enable MTD in another specific platform (like Cyber-Physical Systems and IoT). Some other works use other platforms to improve the security levels of the cloud (e.g., using SDN).

In the following paragraphs, we highlight four relevant research opportunities on MTD in cloud computing. This is a non-exhaustive list, but it provides the significant research gaps found in our systematic mapping study.

Research opportunity 1 - ***Theoretical research about MTD in Cloud computing***. The majority of the theoretical papers found are generic. MTD theories that consider the characteristics of clouds and their virtualized environments represent a research opportunity. For example, a relevant MTD problem is the *MTD timing problem*, where one tries to define optimal schedules to perform MTD actions taking into account the desired system attributes (e.g., security, performance or sustainability).

Research opportunity 2 - ***A unified framework for MTD evaluation***. The development of security benchmarks is a complex problem due to the inherent unpredictability of the attackers. However, previous research (Dumitraş and Shou, 2011; Vieira and Madeira, 2005) provides directions for the design of such benchmarks. The current research on MTD in the cloud focuses on proposing new techniques to avoid (or reduce the likelihood) of specific security threats. The problem is that, without unified evaluation metrics, it is hard to compare and decide among the available MTD methods. Proposing a unified MTD evaluation approach may be an insurmountable challenge. However, starting proving evaluation approaches for specific scenarios (e.g., MTD in the cloud that applies VM migration to avoid side-channel attacks) seems to be an interesting research opportunity.

Research opportunity 3 - ***Multi-layer MTD***. As mentioned earlier, researchers mainly explored cloud features as enabling mechanisms for MTD deployments. There is still a gap in the development of multi-layer MTD frameworks for cloud computing. Just applying dynamic platform and network techniques are not enough to mitigate sophisticated attacks that aim at the system confidentiality or users' privacy. The development of a self-adaptive framework capable of dynamic multi-layer MTD selection is an interesting research challenge.

Research opportunity 4 - ***Impact of MTD in context-oriented clouds***. As presented in Buyya et al. (2018), there is a need for holistic evaluations in cloud computing environments. Applying MTD in context-oriented clouds may impose severe system overhead. Although the current research in MTD in the cloud is covering diverse platforms, we noticed a research gap in the evaluation of the MTD impact in context-oriented clouds. For example, some Infrastructure-as-a-Service (IaaS) clouds are devoted to offering *high-availability* to its clients. Let us suppose that, besides high availability, the system also needs to improve security levels. The evaluation of the possible impacts of applying MTD in such scenarios seems to be an interesting research problem.

8. Conclusions

This work presented a systematic mapping of Moving Target Defense in cloud research. To achieve this goal, we collected 224 papers from five computer science scientific databases. The selection process resulted in 95 papers for analysis. The papers were classified according to four main properties: research area, strategy, evaluation metrics, and platforms.

We present here simplified answers to our research questions. *RQ1: How has the frequency on moving target defense on cloud computing changed in the last ten years?* We noticed a growing interest in the MTD in cloud research in the last ten years. *RQ2: In which forums have research on moving target defense on cloud computing been published?* The most relevant conference in the area is the *ACM Workshop on Moving Target Defense*, and the most relevant journal is *Procedia Computer Science. RQ3: What are the most investigated techniques in moving target defense on cloud computing research?* In the dynamic platform papers, the most used technique is VM migration. In dynamic network papers, the most used technique is the network address randomization. A significant number of dynamic network papers rely on SDN flexibility to perform dynamic network changes. Finally, the most used technique in dynamic application papers is Software Behavior Encryption.

The complete list of selected papers is available online.[2] The research community can send new suggestions for paper inclusion or classification corrections.

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.

Acknowledgments

This work has been partially supported by Portuguese funding institution FCT - Foundation for Science and Technology, Ph.D. grant SFRH/BD/146181/2019 and project ATMOSPHERE, funded by the European Commission under the Cooperation Programme, Horizon 2020 grant agreement no 777154.

[2] https://www.matheustorquato.com/publications/systematic-map-of-moving-target-defense-on-cloud-computing .

References

Adili, M.T., Mohammadi, A., Manshaei, M.H., Rahman, M.A., 2017. A cost-effective security management for clouds: a game-theoretic deception mechanism. In: Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on. IEEE, pp. 98–106.

Agarwal, A., Duong, T.N.B., 2019. Secure virtual machine placement in cloud data centers. Fut. Gener. Comput. Syst. 100, 210–222.

Ahmed, N.O., Bhargava, B., 2016. Mayflies: a moving target defense framework for distributed systems. In: Proceedings of the 2016 ACM Workshop on Moving Target Defense. ACM, pp. 59–64.

Alavizadeh, H., Hong, J.B., Jang-Jaccard, J., Kim, D.S., 2018. Comprehensive security assessment of combined MTD techniques for the cloud. In: Proceedings of the 5th ACM Workshop on Moving Target Defense. ACM, pp. 11–20.

Alavizadeh, H., Jang-Jaccard, J., Kim, D.S., 2018. Evaluation for combination of shuffle and diversity on moving target defense strategy for cloud computing. In: 2018 17th IEEE International Conference On Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE). IEEE, pp. 573–578.

Alavizadeh, H., Kim, D.S., Hong, J.B., Jang-Jaccard, J., 2017. Effective security analysis for combinations of MTD techniques on cloud computing (short paper). In: International Conference on Information Security Practice and Experience. Springer, pp. 539–548.

Alavizadeh, H., Kim, D.S., Jang-Jaccard, J., 2019. Model-based evaluation of combinations of shuffle and diversity MTD techniques on the cloud. Fut. Gener. Comput. Syst..

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al., 2010. A view of cloud computing. Commun. ACM 53 (4), 50–58.

Aydeger, A., Saputro, N., Akkaya, K., 2019. A moving target defense and network forensics framework for ISP networks using SDN and NFV. Fut. Gener. Comput. Syst. 94, 496–509.

Azab, M., Eltoweissy, M., 2011. Defense as a service cloud for cyber-physical systems. In: Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on. IEEE, pp. 392–401.

Azab, M., Eltoweissy, M., Attiya, G., et al., 2017. Towards online smart disguise: real-time diversification evading co-residency based cloud attacks. In: 2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC). IEEE, pp. 235–242.

Azab, M., Mokhtar, B., Abed, A.S., Eltoweissy, M., 2016. Toward smart moving target defense for linux container resiliency. In: Local Computer Networks (LCN), 2016 IEEE 41st Conference on. IEEE, pp. 619–622.

Azab, M., Mokhtar, B.M., Abed, A.S., Eltoweissy, M., 2016. Smart moving target defense for linux container resiliency. In: Collaboration and Internet Computing (CIC), 2016 IEEE 2nd International Conference on. IEEE, pp. 122–130.

Bazm, M.-M., Lacoste, M., Südholt, M., Menaud, J.-M., 2017. Side-channels beyond the cloud edge: new isolation threats and solutions. In: Cyber Security in Networking Conference (CSNet), 2017 1st. IEEE, pp. 1–8.

Bonguet, A., Bellaiche, M., 2017. A survey of denial-of-service and distributed denial of service attacks and defenses in cloud computing. Future Internet 9 (3), 43.

Buyya, R., Srirama, S.N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., Gelenbe, E., Javadi, B., Vaquero, L.M., Netto, M.A., et al., 2018. A manifesto for future generation cloud computing: research directions for the next decade. ACM Comput. Surv. 51 (5), 105.

Cai, G., Wang, B., Luo, Y., Li, S., Wang, X., 2016. Characterizing the running patterns of moving target defense mechanisms. In: Advanced Communication Technology (ICACT), 2016 18th International Conference on. IEEE, pp. 191–196.

Cai, G., Wang, B., Hu, W., Wang, T., 2016. Moving target defense: state of the art and characteristics. Front. Inf. Technol. Electron.Eng. 17 (11), 1122–1153.

Casola, V., De Benedictis, A., Rak, M., Villano, U., 2018. A security SLA-driven moving target defense framework to secure cloud applications. In: Proceedings of the 5th ACM Workshop on Moving Target Defense. ACM, pp. 48–56.

Chang, S.-Y., Park, Y., Babu, B.B.A., 2018. Fast ip hopping randomization to secure hop-by-hop access in sdn. IEEE Trans. Netw. Serv. Manage. 16 (1), 308–320.

Chowdhary, A., Alshamrani, A., Huang, D., Liang, H., 2018. MTD analysis and evaluation framework in software defined network (mason). In: Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization. ACM, pp. 43–48.

Chowdhary, A., Pisharody, S., Huang, D., 2016. SDN based scalable MTD solution in cloud network. In: Proceedings of the 2016 ACM Workshop on Moving Target Defense. ACM, pp. 27–36.

Chung, C.-J., Xing, T., Huang, D., Medhi, D., Trivedi, K., 2015. Serene: on establishing secure and resilient networking services for an SDN-based multi-tenant datacenter environment. In: Dependable Systems and Networks Workshops (DSN-W), 2015 IEEE International Conference on. IEEE, pp. 4–11.

Das, S., Mahfouz, A.M., Shiva, S., 2019. A stealth migration approach to moving target defense in cloud computing. In: Proceedings of the Future Technologies Conference. Springer, pp. 394–410.

Debroy, S., Calyam, P., Nguyen, M., Stage, A., Georgiev, V., 2016. Frequency-minimal moving target defense using software-defined networking. In: Computing, Networking and Communications (ICNC), 2016 International Conference on. IEEE, pp. 1–6.

Dsouza, G., Hariri, S., Al-Nashif, Y., Rodriguez, G., 2013. Resilient dynamic data driven application systems (RDDDAS). Procedia Comput. Sci. 18, 1929–1938.

Dumitraş, T., Shou, D., 2011. Toward a standard benchmark for computer security research: the worldwide intelligence network environment (wine). In: Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security. ACM, pp. 89–96.

El Mir, I., Chowdhary, A., Huang, D., Pisharody, S., Kim, D.S., Haqiq, A., 2016. Software defined stochastic model for moving target defense. In: International Afro-European Conference for Industrial Advancement. Springer, pp. 188–197.

El Mir, I., Haqiq, A., Kim, D.S., 2017. A game theoretic approach for cloud computing security assessment using moving target defense mechanisms. In: Proceedings of the Mediterranean Symposium on Smart City Applications. Springer, pp. 242–254.

Fernandez, A., Insfran, E., Abrahão, S., 2011. Usability evaluation methods for the web: asystematic mapping study. Inf. Softw. Technol. 53 (8), 789–817.

Fleck, D., Stavrou, A., Kesidis, G., Nasiriani, N., Shan, Y., Konstantopoulos, T., 2018. Moving-target defense against botnet reconnaissance and an adversarial coupon-collection model. In: 2018 IEEE Conference on Dependable and Secure Computing (DSC). IEEE, pp. 1–8.

Groat, S., Moore, R., Marchany, R., Tront, J., 2013. Securing static nodes in mobile-enabled systems using a network-layer moving target defense. In: 2013 1st International Workshop on the Engineering of Mobile-Enabled Systems (MOBS). IEEE, pp. 42–47.

Hong, J.B., Enoch, S.Y., Kim, D.S., Nhlabatsi, A., Fetais, N., Khan, K.M., 2018. Dynamic security metrics for measuring the effectiveness of moving target defense techniques. Comput. Secur. 79, 33–52.

Hosseinzadeh, S., Laurén, S., Rauti, S., Hyrynsalmi, S., Conti, M., Leppänen, V., 2015. Obfuscation and diversification for securing cloud computing. In: International Workshop on Enterprise Security. Springer, pp. 179–202.

Jajodia, S., Ghosh, A.K., Subrahmanian, V., Swarup, V., Wang, C., Wang, X.S., 2012. Moving Target Defense II: Application of Game Theory and Adversarial Modeling, 100. Springer.

Jajodia, S., Ghosh, A.K., Swarup, V., Wang, C., Wang, X.S., 2011. Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats, 54. Springer Science & Business Media.

Jajodia, S., Litwin, W., Schwarz, T., 2015. Numerical SQL value expressions over encrypted cloud databases. In: Database and Expert Systems Applications. Springer, pp. 455–478.

Jajodia, S., Litwin, W., Schwarz, T., 2016. On-the-fly AES256 decryption/encryption for trusted cloud SQL DBS: position statement. In: 2016 27th International Workshop on Database and Expert Systems Applications (DEXA). IEEE, pp. 19–23.

Jia, Q., Wang, H., Fleck, D., Li, F., Stavrou, A., Powell, W., 2014. Catch me if you can: a cloud-enabled DDoS defense. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, pp. 264–275.

Jin, H., Li, Z., Zou, D., Yuan, B., 2019. Dseom: a framework for dynamic security evaluation and optimization of MTD in container-based cloud. IEEE Trans. Depend. Secure Comput..

Kahla, M., Azab, M., Mansour, A., 2018. Secure, resilient, and self-configuring fog architecture for untrustworthy IoT environments. In: 2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE). IEEE, pp. 49–54.

Karim, Z., Sebbara, A., Baddic, Y., Boulmalfa, M., 2019. Secure multipath mutation SMPM in moving target defense based on SDN. Procedia Comput. Sci..

Kashkoush, M., Azab, M., Eltoweissy, M., Attiya, G., 2017. Towards online smart disguise: Real-time diversification evading co-residency based cloud attacks. In: 2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC), pp. 235–242. doi:10.1109/CIC.2017.00039.

Kashkoush, M.S., Azab, M., Attiya, G., Abed, A.S., 2018. Online smart disguise: real-time diversification evading coresidency-based cloud attacks. Cluster Comput. 1–16.

Kitchenham, B., Brereton, P., Budgen, D., 2010. The educational value of mapping studies of software engineering literature. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1. ACM, pp. 589–598.

Krutz, R.L., Vines, R.D., 2010. Cloud Security: A Comprehensive Guide to Secure Cloud Computing. Wiley Publishing.

Kurra, H., Al-Nashif, Y., Hariri, S., 2013. Resilient cloud data storage services. In: Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference. ACM, p. 17.

Le Goues, C., Nguyen-Tuong, A., Chen, H., Davidson, J.W., Forrest, S., Hiser, J.D., Knight, J.C., Van Gundy, M., 2013. Moving Target Defenses in the Helix Self-regenerative Architecture. In: Moving Target Defense II. Springer, pp. 117–149.

Lei, C., Zhang, H.-Q., Tan, J.-L., Zhang, Y.-C., Liu, X.-H., 2018. Moving target defense techniques: a survey. Secur. Commun. Netw. 2018.

Lei, C., Zhang, H.-Q., Wan, L.-M., Liu, L., Ma, D., 2018. Incomplete information Markov game theoretic approach to strategy generation for moving target defense. Comput. Commun. 116, 184–199.

Leslie, D., Sherfield, C., Smart, N.P., 2015. Threshold flipthem: when the winner does not need to take all. In: International Conference on Decision and Game Theory for Security. Springer, pp. 74–92.

Liu, L., Wang, A., Zang, W., Yu, M., Xiao, M., Chen, S., 2018. Shuffler: Mitigate cross-VM side-channel attacks via hypervisor scheduling. In: International Conference on Security and Privacy in Communication Systems. Springer, pp. 491–511.

Luo, Y.-B., Wang, B.-S., Cai, G.-L., Wang, X.-F., Zhang, B.-F., 2016. High performance low latency network address and port hopping mechanism based on netfilter. In: International Conference on Intelligent and Interactive Systems and Applications. Springer, pp. 239–244.

Lysenko, S., Savenko, O., Bobrovnikova, K., Kryshchuk, A., 2018. Self-adaptive system for the corporate area network resilience in the presence of botnet cyberattacks. In: International Conference on Computer Networks. Springer, pp. 385–401.

Ma, D., Lei, C., Wang, L., Zhang, H., Xu, Z., Li, M., 2016. A self-adaptive hopping approach of moving target defense to thwart scanning attacks. In: International Conference on Information and Communications Security. Springer, pp. 39–53.

Mell, P., Grance, T., et al., 2011. The NIST definition of cloud computing.

Moon, S.-J., Sekar, V., Reiter, M.K., 2015. Nomad: mitigating arbitrary cloud side channels via provider-assisted migration. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, pp. 1595–1606.

Moving, 2018. Target Defense. https://www.dhs.gov/science-and-technology/csd-mtdAccessed: 2018-12-09

Nguyen, M., Pal, A., Debroy, S., 2018. Whack-a-mole: Software-defined networking driven multi-level DDoS defense for cloud environments. In: 2018 IEEE 43rd Conference on Local Computer Networks (LCN). IEEE, pp. 493–501.

Okhravi, H., Rabe, M., Mayberry, T., Leonard, W., Hobson, T., Bigelow, D., Streilein, W., 2013. Survey of Cyber Moving Target Techniques. Technical Report. Massachusetts Inst of Tech Lexington Lincoln Lab.

Pacheco, J., Tunc, C., Hariri, S., 2016. Design and evaluation of resilient infrastructures systems for smart cities. In: 2016 IEEE International Smart Cities Conference (ISC2). IEEE, pp. 1–6.

Peng, W., Li, F., Huang, C.-T., Zou, X., 2014. A moving-target defense strategy for cloud-based services with heterogeneous and dynamic attack surfaces. In: Communications (ICC), 2014 IEEE International Conference on. IEEE, pp. 804–809.

Peng, W., Li, F., Zou, X., 2014. Moving target defense for cloud infrastructures: lessons from botnets. In: High Performance Cloud Auditing and Applications. Springer, pp. 35–64.

Penner, T., Guirguis, M., 2017. Combating the bandits in the cloud: a moving target defense approach. In: Cluster, Cloud and Grid Computing (CCGRID), 2017 17th IEEE/ACM International Symposium on. IEEE, pp. 411–420.

Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008. Systematic mapping studies in software engineering.. In: EASE, 8, pp. 68–77.

Petersen, K., Vakkalanka, S., Kuzniarz, L., 2015. Guidelines for conducting systematic mapping studies in software engineering: an update. Inf. Softw. Technol. 64, 1–18.

Popović, K., Hocenski, Ž., 2010. Cloud computing security issues and challenges. In: The 33rd International Convention MIPRO. IEEE, pp. 344–349.

Ren, K., Wang, C., Wang, Q., 2012. Security challenges for the public cloud. IEEE Internet Comput. 16 (1), 69–73.

RightScale, 2018. Rightscale 2018 State of the Cloud Report.

Roberto, R., Lima, J.P., Teichrieb, V., 2016. Tracking for mobile devices: a systematic mapping study. Comput. Graph. 56, 20–30.

Sengupta, S., Chowdhary, A., Huang, D., Kambhampati, S., 2019. General sum Markov games for strategic detection of advanced persistent threats using moving target defense in cloud networks. In: International Conference on Decision and Game Theory for Security. Springer, pp. 492–512.

Sengupta, S., Chowdhary, A., Sabur, A., Huang, D., Alshamrani, A., Kambhampati, S., 2019b. A survey of moving target defenses for network security. arXiv:1905.00964.

Sianipar, J., Sukmana, M., Meinel, C., 2018. Moving sensitive data against live memory dumping, spectre and meltdown attacks. In: 2018 26th International Conference on Systems Engineering (ICSEng). IEEE, pp. 1–8.

Song, F., Zhou, Y.-T., Wang, Y., Zhao, T.-M., You, I., Zhang, H.-K., 2019. Smart collaborative distribution for privacy enhancement in moving target defense. Inf. Sci. 479, 593–606.

Torkura, K.A., Sukmana, M.I., Kayem, A.V., 2018. A cyber risk based moving target defense mechanism for microservice architectures. In: 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). IEEE, pp. 932–939.

Urias, V.E., Stout, W.M., Loverro, C., 2015. Computer network deception as a moving target defense. In: Security Technology (ICCST), 2015 International Carnahan Conference on. IEEE, pp. 1–6.

Vieira, M., Madeira, H., 2005. Towards a security benchmark for database management systems. In: 2005 International Conference on Dependable Systems and Networks (DSN'05). IEEE, pp. 592–601.

Villarreal-Vasquez, M., Bhargava, B., Angin, P., Ahmed, N., Goodwin, D., Brin, K., Kobes, J., 2017. An MTD-based self-adaptive resilience approach for cloud systems. In: Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on. IEEE, pp. 723–726.

Wahab, O.A., Bentahar, J., Otrok, H., Mourad, A., 2019. Resource-aware detection and defense system against multi-type attacks in the cloud: repeated Bayesian stackelberg game. IEEE Trans. Depend. Secure Comput..

Wang, H., Jia, Q., Fleck, D., Powell, W., Li, F., Stavrou, A., 2014. A moving target DDoS defense mechanism. Comput. Commun. 46, 10–21.

Wang, H., Li, F., Chen, S., 2016. Towards cost-effective moving target defense against DDoS and covert channel attacks. In: Proceedings of the 2016 ACM Workshop on Moving Target Defense. ACM, pp. 15–25.

Yang, C., Guo, Y., Hu, H., Wang, Y., Tong, Q., Li, L., 2019. Driftor: mitigating cloud-based side-channel attacks by switching and migrating multi-executor virtual machines. Front.Inf. Technol. Electron.Eng. 20 (5), 731–748.

Yang, Y., Cheng, L., 2018. An SDN-based MTD model. Concurren. Comput. e4897.

Zhang, L., Wang, Z., Fang, J., Guo, Y., 2016. A SDN proactive defense scheme based on IP and MAC address mutation. In: International Wireless Internet Conference. Springer, pp. 51–60.

Zhang, M., Wang, L., Jajodia, S., Singhal, A., Albanese, M., 2016. Network diversity: a security metric for evaluating the resilience of networks against zero-day attacks. IEEE Trans. Inf. Forensics Secur. 11 (5), 1071–1086.

Zhang, Y., Li, M., Bai, K., Yu, M., Zang, W., 2012. Incentive compatible moving target defense against VM-colocation attacks in clouds. In: IFIP International Information Security Conference. Springer, pp. 388–399.

Zheng, J., Namin, A.S., 2019. A survey on the moving target defense strategies: an architectural perspective. J. Comput. Sci. Technol. 34 (1), 207–233.

Zhuang, R., DeLoach, S.A., Ou, X., 2014. Towards a theory of moving target defense. In: Proceedings of the First ACM Workshop on Moving Target Defense. ACM, pp. 31–40.

Matheus Torquato is a Ph.D. candidate at the University of Coimbra. His research interests comprise subjects like Cloud Computing, Performance, Dependability, and Security Modeling. His current research focuses in the design and development of analytical models to evaluate performance, dependability, and security of moving target defense deployments in cloud computing. He received his Master Degree in Computer Science from the Federal University of Pernambuco. He is currently on leave from his teaching activities at the Federal Institute of Alagoas, Campus Arapiraca to pursue Ph.D. at the University of Coimbra. His website is http://www.matheustorquato.com.

Marco Vieira received the Ph.D. degree from UC, Portugal, in 2005. He currently is a Full Professor with the University of Coimbra, Coimbra, Portugal. His research interests include dependability and security assessment and benchmarking, fault injection, software processes, and software quality assurance, subjects in which he has authored or coauthored more than 200 papers in refereed conferences and journals. He has participated and coordinated several research projects, both at the national and European level. He has served on program committees of the major conferences of the dependability area and acted as referee for many international conferences and journals in the dependability and security areas.

Appendix B: VM Migration Scheduling as MTD against Memory DoS Attacks: An Empirical Study

In the following pages, we present the full content of our empirical observation of Virtual Machine migration as Moving Target Defense in a scenario with two different applications: a machine learning application and the TPC-C benchmark [Leutenegger and Dias, 1993].

VM Migration Scheduling as Moving Target Defense against Memory DoS Attacks: An Empirical Study

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

***Abstract*—Memory Denial of Service (DoS) attacks are easy-to-launch, hard to detect, and significantly impact their targets. In memory DoS, the attacker targets the memory of his Virtual Machine (VM) and, due to hardware isolation issues, the attack affects the co-resident VMs. Theoretically, we can deploy VM migration as Moving Target Defense (MTD) against memory DoS. However, the current literature lacks empirical evidence supporting this hypothesis. Moreover, there is a need to evaluate how the VM migration timing impacts the potential MTD protection. This practical experience report presents an experiment on VM migration-based MTD against memory DoS. We evaluate the impact of memory DoS attacks in the context of two applications running in co-hosted VMs: machine learning and OLTP. The results highlight that the memory DoS attacks lead to more than 70% reduction in the applications' performance. Nevertheless, timely VM migrations can significantly mitigate the attack effects in both considered applications.**

***Index Terms*—Memory DoS, Moving Target Defense, VM migration, Dynamic platform technique, Denial of Service**

I. Introduction

Moving Target Defense (MTD) consists of dynamically changing the available attack surface to thwart or defend from attacks [1], [2]. In virtualized environments, Virtual Machine (VM) migration appears among the most used MTD strategies [3], being preferred among other MTD techniques for several reasons, including i) VM migration is usually a native feature of virtualized environments; ii) it does not require expertise to deploy; iii) it is already a usual task of the virtualized environment management.

VM migration consists of moving the VMs in the available physical machines (PM) [4]. In the MTD context, VM migration is frequently applied to prevent malicious VMs from affecting the co-resident VMs or the underlying PM [5], [6] (i.e., prevent host-based attack success). For example, it is possible to use VM migration-based MTD to move benign clients away from a compromised PM [7].

Specifically, on the threats with potential defense through VM migration, we highlight the memory Denial of Service (memory DoS). Leveraging on issues in the hardware memory isolation [8], the attacker can run an attack against the memory of his own VM, trying to affect the co-resident VMs availability by overloading memory.

One of the critical factors to deploy an effective MTD against memory DoS is timing (i.e., when to apply MTD) [9]. In fact, in the context of VM migration against memory DoS, the MTD timing is still an open problem. Previous research tried to tackle this issue through stochastic modeling [10], [11]. However, these works neglect empirical investigation of VM migration-based MTD against memory DoS. Zhang et al. [12] refers to VM migration as a potential defense for memory DoS, but the authors use an alternative mitigation method based on *execution throttling*.

This practical experience report aims to fill this research gap through an experiment on VM migration-based MTD against memory DoS. We intend to investigate the impact of different scheduling of VM migration in the potential MTD protection. The following research questions guide this research:

- *RQ_1:* What is the impact of memory DoS in different applications running on co-resident Virtual Machines?
- *RQ_2:* Is Virtual Machine migration effective as Moving Target Defense against memory DoS?
- *RQ_3:* Does the Virtual Machine migration scheduling policy play a significant role in the mitigation of memory DoS effects?

The experimental approach is as follows. First, we set up an environment with two VMs, Attacker VM and Victim VM, running inside the same physical host. While the Attacker VM runs memory DoS attacks, the Victim VM runs benign applications, namely, an online transaction processing (OLTP) application benchmark and a machine learning (ML) application. Then, as MTD, we migrate the Attacker VM at different scheduling times, observing five-minute intervals. Note that these five-minute intervals are selected arbitrarily to

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

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

978-1-6654-2744-9/21/$31.00 ©2021 European Union

fit our experiment design. Here, we focus on understanding the VM migration-based MTD effectiveness against memory DoS instead of defining generic methods for selecting VM migration intervals. Besides that, the search for the critical threshold for VM migration is out of the scope of this paper.

Our results show that the memory DoS attacks performed do not interfere in the ML application accuracy, and effects are only noticeable in the ML time to fit metric[1]. In the ML application scenario, the VM migration is enough to clear the memory DoS effect). Regarding the OLTP application, we notice that delayed migration leads to cumulative service degradation. However, in all studied scenarios, the OLTP application stayed alive during the attack (i.e., the attack is not enough to crash the application). We present linear regression curves to help characterize the VM migration-based MTD protection in the context of this OLTP application.

This paper tries to fill a research gap by providing empirical evidence of the VM migration-based MTD effectiveness against memory DoS. We can highlight the following:

- Easy-to-reproduce experimental approach. We try to describe our methodology in detail to help researchers and system managers to reproduce the experiment in their environments. All the tools and source code are publicly available.
- We investigate the impact of memory DoS attacks in two different applications, namely OLTP and machine learning. From the investigation, we present a comprehensive set of results providing evidence of the effectiveness of VM migration-based MTD against memory DoS attacks.
- We consider the effects of an attack that is easy-to-launch, challenging to detect, and causes substantial performance degradation. Therefore, our study may help system managers to handle this relevant security threat.

The remainder of this paper is organized as follows. Section III presents the experiments. Section II presents the related work. Section IV presents the results. Section V briefly discusses the specific memory DoS severity. Section VI presents threats to validity and limitations of our work. Section VII concludes the paper.

II. Related Works

Our previous works [10], [11] provide VM migration as MTD evaluation based on modeling. These papers neglect experimental validation for the models. This practical experience report extends these works by providing the needed empirical background of VM migration as MTD.

The inspiring work of Zhang et al. [12] provided memory DoS background (including the attack source code). Li et al. [8] provided insights on the memory DoS attack detection. Although both papers mentioned VM migration as a potential defense for memory DoS, the authors followed different approaches from ours. Zhang et. al. [12] dealt with the problem using *execution throttling*. Li et al [8] focused on the memory DoS detection instead of MTD proposal.

Wang et al. [13] proposed a comprehensive framework for defending against co-resident threats. Their framework features a score calculation and attack-aware VM reallocation. Likewise, Liang et al. [14] defensive approach consists of a grouping-based VM placement strategy. In both papers, the authors validated their approaches using CloudSim. Unlike their works, we decided to deploy VM migration in a real testbed. Besides that, instead of proposing a new framework, our goal is to observe how the *off-the-shelf* VM migration MTD scheduling may protect the considered applications.

III. Experimental approach

Our main goal is to assess the impact of a memory DoS attack on applications running in co-resident VMs. Besides that, we aim to study whether different VM migration scheduling policies effectively mitigate possible effects of memory DoS.

Figure 1 presents the experimental testbed, which includes two physical machines: Source Node (Intel Xeon E5-2620 2.00GHz + 16GB of RAM with Error Correction Code enabled) - main host for the Victim VM and Attacker VM; Target Node (Intel Core i7-9700 3.00GHz + 16GB of RAM) - host for the Attacker VM migration. Both the Attacker VM and the Victim VM are Kernel Virtual Machine (KVM)[2] VMs with a homogeneous configuration: single-core processor + 3 GB of RAM. The Source Node and the Target Node run Ubuntu Server 20.04.2 with kernel 5.4.0-72 and KVM 4.2.1.

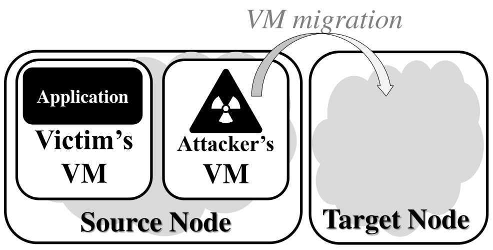


Fig. 1. Testbed architecture

We consider two different applications running in the Victim VM. The first consists of an ML application for face recognition based on an example of scikit learn python library[3]. We provide an example script for the ML application automation in [15], which produces an output file with the accuracy and time to fit metrics. The second is an OLTP application based on the TPC-C[4] benchmark [16]. Specifically, we use the CockroachDB tool [17] for TPC-C benchmark automation.

Regarding the specific attack running in the Attacker VM, we follow the approach presented by Zhang et al. [12]. In

[1]The fitness function is the main task of our ML application. Indeed, the time spent on other tasks is negligible. Thus, the time to fit is roughly the ML processing time.

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

[3]https://scikit-learn.org/stable/auto_examples/applications/plot_face_recognition.html

[4]http://www.tpc.org/tpcc/

practice, we use an infinite loop of unaligned atomic accesses to the main memory of the VM. This attack load generates LOCK signals in the memory, whose accumulation may result in memory unavailability to handle benign processes. Henceforth in this paper, we refer to this attack as *unalignAttk*.

We performed sets of 30-minutes experiments, during which the ATTACKER VM performs *unalignAttk* attacks and the VICTIM VM runs the OLTP or the ML application. For comparison purposes, we present results for three scenarios: i) *golden run* - experiments without attacks and MTD; ii) *MTD* - experiment adopting VM-migration as MTD against *unalignAttk*; and iii) *Only attack* - considering the *unalignAttk* impact while the MTD is off.

IV. CASE STUDIES

This section presents our two case studies. As mentioned earlier, in these case studies, we consider two different applications running inside the VICTIM VM, namely a machine learning application and an OLTP application benchmark (TPC-C benchmark).

A. Machine Learning application

We divided the experiment with the ML application into two steps. First, the *attack severity experiment*, focusing only on investigating the impact of *unalignAttk* on the application (i.e., system without MTD). Second, the *MTD experiment*, where we apply VM migration scheduling as MTD. The former aims to provide an answer to *RQ1* (impact of memory DoS in different applications running on co-resident Virtual Machines), and the latter provides an answer for *RQ2* (Virtual Machine migration effective as Moving Target Defense against memory DoS) and for *RQ3* (role played by the Virtual machine migration scheduling policy in the mitigation of memory DoS effects).

The results (see Figures 2 and 3) include the *golden run* (system without attack) and the *unalignAttk* (system under attack). The X-axis corresponds to the experiment timeline.

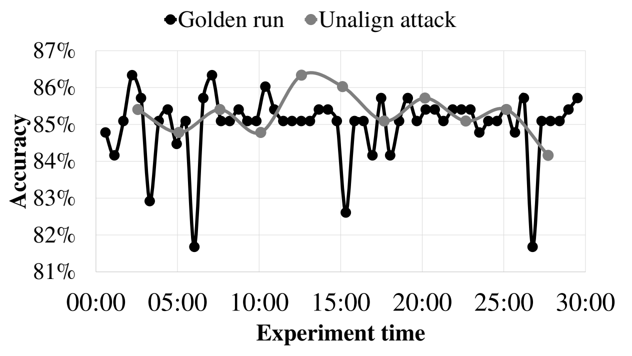


Fig. 2. ML - Accuracy results - attack severity experiment

Figure 2 shows that accuracy is close to 85% for all the ML observations. In both curves (*golden run* and *unalignAttk*), we notice the expected accuracy oscillations over time. These results suggest that considering the scope of our experiments, the *unalignAttk* does not interfere with the accuracy of the ML application. Therefore, as long as the system is up, even in the presence of an *unalignAttk* attack, the ML application results accuracy stays roughly at the same levels of the *golden run*.

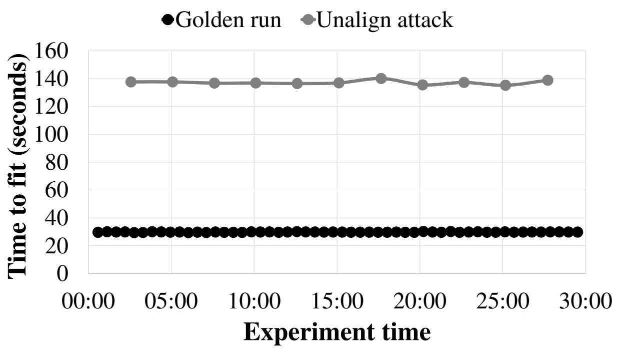


Fig. 3. ML - Time to fit (seconds) results - attack severity experiment

Time to fit results in Figure 3 show how long the ML application takes to process the face recognition. These results highlight a substantial difference between the *golden run* and the *unalignAttk*. Indeed, the ML application processing under *unalignAttk* lasts about four times more than the *golden run*. These results suggest that the *unalignAttk* impairs the ML application performance substantially, reducing the overall number of ML application runs in a given period. Although the *unalignAttk* does not cause a catastrophic failure of ML application, its impact may reflect in the application availability. Depending on the Service Level Agreements (SLA) and some threshold levels, the ML application may be considered unavailable when it takes so long for processing.

In summary, we noticed that the major impact is indeed in the performance and not in the ML accuracy. The *unalignAttk* causes a 460% increase in the time to fit when compared to the *golden run*. The number of ML runs in our 30-minute experiment is of 54 in the *golden run* and 11 in the *unalignAttk*, meaning a 80% reduction. Table I presents a summary of ML application *attack severity experiment*.

The second step of this experiment is the *MTD experiment*, in which we deploy three different schedules of VM migration, namely at the 5th, 10th, and 15th minute of the experiment time. In the same way as the *attack severity experiment*, we noticed that among the regular oscillations, the accuracy results for all the MTD scenarios also approach 85%. Nevertheless, the time to fit results presents more interesting behavior as shown in Figure 4.

These results suggest two conclusions. First, the VM migration MTD is effective to clear *unalignAttk* effects. Second, delayed or premature VM migrations immediately recover the ML application to the *golden run* levels. However, the longer the attack continues, the worse is the performance impact. Note that the ML application is a standalone application without a timeout parameter. VM migration timing is crucial for system availability in more complex client-server scenarios,

TABLE I
ATTACK SEVERITY RESULTS SUMMARY - MACHINE LEARNING APPLICATION

Experiment	Number of runs	avg. accuracy	std. dev. accuracy	avg. time to fit	std. dev. time to fit
golden run	54	85.00%	0.0091	29.7702	0.18283
unalignAttk	11	85.29%	0.0058	137.18	1.31916

TABLE II
MTD RESULTS SUMMARY - MACHINE LEARNING APPLICATION

Experiment	Number of runs	avg. accuracy	std. dev. accuracy	avg. time to fit	std. dev. time to fit
OnlyAttack	11	85.29%	0.0058	137.18	1.3192
MigAt5thMin	46	85.15%	0.0072	34.92	22.6595
MigAt10thMin	39	84.93%	0.0109	41.79	32.9165
MigAt15thMin	33	85.22%	0.0053	49.17	40.5078

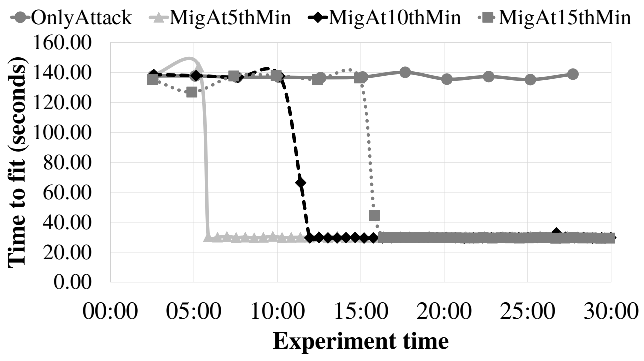


Fig. 4. ML - Time to fit (seconds) results - MTD experiment

as delayed VM migrations may allow the client to accumulate server timeout, leading the client to give up the connection.

Table II presents a summary of the ML application *MTD experiment*. The high standard deviation variance in the time to fit MTD scenarios is due to the abrupt change in the time to fit after VM migration.

B. TPC-C benchmark

Unlike the ML application experiment, where we collect the metrics from every ML run, a 30-minute experiment run provides only a single result of the TPC-C benchmark. Therefore, we need to run a set of 30-minutes TPC-C evaluations. The following results are obtained from 30 runs of *golden run* and 30 runs of *OnlyAttack* (i.e., system without MTD). We run 15 experiments for the MTD results (three for each migration schedule).

Here, we focus in two metrics: **efc(%)** - how close the results are to the theoretical maximum TPC-C performance, and **avg (ms)** - average time for transaction processing in milliseconds. To these, we added two metrics: **tpmC** - TPC-C specific metric to measure the *business throughput* (i.e., number of orders processed per minute), and **ops** - total number of transactions processed.

In the TPC-C benchmark results, we merged all the scenarios into the same plot. This approach is helpful to notice the degradation due to delayed migrations. Therefore, the plots have *golden run* results at the origin, meaning the VM migration at the 0th minute (i.e., system without attack), and *OnlyAttack* results at 30th minute (i.e., system without MTD as each experiment lasts 30 minutes). We perform experiments with VM migration at the 5th, 10th, 15th, 20th, and 25th minute of experiment time.

Figures 5 and 6 presents the results for *avg (ms)* and *efc(%)*, respectively. The plots include the error bars for each scenarios and a linear regression curve. In both cases, the linear regression curves $R-squared$[5] is above 0.99.

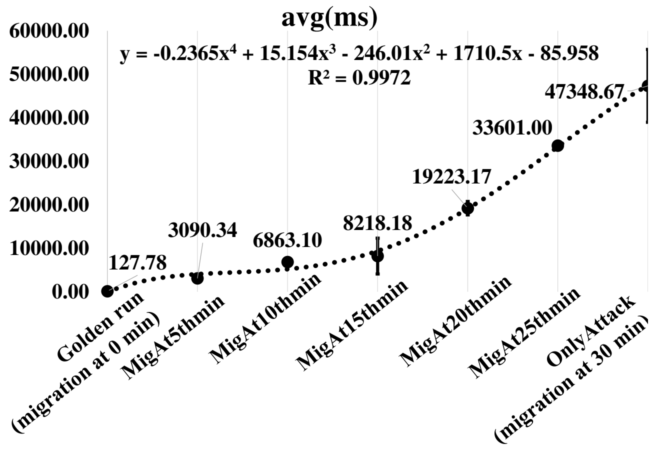


Fig. 5. TPCc - avg (ms)

We notice the expected behavior of *avg (ms)* increasing and *efc* decreasing when we have delayed MTD actions. Specifically about our first research question related to memory DoS impact (i.e., comparison between *golden run* and *Only Attack*), the decrease in the *efc* is of 68.39%, while the increase in the *avg (ms)* approach 370%. These results reveal the service degradation rate due to memory DoS attack effects accumulation.

The linear regression curves are particularly useful to estimate *avg (ms)* and *efc* with other intervals for VM migration. For illustration, we obtain that, to preserve *efc* above 75%, VM migration should occur before 7.5 minutes, and to keep

[5] $R-squared$ (R^2) is a measure that corresponds to the proportion of the variance explained by regression curve.

TABLE III
TPC-C BENCHMARK EXPERIMENT RESULTS

Experiment	avg. tpmc	std. dev. tpmc	efc(%)	std. dev. efc(%)	avg (ms)	std. dev. avg (ms)	ops	std. dev. ops
golden run	123.5	0.4814	96.00	0.0038	127.78	9.4855	8554	31.5369
MigAt5thMin	105.2	1.7688	81.81	1.3809	3090.34	597.9909	7281	102.9208
MigAt10thMin	91.5	1.0497	71.40	0.8155	6863.10	157.8724	6358	75.8650
MigAt15thMin	65.2	6.3168	50.68	4.8976	8218.18	4123.8594	4542	445.1576
MigAt20thMin	58.9	4.4500	45.77	3.4567	19223.17	1545.3677	4060	305.2223
MigAt25thMin	46.7	1.1518	36.30	0.9092	33601.00	983.7309	3273	75.5484
OnlyAttack	35.5	6.5807	27.61	5.1120	47348.67	8384.8285	2461	446.3500

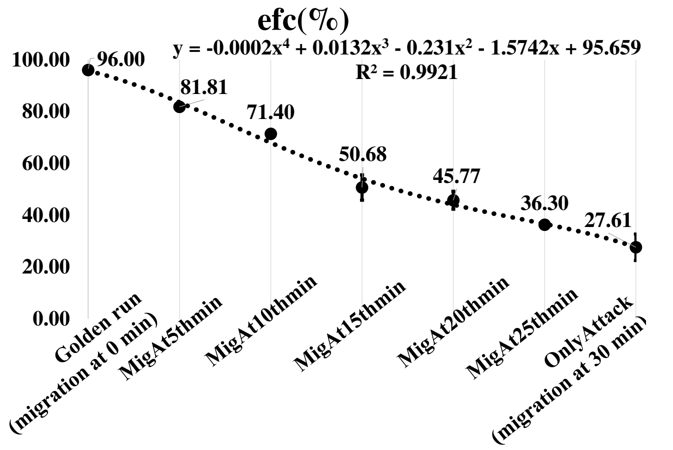


Fig. 6. TPCc - efc(%)

the *avg (ms)* below 15000, VM migration should occur before the 18th minute.

Table III summarizes the TPC-C benchmark results. It includes the results for the metrics and their standard deviation. Specifically about tpmC, one of the metrics of interest in the TPC-C benchmark, the reduction is 71.25% (comparison between *golden run* and *OnlyAttack*).

V. SEVERITY OF *UnalignAttk* ATTACKS

In the previous section, we highlighted the *unalignAttk* impact on the applications. However, it is crucial to investigate whether the effect is only due to a manageable resource overhead. The question (Q) is *Isn't the impact observed already expected due to unalignAttk resource overhead?*.

To answer Q, we propose an observation of *unalignAttk* resources overhead and its comparison with benign workloads. As we are dealing with memory, we set up a workload based on the Linux memtest[6]. We also added the workloads considered in the case studies, namely, TPC-C and Machine learning (ML) applications. In practice, we observed the SOURCE NODE resource consumption while hosting one VM running the workload and one VM in idle state. The VM's configuration is the same used in the previous case studies. The considered workloads are: *golden run* (i.e., two VMs in idle state), *unalign attack*, *memtest*, *TPC-C* and *machine learning application (ML)*. The observation in all scenarios lasts 30 minutes. The results from the CPU and memory usage are presented in Figures 7 and 8, respectively.

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

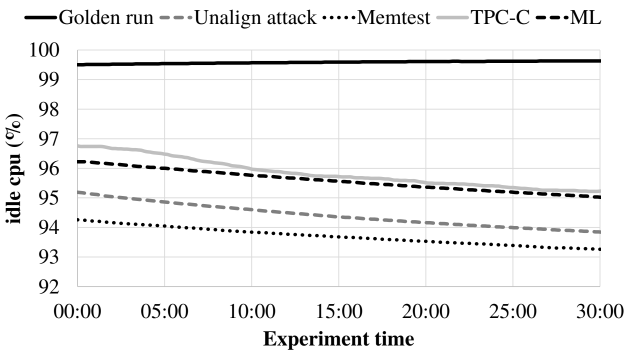


Fig. 7. Percentage of idle CPU

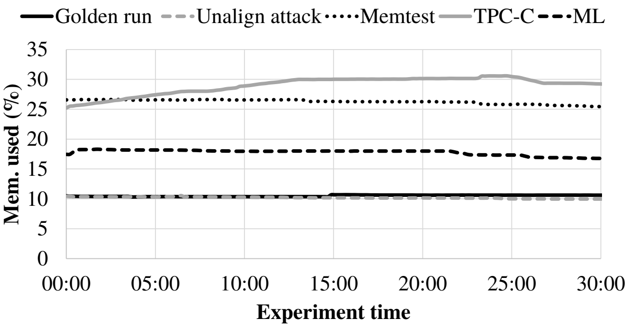


Fig. 8. Memory usage (%)

We notice that, *memtest* requires more SOURCE NODE resources than the *unalignAttk*. Presumably, considering only the resource consumption, the possible impact of *memtest* in the VICTIM VM should be higher than the *unalignAttk* impact. To verify this, we run four one-hour experiments combining the *memtest* and *unalignAttk* running in the ATTACKER VM with the ML application and the TPC-C benchmark running in the VICTIM VM. Table V and IV show the results.

The results highlight that the *unalignAttk* produces a higher impact on the applications when compared to the *memtest*. Actually, the *memtest* results are close to the *golden run* results. While the *memtest* causes a reduction (when compared to the *golden run*) of 22.2% in the TPC-C benchmark *efc* metric, the *unalignAttk* produces a 79.1% reduction. About the ML application, *memtest* reduces the number of runs in 7.5%,

TABLE IV
COMPARISON BETWEEN *unalignAttk* AND *memtest* - TPC-C METRICS

Attack	tpmC	efc	avg (ms)	ops(total)
golden run	124.4	96.7%	126.4	17202
memtest	95.9	74.5%	7385.3	13263
unalign	22.6	17.6%	82225.5	3131

TABLE V
COMPARISON BETWEEN *unalignAttk* AND *memtest* - ML METRICS

Attack	Number of runs	Avg. accuracy	Avg. time to fit(s)
golden run	40	85.05%	29.87
memtest	37	85.10%	34.02
unalign	19	84.88%	125.26

while the *unalignAttk* reduction is of 52.5%. The results of this experiment confirms the findings of [12] which highlights memory DoS attack severity.

VI. THREATS TO VALIDITY AND LIMITATIONS

We identified two main threats to the validity of our results, and they lie in our experiment design: 1) the observed results are obtained from a small architecture; 2) limited observation time in all the experiments. In the best scenario, we should run more extended experiments in bigger datacenters. We are aware of these limitations. However, below we provide some explanations for mitigating these threats.

Threat 1) We manage to dedicate a small but powerful setup for our experimentation. Note that the PMs have 16GB of RAM with 6 and 8-core processors. To scale a representative scenario, our VMs have only 3 GB of RAM each, with a single-core processor. Therefore, there are plenty of idle PM resources while running the VMs simultaneously, taking less than 40% of the available resources.

Threat 2) Note that each experimentation does not involve only the 30 minutes of the workload. We need to clean the system for each run of the experiment, meaning generate new VM images, complete PM OS reboot, export filesystem for VM migration, and create new VMs. Besides that, all the experiment runs are sequential, which means that it was impossible to paralleling the experiment runs (as we have only a single testbed).

VII. CONCLUSION

This paper presented a practical experience report of VM migration scheduling as MTD against memory DoS attacks. We evaluated the memory DoS attack severity and the MTD effectiveness in different scenarios. Namely, we considered memory DoS attacks against i) a machine learning application and ii) the TPC-C benchmark. Our results show that the memory DoS attack causes a significant impact on the applications. Besides that, results suggest that the VM migration-based MTD effectively reduces the effect of memory DoS attacks in both applications.

This work fills a research gap of lack of empirical evidence of VM migration-based MTD effectiveness against memory DoS. We are aware that the results are limited to our system architecture. However, the adopted tools and source code are publicly available. Thus, it is possible to reproduce the proposed methodology in other scenarios. Hopefully, system managers and researchers may find our approach useful to support MTD experimentation and MTD policy design.

In the future, we aim to reproduce the experiments in a bigger datacenter comprising other representative workloads as client-server applications. Besides that, we intend to run more extended experiments to notice possible error accumulation after sequenced VM migration.

REFERENCES

[1] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, *Moving target defense: creating asymmetric uncertainty for cyber threats*. Springer Science & Business Media, 2011, vol. 54.

[2] J.-H. Cho, D. P. Sharma, H. Alavizadeh, S. Yoon, N. Ben-Asher, T. J. Moore, D. S. Kim, H. Lim, and F. F. Nelson, "Toward proactive, adaptive defense: A survey on moving target defense," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 709–745, 2020.

[3] M. Torquato and M. Vieira, "Moving target defense in cloud computing: A systematic mapping study," *Computers & Security*, vol. 92, p. 101742, 2020.

[4] P. G. J. Leelipushpam and J. Sharmila, "Live vm migration techniques in cloud environment—a survey," in *2013 IEEE Conference on Information & Communication Technologies*. IEEE, 2013, pp. 408–413.

[5] H. Wang, F. Li, and S. Chen, "Towards cost-effective moving target defense against ddos and covert channel attacks," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, 2016, pp. 15–25.

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

[7] Q. Jia, H. Wang, D. Fleck, F. Li, A. Stavrou, and W. Powell, "Catch me if you can: A cloud-enabled ddos defense," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2014, pp. 264–275.

[8] Z. Li, T. Sen, H. Shen, and M. C. Chuah, "Impact of memory dos attacks on cloud applications and real-time detection schemes," in *49th International Conference on Parallel Processing-ICPP*, 2020, pp. 1–11.

[9] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang, and S. Kambhampati, "A survey of moving target defenses for network security," *IEEE Communications Surveys & Tutorials*, 2020.

[10] M. Torquato, P. Maciel, and M. Vieira, "Security and availability modeling of vm migration as moving target defense," in *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2020, pp. 50–59.

[11] ——, "Analysis of vm migration scheduling as moving target defense against insider attacks," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, 2021, pp. 194–202.

[12] T. Zhang, Y. Zhang, and R. B. Lee, "Dos attacks on your memory in cloud," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 253–265.

[13] X. Wang, L. Wang, F. Miao, and J. Yang, "Svmdf: A secure virtual machine deployment framework to mitigate co-resident threat in cloud," in *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2019, pp. 1–7.

[14] X. Liang, X. Gui, A. Jian, and D. Ren, "Mitigating cloud co-resident attacks via grouping-based virtual machine placement strategy," in *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2017, pp. 1–8.

[15] M. Torquato and M. Vieira. (2021). [Online]. Available: https://github.com/matheustor4/scriptML

[16] S. T. Leutenegger and D. Dias, "A modeling study of the tpc-c benchmark," *ACM Sigmod Record*, vol. 22, no. 2, pp. 22–31, 1993.

[17] CockroachDB. (2021). [Online]. Available: https://www.cockroachlabs.com/docs/v20.1/performance-benchmarking-with-tpc-c-10-warehouses

Appendix C: An Experimental Study of Software Aging and Rejuvenation in dockerd

In the following pages, we present the full content of the paper presenting our experimental study of software aging in the Docker platform.

2019 15th European Dependable Computing Conference (EDCC)

An Experimental Study of Software Aging and Rejuvenation in *dockerd*

Practical Experience Report

Matheus Torquato*†, Marco Vieira*
*Department of Informatics Engineering, University of Coimbra, Coimbra, Portugal
†Federal Institute of Alagoas, Campus Arapiraca, Arapiraca, Brazil
matheus.torquato@ifal.edu.br, matheustor4.professor@gmail.com*, mvieira@dei.uc.pt*

***Abstract*—Virtualized containers are being extensively used to host applications as they substantially reduce the overhead caused by conventional virtualization techniques. Therefore, as containers adoption grows, the need for dependability also increases. *Dockerd*, the process that is in charge of Docker containers management, is supposed to support long-running systems, which makes it prone to the well-known problem of software aging. This paper presents an experimental study of software aging and rejuvenation targeting the *dockerd* daemon. We used the SWARE approach to conduct the experimentation, which encompasses three phases: i) *stress* - stress environment with the accelerated workload to induce bugs activation; (ii) *wait* - stop the workload submission to observe possible accumulated effects and; (iii) *rejuvenation* - submit a rejuvenation action to perceive changes in the internal software state. The experiment runs for 26 days, and results show that *dockerd* suffers from software aging effects after the stress phase. The accumulated effects remain in the system until a complete cleanup, comprising removing all the containers and rebooting the operating system.**

***Index Terms*—Software aging and rejuvenation, Container, Docker, Dependability, SWARE approach.**

I. Introduction

Through containers, system developers can design software in the local environment and export it to run identically in other environments [1]. Containers have a lower virtualization overhead than Virtual Machines, as they use fewer layers between applications and hardware resources. Containers are usually smaller than Virtual Machines, and it is possible to run hundreds of them on a single physical machine [2]. Besides that, containers also facilitate the development of microservice architectures through tools as containers orchestration. Reputed companies as Netflix and SoundCloud leverage from microservices architectures [3]. As containers are now part of wider environments, their usage may introduce new challenges for attaining high levels of system dependability.

A previous survey showed that Docker containers are used to host diverse applications, ranging from medical to space research. Docker is also used in banks, insurance companies, and other scientific use cases [4].

The increasing adoption of Docker containers raises concerns regarding system dependability, which calls for further works on the evaluation of Docker dependability aspects. One of the core components of the Docker architecture is the *dockerd* daemon. *Dockerd* receives commands from the user and manages other Docker architecture components. As *dockerd* is supposed to provide very long execution times, it may be subject of one relevant dependability problem known as software aging (more details about software aging in the Section II).

The main research question target by this work is: *Does* dockerd *daemon suffer from any software aging effect?*. To answer this, we conducted an experiment of software aging and rejuvenation on the *dockerd* daemon. In practice, we applied the SWARE approach to support the tests [5], which are driven by three fundamental principles: i) stress the system with a heavy workload to accelerate the activation of possible aging-related bugs; ii) observe system reaction after stress to perceive possible aging effects accumulation; and iii) once software aging is detected, apply a rejuvenation action and verify its effectiveness. More details about the SWARE approach are in Section III-B.

Results suggest that the *dockerd* daemon suffers from software aging (Section IV). We draw this conclusion from the behavior of the system after an accelerated workload exposure. We noticed that the levels of resources consumption, mainly memory, and virtual memory, tends to stay degraded even when the workload exposure ceases. The system only recovers to a stable resources consumption levels after a complete cleanup which comprises the removal of all containers and an Operating System reboot.

We also observed a *dockerd* catastrophic failure during the experiment. After this failure, *dockerd* became unresponsive, and some brute-force actions were needed to recover the system. More details are in the Section IV-B.

The main contributions of this paper are:

- A structured study of software aging on the *dockerd* daemon;
- A set of results that highlight *dockerd* software aging weaknesses;
- A report of a *dockerd* failure and its workaround.

Up to our knowledge, our research is the first to investigate software aging and rejuvenation in Docker architectures. Our results highlight evidence of software aging in the *dockerd* daemon. Leveraging the results, presented here, the Docker users' community can design policies to minimize possible

978-1-7281-3929-6/19/$41.00 ©2019 IEEE
DOI 10.1109/EDCC.2019.00014

Authorized licensed use limited to: b-on: Universidade de Coimbra. Downloaded on September 13,2023 at 23:00:11 UTC from IEEE Xplore. Restrictions apply.

software aging effects as timely rejuvenation actions. Besides that, the reported *catastrophic* failure in *dockerd* may also brings useful information for Docker platform management.

The remainder of this paper is organized as follows. Section II presents concepts about software aging and rejuvenation. Section III provides details about the experimental study, including information about the Docker architecture, the SWARE approach, and our experiment. Section IV comprises the results and discussion and a mini-report of *dockerd* failure and repair. In Section V, we discuss possible threats to validity. Finally, Section VI presents conclusions and ideas for future work.

II. Software Aging and Rejuvenation

Software aging is the result of the accumulation of *aging-related* bugs effects. *Aging-related* bugs are similar to *Heisenbugs* [6], whose activation may occur after a long time of software execution. In practice, *aging-related* bugs activation occur when the system reaches specific conditions (e.g. lack of computational resources), which are usually difficult to reproduce [7].

Software rejuvenation is the countermeasure for software aging. Huang et al. [8] presented the first definitions of software rejuvenation. The work defines rejuvenation as a proactive technique to prevent aging effects from reaching critical levels. The rejuvenation actions usually rely on restarting an application to conduct it to a clean state, thus eliminating the accumulation of aging effects.

There are several works presenting software aging and rejuvenation evaluations in virtualized environments, including both modeling and experimental evaluation approaches. The evaluations based on modeling usually aim to provide specific software rejuvenation schedules to maximize system availability [9] [10]. In the experimental perspective, we can find works about software aging and rejuvenation in: KVM [11], Xen [12], OpenStack [13], and Virtual Machine migration as KVM software rejuvenation [14]. To the best of our knowledge, our work is the first to explore software aging and rejuvenation in *dockerd* daemon.

III. Experimental Study

In this section, we describe the details of our experimental study, including details of the Docker platform, SWARE approach, and the considered experimental setup.

A. Docker

Docker[1] is an open source platform to deploy and manage virtualized containers. Figure 1 presents a simplified view of the Docker architecture. As shown, the *dockerd* daemon is responsible for receiving commands for the user through an API and for managing Docker objects like images and containers. Depending on the command, *dockerd* also communicates with `Registry`, which is a repository of Docker images, to obtain new images. *Dockerd* daemon is at the core of the Docker architecture. Thus, its failure directly affects system manageability.

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

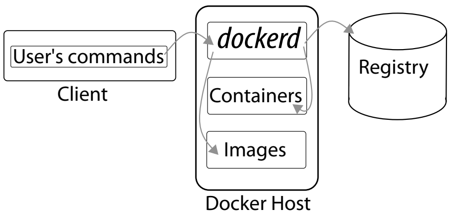


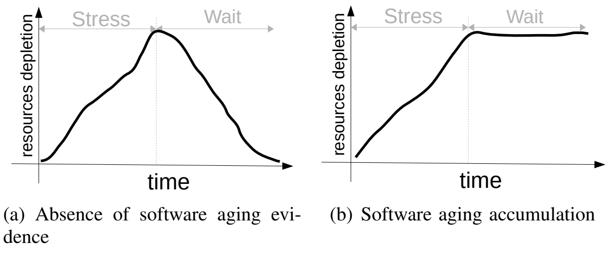
Fig. 1. Simplified Docker Architecture

B. SWARE approach

A previous work [5] presented the SWARE approach as a support for software aging and rejuvenation experiments. The authors applied SWARE to investigate software aging in the Kernel Virtual Machine (KVM) software and to verify the effectiveness of Virtual Machine migration as a software rejuvenation.

The goal of the SWARE approach is to allow the characterization of software aging indicators and software rejuvenation effectiveness on a single experiment. It includes three phases: **St**ress, **Wa**it and **Re**juvenation. The Stress phase is the period when the system is exposed to a heavy workload. The main goal of Stress phase is to activate possible *aging-related* bugs that may exist in the system. To achieve this goal, the workload submitted in the Stress phase forces the system to pass through different levels of utilization. The main characteristics of the workload are i) heavy intensity - to accelerate *aging-related* bugs activation (if they exist), and ii) direct relation with the tested component - to ensure that the possible software aging accumulation is in the desired component. The specific workload used in our experiment is described in Section III-C.

The Wait phase serves to find out possible effects of *aging-related* bugs activated in the Stress phase. This phase is started when the workload exposure ceases. Therefore, any software aging effect accumulated due to the workload execution may be observed in this phase. Otherwise, if the system recovers without a rejuvenation action, there is an absence of software aging evidence. Figure 2 summarizes the expected behavior of the Wait phase.

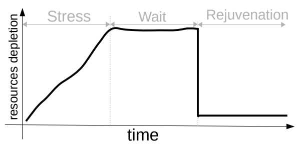


(a) Absence of software aging evidence (b) Software aging accumulation

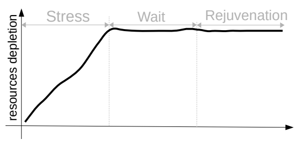
Fig. 2. Wait phase expected behavior

Authorized licensed use limited to: b-on: Universidade de Coimbra. Downloaded on September 13,2023 at 23:00:11 UTC from IEEE Xplore. Restrictions apply.

Finally, the Rejuvenation phase aims at highlighting if the proposed rejuvenation action is effective or not. After the end of the Wait phase, we perform a rejuvenation of the environment. If the system returns to a stable state, the rejuvenation is effective. Figure 3 depicts the expected behavior for the Rejuvenation phase.



(a) Proposed rejuvenation is effective



(b) Proposed rejuvenation is innocuous to counteract detected software aging

Fig. 3. Rejuvenation phase expected behavior

In the SWARE approach, the duration of each phase is not strictly defined. However, the recommendation is to have Wait and Rejuvenation phases long enough to ensure the detection of software aging accumulation and the rejuvenation effectiveness (which depends on the system being tested).

C. Experimental Setup

We run the experiments on a machine with the following configuration: Intel Core i5 CPU 650 (3.20GHz), 8 GB of RAM, 1 GB of Swap memory. The machine runs the **Docker version 18.09.0, build 4d60db4** on a Ubuntu Server 16.04.5 (64 bit) with Linux kernel 4.4.0-142.

The first step to deploy SWARE is to select a workload to stress the *dockerd* daemon. As mentioned before, to avoid side-effects and doubts in the software aging detection, the selected workload has to be directly related to the *dockerd* daemon. Thus, we decided, after some preliminary tests, to use a simple workload which consists of a stress loop of creation and deletion of containers. A similar idea was used in an experiment of software aging in Cloud [15]. We highlight that the stress workload is not intended to represent a real trace. Instead, it is intended to stress the system to pass through different levels of resources consumption. Our guidelines in the workload selection are i) workload heavy enough to stress the system and ii) workload light enough to avoid premature system failures due to resources exhaustion. Then, we defined the cycle of creation and deletion of 50 containers. We also added some wait times to avoid peak overload during workload execution. We set 10 seconds of interval between containers instantiation, and 50 seconds after containers deletion. As containers deletion may require more computing power, the 50-second wait period allows the system to deallocate used resources. We used a public container image *httpd* in our tests. Algorithm 1 presents the pseudocode of the proposed software aging workload.

The duration of the Stress and Wait phases have to be sufficiently long to achieve their goals. In practice, the Stress phase has to last enough to lead the system to pass through several levels of system utilization, trying to accelerate aging-bugs activation, while the Wait phase has to be long enough to highlight if there is a persistent degraded state caused by the workload submitted in the Stress phase. We set the duration of the Stress and Wait phases to 10 days, because, as the stress workload is heavy, ten days may suffice to activate aging-related bugs. Besides that, the longer duration may activate software aging in other software components, which may impair our desired analysis.

Algorithm 1 Software aging workload

```
loop
    while InstantiedContainers $< 50$ do
        InstantiateContainer(httpd);
        Wait(10 seconds);
    end while
    KillAllContainers();
    Wait(50 seconds);
end loop
```

Dockerd presented a catastrophic failure during the workload exposure (more details in Section IV-B), reducing the Stress Phase to about 7.5 days and enlarging the Wait Phase to 12.5 days. Due to that *dockerd* failure, the software aging workload script did not finish correctly and two *httpd* remaining running during the Wait Phase. The results of the Wait phase are the most sensitive for software aging detection because they highlight the accumulated effects from the Stress phase. So, we decided to not interfere with the system to manually delete the two remaining containers. Besides that, we later show in this paper a comparison of a system with two containers under normal conditions (i.e., the absence of software aging accumulation) with the results of the Wait phase.

Dockerd recovery (also described in Section IV-B) marks the start of the Rejuvenation Phase, which lasted for 6 days. Figure 4 represents the phases and duration of the experiment. The experiment reported in this paper is our second run using the same methodology. We highlight that the results are similar in both runs.

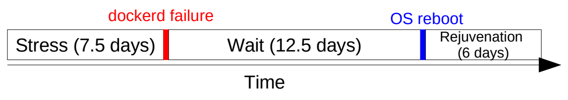


Fig. 4. Experiment phases duration

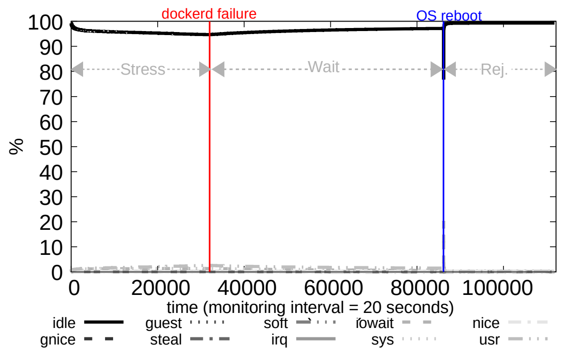
IV. Results and Discussion

We divided this section in two: the first presents the software aging and rejuvenation experiment results and discussion, and the second is a mini-report of the experienced *dockerd* failure and our adopted workaround.

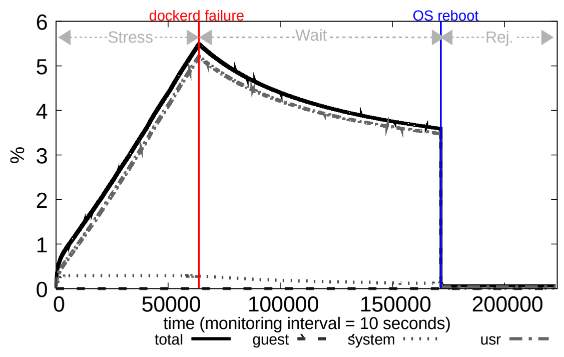
A. Software aging and rejuvenation experiment

Figure 5 presents the results of the experiment. We divided the figure into two columns: the left side is related to the resource consumption on the physical machine (Node) that runs the *dockerd*, and the right side presents the specific data

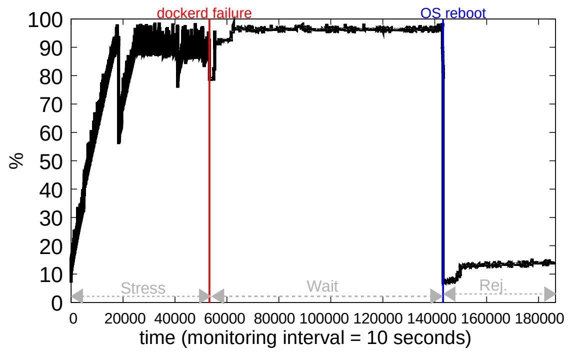
Authorized licensed use limited to: b-on: Universidade de Coimbra. Downloaded on September 13,2023 at 23:00:11 UTC from IEEE Xplore. Restrictions apply.



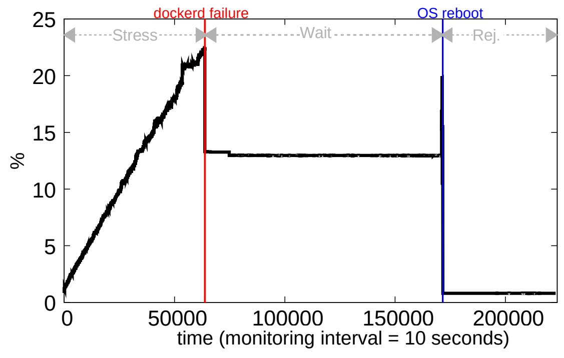
(a) CPU Usage - Node



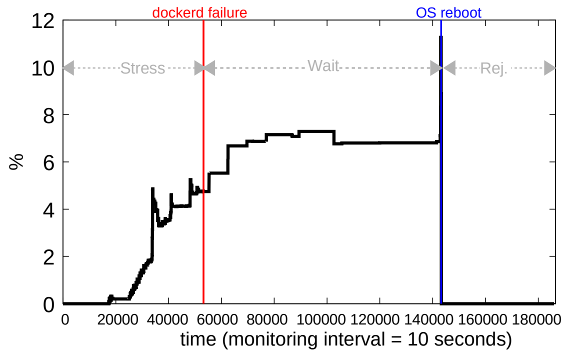
(b) CPU Usage - dockerd process



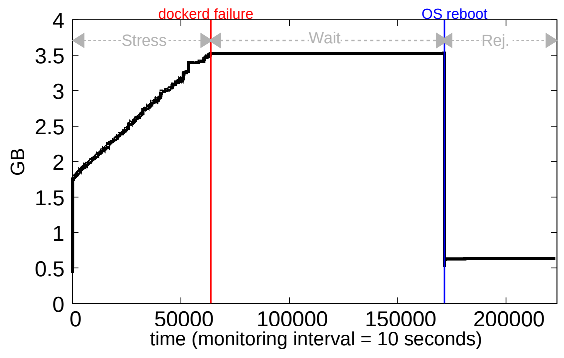
(c) Memory Usage (%) - Node



(d) Memory usage (%) - dockerd process



(e) Swap Usage - Node



(f) Memory usage - VSZ - dockerd process

Fig. 5. Software aging and rejuvenation experiment results

from the *dockerd* process monitoring. Each plot has markers for each experiment phase. We highlighted the *dockerd* failure time (end of the Stress Phase) with a red line and the Operating System reboot with a blue line (end of the Wait Phase). We used the tools `mpstat`, `sar` and `pidstat` for resources monitoring. The monitoring interval for each resource is in the x-axis label of each plot.

Figures 5(a) and 5(b) present the CPU usage results for both, Node and *dockerd*. These plots have all the values from the `mpstat` and `pidstat`, respectively. The impact of the software aging workload in the *Node* CPU usage is negligible (Fig 5(a)). It is possible to notice that the processor stays idle in the majority of the time. The CPU usage for the *dockerd* process (Fig 5(b)) increases during the Stress phase. Furthermore, *dockerd* still requires processing power during the Wait phase, which means that the process is still using the processor, even without a workload being submitted. As mentioned earlier, the system remains with two containers running in the Wait phase. Therefore, these containers may be causing the CPU usage overhead. But, we highlight that the behavior of the Wait phase suggests software aging as *dockerd* releases the processor after a software rejuvenation routine which comprises containers deletion, *dockerd* cleanup and Operating System reboot. To support the investigation of software aging evidence in *dockerd*, we compared the results of the *Wait* phase with a system with two containers in normal

Authorized licensed use limited to: b-on: Universidade de Coimbra. Downloaded on September 13,2023 at 23:00:11 UTC from IEEE Xplore. Restrictions apply.

(i.e., non-aging) conditions.

Memory results are the most significant in software aging research, as software aging effects are usually related to memory leaks. Figures 5(c) and 5(d) present the memory usage of the Node and *dockerd* process, respectively. As expected, we notice an increasing usage of memory resources during the Stress Phase.

Both Node and *dockerd* results show that the memory usage remains high during the entire Wait phase. *Dockerd* virtual memory (VSZ) usage follows the same behavior (Figure 5(f)). This persistent degraded state corroborates software aging existence, as the system does not return to normal levels of memory usage without software rejuvenation.

Node swap (Figure 5(e)) results show a somewhat increasing usage in the Wait phase. This behavior suggests that the Node needs to resort to disk space to handle the accumulated effects.

The resources depletion ceases after software rejuvenation. The high levels of resources consumption before OS reboot are due to the cleanup process needed (more details in the Section IV-B) for software rejuvenation.

To go deeper in the analysis, we compared the Wait phase results, where the system is supposedly suffering from software aging (let's call it an aged system), with the results of a system without software aging accumulation (let's call it a clean system). This comparison may highlight if the degradation observed in the Wait phase is related to software aging or not.

To perform a fair comparison, we configured the clean system considering the context of the aged system when it was in the Wait phase. Thus, we created two *httpd* containers in the clean system and collected the resources consumption for two days (Baseline). We did not submit any workload during this. The results obtained are presented in Figure 6. The left-hand presents the results from the Wait phase (of the aged system) and the right-hand has the results of the baseline configuration (clean system). We omit the Node results to focus only on the *dockerd* process behavior.

The comparison highlights that the system indeed accumulates software aging effects in the Stress phase. Clean system results present lower levels of all resources consumption when compared to the Wait phase results. CPU usage reveals that *dockerd* is using processing power in the Wait phase. However, the results from the baseline experiment show that *dockerd* barely uses CPU when managing only two containers. Similar behavior is observed for the other resources consumption results. In fact, *Dockerd* mean memory usage is 13% during the Wait phase, while in the baseline it is 0.86%. *Dockerd.* Furthermore, the virtual memory usage during the Wait phase is about four times higher than the baseline. Figure 7 presents the numerical comparison of both sets of results. The data in the table has the minimum, mean, and the maximum value for each considered resource.

The results in Figure 7 are specific to our experiment. Thus, in other environments or experiments, they may vary. However, these results are useful to highlight how *dockerd*

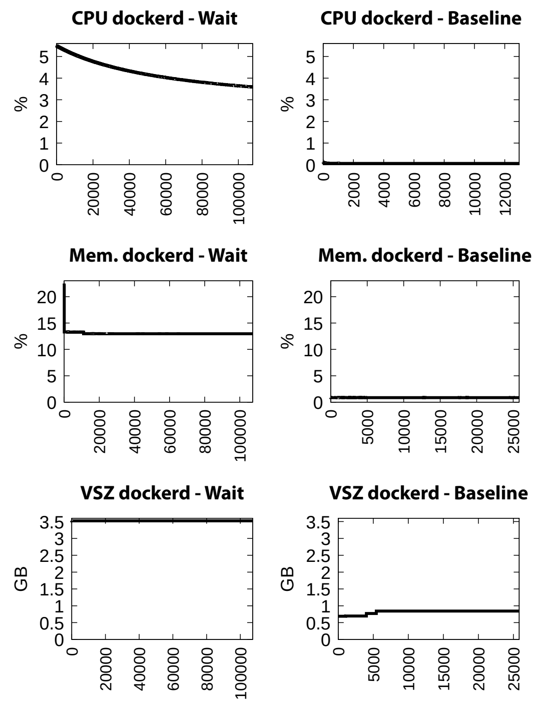
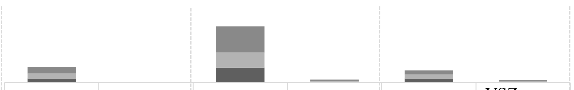


Fig. 6. Wait phase results x Baseline results



	CPU (%) Wait	CPU (%) Baseline	Mem. usage (%) Wait	Mem. usage (%) Baseline	VSZ usage (GB) Wait	VSZ usage (GB) Baseline
■ max	5.49	0.12	22.5	0.9	3.525	0.845
■ mean	4.24	0.06	13	0.86	3.524	0.817
■ min	3.59	0.06	12.96	0.85	3.524	0.689

Fig. 7. Wait x Baseline - Bar Chart and Table

software aging accumulation may lead the system to higher levels of resources depletion. The comparison results offer more confidence in the detection of software aging on the *dockerd* daemon.

B. Docker Failure Report

The accelerated workload during the Stress Phase leads *dockerd* to a *catastrophic* failure. As *catastrophic* we mean that we have to perform several operations in the disk and in the system to recover *dockerd* [16]. We notice that, after such failure, *dockerd* became unresponsive to user commands. Even simple commands like `docker ps`[2] did not receive *dockerd* response. Although the results in this paper are form a single experiment (for space reasons), we conducted two experiments. We perceive a catastrophic failure in the Stress Phase of both runs.

[2]Command to list running Docker containers.

Authorized licensed use limited to: b-on: Universidade de Coimbra. Downloaded on September 13,2023 at 23:00:11 UTC from IEEE Xplore. Restrictions apply.

Dockerd recovery was performed in four steps: i) deletion of the contents of `overlay2/` and `containers/` folders; ii) deletion of any remaining containers; iii) docker cleanup using `docker system prune` command, and iv) Operating System reboot. We realized this approach for *dockerd* recovery using try-and-error approach in our first experiment run.

We highlight that this recovery approach fits our purposes (i.e., software rejuvenation). As all the containers are stateless, and our environment is entirely dedicated to the experiment, we are comfortable to perform those administrative operations. Therefore, we do not assure that the same approach will work on other Docker implementations or configurations.

V. Threats to validity

There are two main threats to the validity of our results: 1) the observed resources depletion may be related to other software components than the *dockerd* daemon; and 2) the resources consumption persists in higher levels for other reasons than software aging. Below we discuss these threats.

Threat 1) To avoid possible side-effects from other software components we designed a stress workload directly related to the *dockerd* daemon. Besides that, during the experiment, the system is dedicated only to run this workload. However, as we notice in the Node memory usage results (Figure 5(c)), the system reaches almost 100% of memory usage, while *dockerd* is responsible for only about 20% of this amount. Thus, in future experiments, we will aim at expanding the resources monitoring for other software components.

Threat 2) The persistent degradation is one of the main characteristics of software aging. In the experiments, we notice this behavior in the *dockerd* daemon. The persistent degradation is removed after cleanup operations that comprise software restart, which is a common software rejuvenation approach.

VI. Conclusion and future works

This paper presented an experimental study of software aging and rejuvenation on the *dockerd* daemon, following the SWARE approach. *Dockerd* is at the core of the Docker architecture, thus needs to provide high levels of dependability to ensure the architecture manageability.

We highlight the following conclusions from our study. Our obtained results show **evidence of software aging in *dockerd* daemon**. *Dockerd* accumulates persistent degradation of resources after exposure to an accelerated workload of container instantiation and deletion. Besides that, *Dockerd* recovers from the degraded state after a cleanup and Operating System reboot.

We also presented a **comparison of the system state with and without software aging**. Our results show a significant difference between the resources consumption of *dockerd* with and without software aging accumulation. Finally, we experienced **a *catastrophic* failure on *dockerd*** during the accelerated workload submission and we presented a **workaround to the problem**.

In the future works, we intend to verify possible side effects of *dockerd* software aging in the running containers and to expand the observed software components during SWARE execution. Besides that, we also intend to investigate alternative approaches for *dockerd* rejuvenation. Use of software aging degration prediction techniques is also a future work.

VII. Acknowledgments

This work has been partially supported by project ATMOSPHERE, funded by the European Commission under the Cooperation Programme, Horizon 2020 grant agreement no 777154.

References

[1] A. Mouat, *Using Docker: Developing and deploying software with containers.* " O'Reilly Media, Inc.", 2015.

[2] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[3] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2016, pp. 44–51.

[4] D. Messina. (2018) 5 years later, where are you on your docker journey? - docker blog. [Online]. Available: https://blog.docker.com/2018/03/5-years-later-docker-journey/

[5] M. Torquato, P. Maciel, J. Araujo, and I. Umesh, "An approach to investigate aging symptoms and rejuvenation effectiveness on software systems," in *Information Systems and Technologies (CISTI), 2017 12th Iberian Conference on.* IEEE, 2017, pp. 1–6.

[6] M. Grottke and K. Trivedi, "A classification of software faults," *Journal of Reliability Engineering Association of Japan*, vol. 27, no. 7, pp. 425–438, 2005.

[7] K. Vaidyanathan and K. S. Trivedi, "Extended classification of software faults based on aging," in *Fast Abstract, Int. Symp. Software Reliability Eng., Hong Kong*, 2001.

[8] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on.* IEEE, 1995, pp. 381–390.

[9] M. Torquato, I. Umesh, and P. Maciel, "Models for availability and power consumption evaluation of a private cloud with vmm rejuvenation enabled by vm live migration," *The Journal of Supercomputing*, vol. 74, no. 9, pp. 4817–4841, 2018.

[10] M. Torquato and M. Vieira, "Interacting srn models for availability evaluation of vm migration as rejuvenation on a system under varying workload," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW).* IEEE, 2018, pp. 300–307.

[11] R. Matos, J. Araujo, V. Alves, and P. Maciel, "Characterization of software aging effects in elastic storage mechanisms for private clouds," in *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on*, Nov 2012, pp. 293–298.

[12] F. Machida, J. Xiang, K. Tadano, and Y. Maeno, "Combined server rejuvenation in a virtualized data center," in *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing.* IEEE, 2012, pp. 486–493.

[13] C. Melo, J. Araujo, V. Alves, and P. R. M. Maciel, "Investigation of software aging effects on the openstack cloud computing platform." *JSW*, vol. 12, no. 2, pp. 125–137, 2017.

[14] M. Torquato, J. Araujo, I. Umesh, and P. Maciel, "Sware: a methodology for software aging and rejuvenation experiments," *J Inf Syst Eng Manag*, vol. 3, no. 2, p. 15, 2018.

[15] J. Araujo, R. Matos, P. Maciel, and R. Matias, "Software aging issues on the eucalyptus cloud computing infrastructure," in *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on.* IEEE, 2011, pp. 1411–1416.

[16] P. Koopman, J. Sung, C. Dingman, D. Siewiorek, and T. Marz, "Comparing operating systems using robustness benchmarks," in *Proceedings of SRDS'97: 16th IEEE Symposium on Reliable Distributed Systems.* IEEE, 1997, pp. 72–79.

Authorized licensed use limited to: b-on: Universidade de Coimbra. Downloaded on September 13,2023 at 23:00:11 UTC from IEEE Xplore. Restrictions apply.

Appendix D: PyMTDEvaluator Output Report

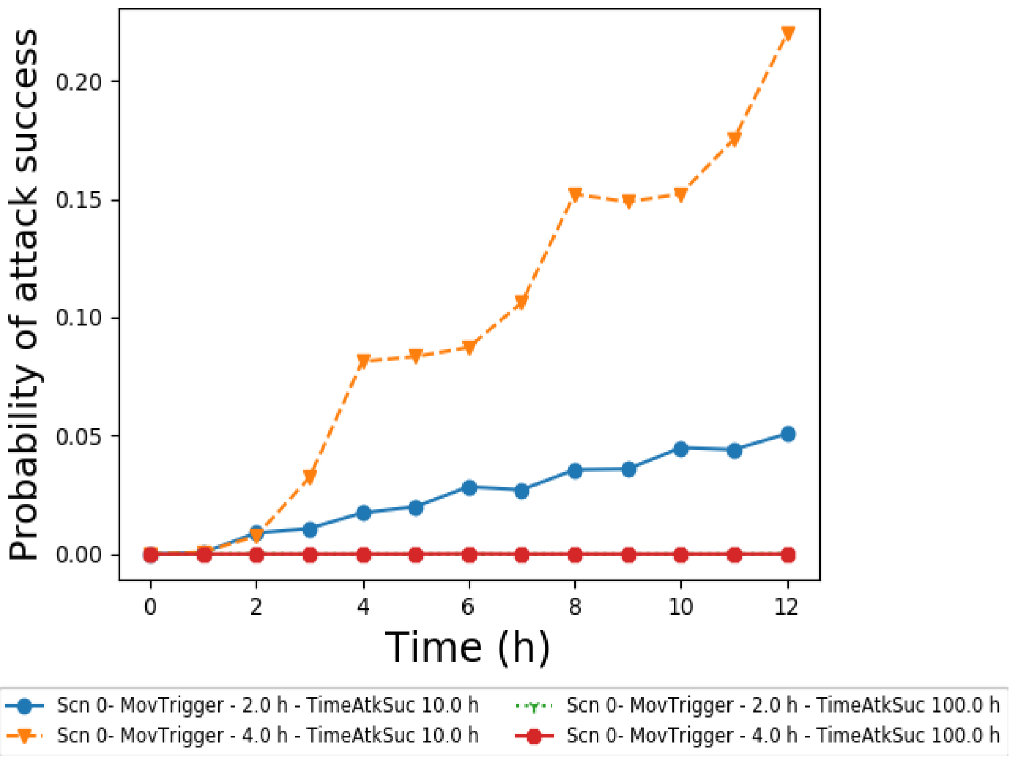
In this appendix, we present an example of the *PyMTDEvaluator* output report.

```
PyMTDEvaluator Report

Fri Jul 1 08:53:14 2022

PyMTDEvaluator - Summary of Results
+++++++++++++++++++++++++
Scenario 0


Parameters
----------
Movement Trigger = 2.0 h
Time for Attack Success = 10.0 h
-------
Results
--------
Expected downtime due to movements (evaluation time) = 0.0 min
Expected annual downtime due to movements = 1.8 min
Expected total cost (evaluation time) = $ 5.4
Expected Threshold = 12 h
System Capacity 95% CI (evaluation time) = [82.64, 85.81, 88.98] %
System Availability 95% CI (evaluation time) = [0.95246, 0.96626, 0.98006]
Downtime 95% CI (evaluation time) = [10482.48, 17735.12, 24987.75] min
----------
Example run results
----------
Survival Time (evaluation time) = 12 h
System Capacity 95% CI -while available- (evaluation time) = [90.77, 94.23, 97.69]
Downtime (evaluation time) = 0.0min
+++++++++++++++++++++++++

Parameters
----------
Movement Trigger = 4.0 h
Time for Attack Success = 10.0 h
-------
Results
--------
Expected downtime due to movements (evaluation time) = 0.0 min
Expected annual downtime due to movements = 0.85 min
Expected total cost (evaluation time) = $ 2.7
Expected Threshold = 12 h
System Capacity 95% CI (evaluation time) = [65.74, 73.48, 81.22] %
System Availability 95% CI (evaluation time) = [0.8166, 0.87371, 0.93082]
Downtime 95% CI (evaluation time) = [36363.56, 66379.63, 96395.69] min
----------
Example run results
----------
Survival Time (evaluation time) = 12 h
System Capacity 95% CI -while available- (evaluation time) = [68.8, 78.57, 88.34]
Downtime (evaluation time) = 0.0min
+++++++++++++++++++++++++

Parameters
----------
Movement Trigger = 2.0 h
Time for Attack Success = 100.0 h
```

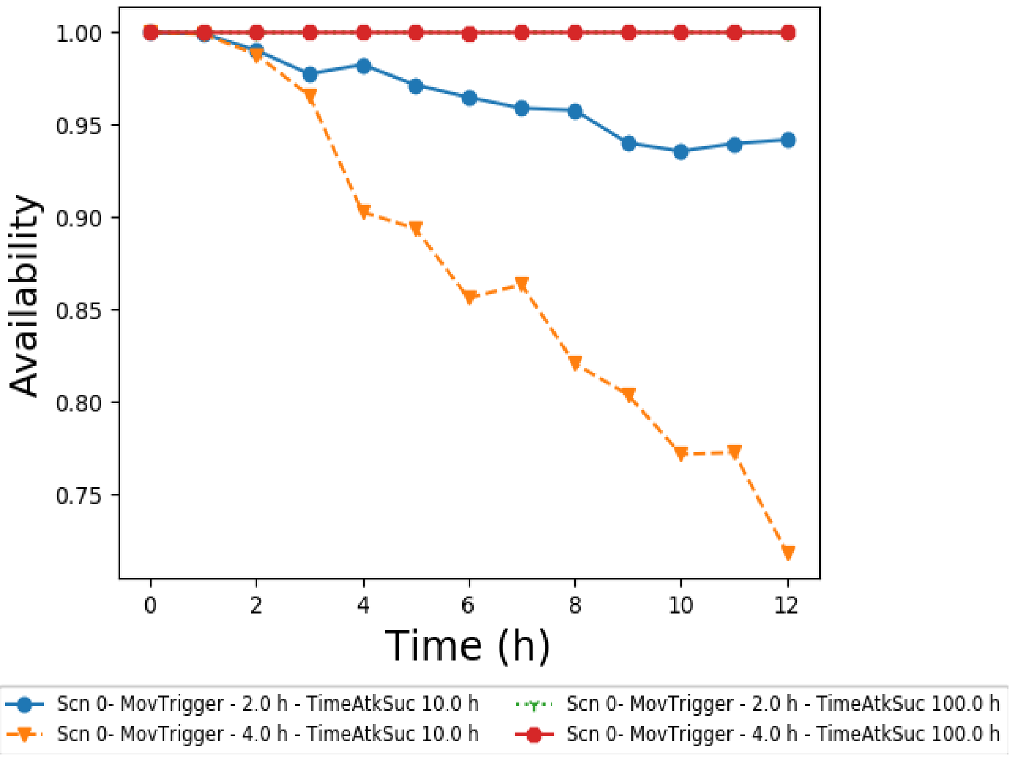
```
-------
Results
--------
Expected downtime due to movements (evaluation time) = 0.0 min
Expected annual downtime due to movements = 1.85 min
Expected total cost (evaluation time) = $ 5.4
Expected Threshold = 2 h
System Capacity 95% CI (evaluation time) = [98.01, 98.44, 98.88] %
System Availability 95% CI (evaluation time) = [0.99996, 0.99998, 1.00001]
Downtime 95% CI (evaluation time) = [-2.73, 9.12, 20.97] min
----------
Example run results
----------
Survival Time (evaluation time) = 12 h
System Capacity 95% CI -while available- (evaluation time) = [100.0, 100.0, 100.0]
Downtime (evaluation time) = 0.0min
++++++++++++++++++++++++++

Parameters
----------
Movement Trigger = 4.0 h
Time for Attack Success = 100.0 h
-------
Results
--------
Expected downtime due to movements (evaluation time) = 0.0 min
Expected annual downtime due to movements = 0.86 min
Expected total cost (evaluation time) = $ 1.8
Expected Threshold = 6 h
System Capacity 95% CI (evaluation time) = [96.56, 97.65, 98.75] %
System Availability 95% CI (evaluation time) = [0.99997, 0.99999, 1.00001]
Downtime 95% CI (evaluation time) = [-4.41, 4.42, 13.25] min
----------
Example run results
----------
Survival Time (evaluation time) = 12 h
System Capacity 95% CI -while available- (evaluation time) = [77.7, 85.0, 92.3]
Downtime (evaluation time) = 0.0min
++++++++++++++++++++++++++
```

Scenario 1

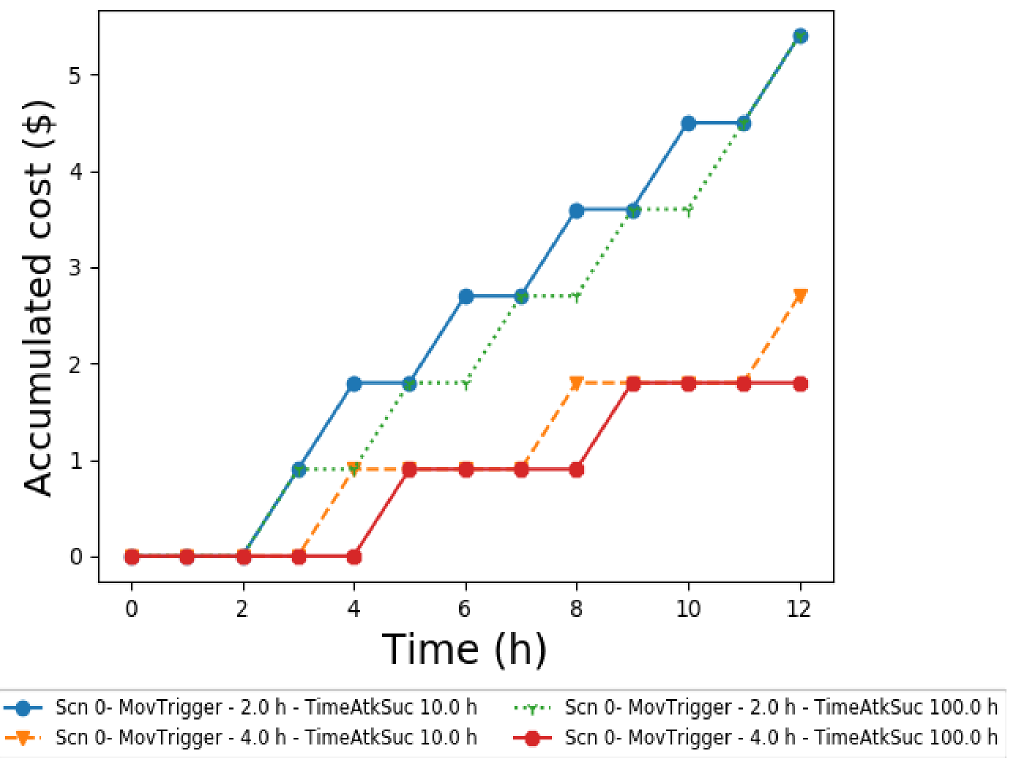
Probability of attack success



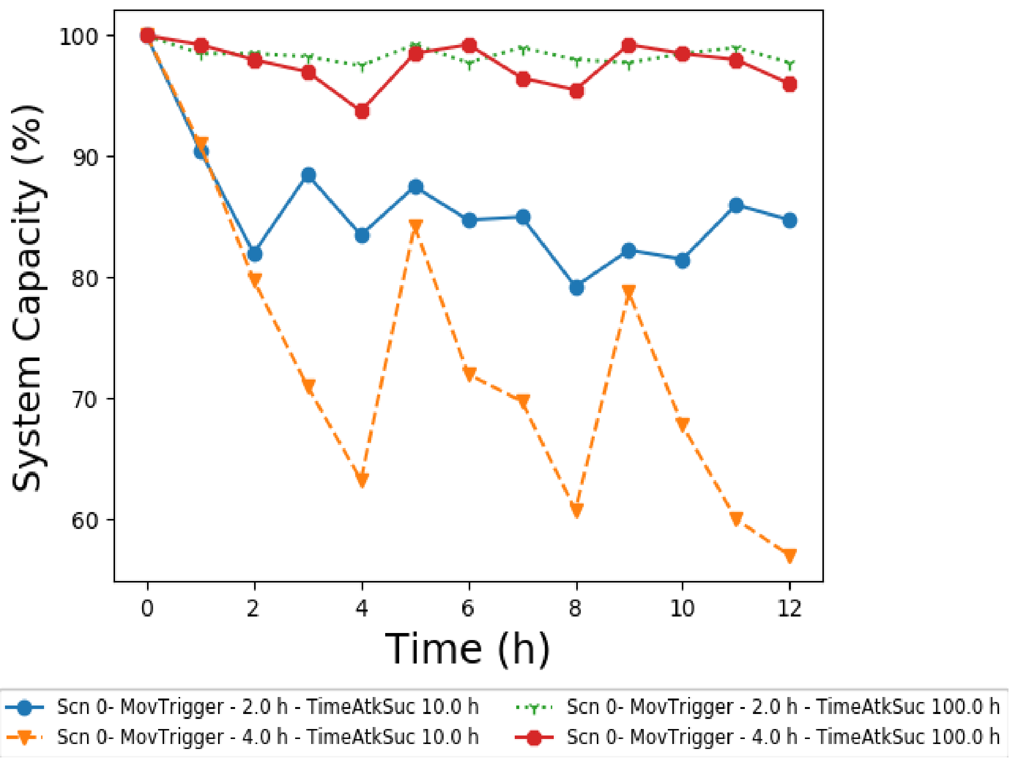
Availability



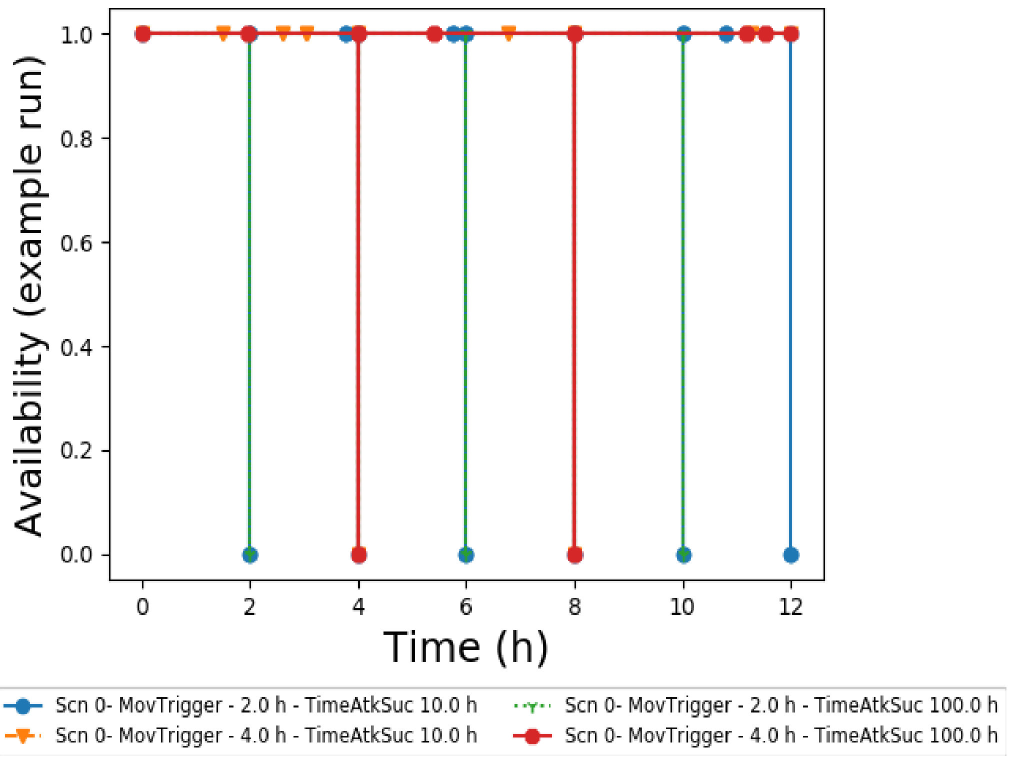
Accumulated cost



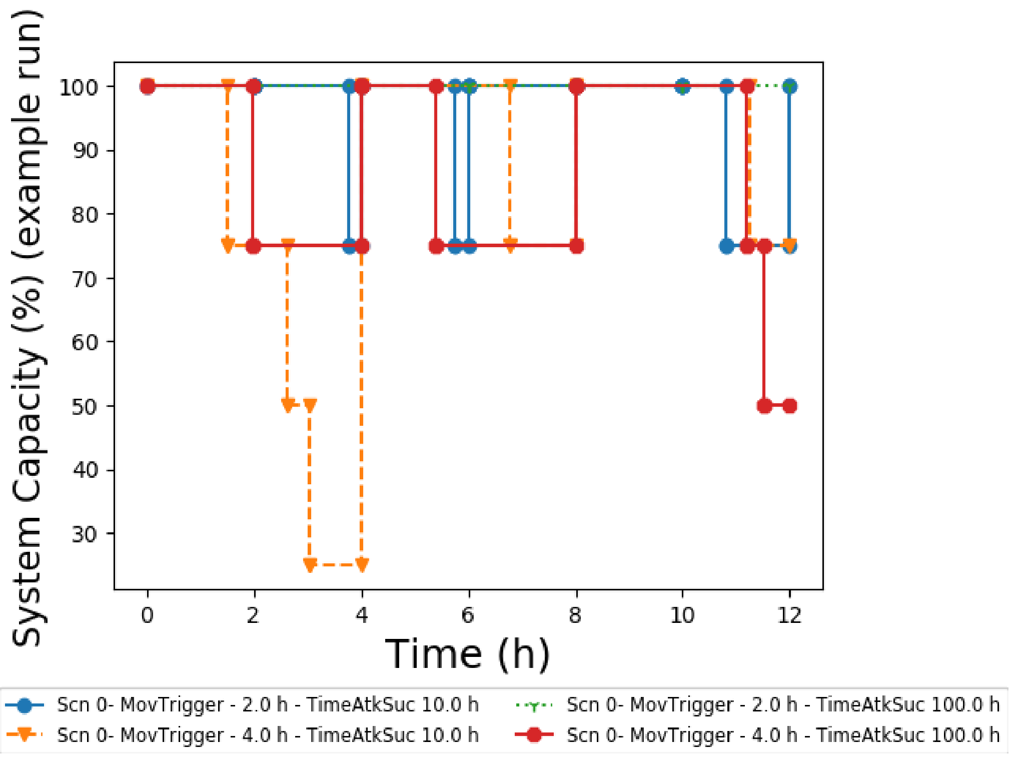
Capacity



Availability (example run)



Capacity (example run)


System Capacity (%) (example run)
Time (h)
Scn 0- MovTrigger - 2.0 h - TimeAtkSuc 10.0 h
Scn 0- MovTrigger - 2.0 h - TimeAtkSuc 100.0 h
Scn 0- MovTrigger - 4.0 h - TimeAtkSuc 10.0 h
Scn 0- MovTrigger - 4.0 h - TimeAtkSuc 100.0 h