Migração Automatizada de VMs na Defesa de *Brokers MQTT* Contra *Memory Denial of Service*

Matheus Torquato[1,2], Charles F. Gonçalves[2,5], Michele Nogueira[3], Denis Rosário[4], Eduardo Cerqueira[4]

[1]Instituto Federal de Alagoas (IFAL) – Arapiraca – AL – Brasil
[2]Universidade de Coimbra, CISUC/LASI, DEI – Coimbra, Portugal
[3]Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte – MG – Brasil
[4]Universidade Federal do Pará (UFPA) – Belém – PA – Brasil
[5]Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Belo Horizonte – MG – Brasil

matheus.torquato@ifal.edu.br, charles@dei.uc.pt
michele@dcc.ufmg.br, {denis, cerqueira}@ufpa.br

Abstract. *One of the main protocols of the Internet of Things (IoT) is the Message Queuing Telemetry Transport (MQTT). The MQTT protocol enables communication between IoT devices through brokers, which act as central points in the network topology. Brokers are frequent targets of severe attacks, such as denial of service (DoS) attacks, which can lead to the unavailability of important services, compromising critical applications such as health monitoring and precision agriculture. One of the main challenges in this context lies in the limitations of standard defense mechanisms. These mechanisms, such as firewalls, typically apply static configurations and, therefore, cannot respond to dynamic attacks. This article addresses this problem by proposing a solution based on the automated migration of virtual machines (VM) as a defense for brokers. The proposed algorithm allows multiple configurations and uses performance evaluation results as parameters for decision making. The results demonstrate a mitigation effect of over 75% in the best scenarios. The technique and code are open source and available for research reproduction.*

Resumo. *Um dos principais protocolos da Internet das Coisas (IoT) é o Message Queuing Telemetry Transport (MQTT). O protocolo MQTT permite a comunicação entre dispositivos IoT por meio de brokers, que atuam como pontos centrais na topologia da rede. Os brokers são alvos frequentes de ataques severos, como os de negação de serviço (DoS). Esses ataques podem levar à indisponibilidade de serviços importantes, comprometendo aplicações críticas, como monitoramento de saúde e agricultura de precisão. Um dos principais desafios nesse contexto está nas limitações dos mecanismos de defesa padrão. Esses mecanismos, como firewalls, geralmente aplicam configurações estáticas e, por isso, não conseguem responder a ataques dinâmicos. Este artigo aborda esse problema ao propor uma solução baseada na migração automatizada de máquinas virtuais como defesa para brokers. O algoritmo proposto permite múltiplas configurações e utiliza resultados de avaliação de desempenho como parâmetros para a tomada de decisão. Os resultados demonstram um efeito de mitigação superior a 75% nos melhores cenários. A técnica e o código estão disponíveis para a reprodução de pesquisas.*

1. Introdução

Um dos principais alvos de ataques em ambientes de Intenet das coisas (do inglês, *Internet of Things* - IoT) é o *broker* do protocolo *Message Queuing Telemetry Transport* (MQTT). De fato, uma varredura utilizando a plataforma Shodan[1] revela que o serviço possui adesão global. Isto ocorre por conta do MQTT ser amplamente utilizado em virtude da sua eficácia na comunicação entre computação em nuvem e IoT [Li et al. 2024]. O protocolo MQTT é utilizado para comunicação entre dispositivos através do *broker MQTT*. Por ser central nas operações de sistemas IoT, o *broker MQTT* é alvo de atacantes visando comprometer a disponibilidade dos serviços oferecidos. Em geral, o *broker MQTT* é implantado em ambientes virtualizados. No entanto, o processo de virtualização introduz novas vulnerabilidades ao ecossistema IoT. Uma das abordagens conhecidas pelos atacantes para produzir uma negação de serviços é o *Memory Denial of Service* (MemDoS) [Islam et al. 2023].

O ataque MemDoS parte de uma máquina virtual (VM) e visa exaurir os recursos da máquina física (PM) que a hospeda. O intuito desse ataque é afetar a PM hospedeira e (principalmente) as outras VMs co-residentes. Portanto, a execução e efeitos do MemDoS estão limitados à PM que hospeda a VM atacante (i.e., é um tipo de ataque baseado em host - *host-based attack*). Em síntese, o mecanismo do ataque MemDoS consiste em executar instruções de `LOCK` na memória principal para torná-la indisponível para outros clientes, tal como descrito em [Zhang et al. 2017]. Portanto, a migração de VMs é promissora como defesa, uma vez que remove a vítima do ambiente compartilhado com o atacante.

Um trabalho anterior tratou um problema similar [Torquato et al. 2024] usando a técnica *execution throttling* (ET), que apenas reduz a intensidade do ataque, mas ainda mantém a vítima na mesma PM da VM do atacante. Além disso, a ET depende fortemente da detecção de intrusão, ou seja, uma falha na detecção acarreta o seu uso indevido causando problemas em VMs benignas. Outro trabalho [Torquato and Vieira 2021] utiliza a técnica de migração como defesa contra MemDoS. Nele a migração possui agendamento fixo (i.e., independente do estado do sistema). Além disso, o trabalho observa impactos em aplicações fora do contexto de IoT. Assim, a proposta deste artigo avança os trabalhos anteriores em dois pontos principais: 1) provê migrações sensíveis ao contexto (i.e., baseadas no tempo de resposta da aplicação), evitando assim migrações desnecessárias e 2) oferece uma proteção superior para vítima do ataque ao removê-la do ambiente compartilhado com o atacante.

A pesquisa apresentada neste artigo toma como base as técnicas de detecção de anomalias e os conceitos de *Moving Target Defense* (MTD). A pergunta de pesquisa endereçada neste artigo é: *"Qual o impacto de diferentes configurações de migração automatizada de VMs na defesa do broker MQTT contra MemDoS?"*. Para respondê-la, este artigo utiliza um algoritmo para automação da migração de VMs e um conjunto de estudos de caso para avaliá-lo em um sistema submetido ao ataque MemDoS. No melhor do nosso conhecimento, este é o primeiro artigo a exercitar a migração de VM como defesa de *broker MQTT* virtualizado. Assim, este artigo apresenta as seguintes contribuições

[1] https://www.shodan.io/search/report?query=mqtt - No momento da escrita desse artigo, os números do MQTT eram mais que o dobro dos CoAP (i.e., outro protocolo competidor). MQTT = 507.059 registros e CoAP = 220.887 registros.

principais: (i) a proposta de um algoritmo para automação da migração de VMs; (ii) um conjunto estruturado de experimentos que permitem a sua reprodução considerando diferentes cenários; e (iii) código aberto.

O *testbed* utilizado consiste em um ambiente virtualizado com suporte à migração de VMs KVM[2]. O ambiente MQTT utilizado roda o protocolo Mosquitto[3]. Para os resultados, observou-se majoritariamente o tempo de resposta do *broker* MQTT. Os resultados mostram que há efeitos residuais observados após a migração da VM. Ou seja, mesmo após aplicar a defesa, na ausência do atacante, a vítima reteve algum nível de degradação do serviço. Essa degradação aparenta estar relacionada com o tempo que a vítima passou sob ataque. Além disso, os resultados indicam melhorias acima de 70% no desempenho do sistema em comparação com cenários onde não há defesa implementada.

Este trabalho está organizado como segue. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 apresenta o algoritmo utilizado para automatizar a migração das VMs. A Seção 4 detalha os resultados e discussão dos experimentos realizados. Por fim, a Seção 5 conclui o trabalho e apresenta os trabalhos futuros.

2. Trabalhos relacionados

O artigo [Torquato et al. 2024] apresenta resultados de desempenho de um *broker MQTT* vítima de MemDoS. O trabalho aplica a técnica ET para para mitigação do MemDoS em vez de migração de VMs. ET é capaz de reduzir a intensidade do MemDoS sem a necessidade de uma PM adicional para migração. Porém, a migração supera a proteção conferida por *ET* uma vez que leva a vítima para um ambiente sem o atacante. Além disso, diferente do trabalho citado, o atual artigo se preocupa em identificação de resíduos do ataque após a migração. Com essa identificação é possível destacar a necessidade de mecanismos adicionais para trazer o sistema de volta para um estado estável.

Apesar da literatura atual indicar a migração de VMs como um dos principais mecanismos MTD para computação em nuvem [Torquato and Vieira 2020], ainda há escassez de trabalhos que explorem sua aplicação no contexto de brokers MQTT. Uma iniciativa relevante é apresentada em [Kusumi and Koide 2024], que aplica MTD ao protocolo MQTT substituindo o *TLS (Transport Security Layer)* por uma estratégia de defesa baseada em MTD. Outro trabalho de destaque é o de [Siddharthan et al. 2022], que utiliza algoritmos de aprendizado de máquina para detecção de intrusão em ambientes MQTT. Diferentemente dessas abordagens, este artigo propõe uma técnica personalizável para disparo de ações de defesa no ambiente. O principal objetivo da técnica é oferecer flexibilidade suficiente para atender a diferentes requisitos de desempenho da aplicação.

A revisão da literatura em [Saputro et al. 2020] evidencia a crescente importância da MTD para mitigar ataques em IoT. No entanto, a maioria dos trabalhos se concentra em soluções baseadas em Redes Definidas por Software (SDN), como a utilização de controladores SDN para isolar dispositivos comprometidos. Neste artigo, propomos uma abordagem complementar, explorando a migração de VMs como uma técnica para dificultar a persistência de ataques em ambientes IoT virtualizados. Ao migrar dinamicamente as VMs que hospedam os serviços IoT, torna-se mais difícil para um atacante manter o

[2]Kernel Virtual Machine
[3]https://mosquitto.org/

progresso do ataque sobre um sistema comprometido, aumentando significativamente a resiliência da infraestrutura.

Assim, a investigação dos impactos de utilizar migração de VMs como defesa de *brokers* MQTT torna-se necessária para destacar vantagens e desvantagens da técnica. A literatura atual carece de evidências para ajudar na quantificação desse impacto. Este trabalho ataca esse problema não apenas ao apresentar resultados experimentais, mas também ao fornecer os meios para reprodução do estudo realizado.

3. Automação da migração de VMs

Esta seção detalha o algoritmo de automação de migração de VMs proposto. O algoritmo decide sobre o disparo automatizado de migrações tomando como base os princípios básicos da detecção de anomalias (i.e., verificar discrepâncias entre o estado observado no sistema e um estado conhecido previamente e tido como normal), que são aplicados largamente no contexto de redes de computadores [Fernandes et al. 2019]. Um dos objetivos principais é verificar se o tempo de resposta do *broker* MQTT está dentro dos valores aceitáveis para aplicações IoT, por exemplo, abaixo de 20 ms. Esses limites aceitáveis são definidos depois de observar o broker MQTT num ambiente IoT controlado (i.e., sem efeitos de ataques ou interferências externas).

O algoritmo proposto permite personalização desses limites aceitáveis. Alguns serviços da IoT têm maior ou menor tolerância a oscilações no tempo de resposta. Por exemplo, sensores de monitoramento de saúde precisam ter rápida resposta, enquanto dispositivos que medem a temperatura de um ambiente agrícola podem tolerar tempo de resposta mais longo. Desse modo, o algoritmo proposto neste artigo possibilita alterações de parâmetros conforme a necessidade da aplicação IoT. A anomalia (i.e., comportamento diferente do esperado) considerada para o disparo da migração advém da observação do tempo de resposta do *broker* MQTT. Em síntese, para detectar anomalias, o algoritmo compara o tempo de resposta observado (DO) no ambiente com o tempo de resposta *ideal* (DI). Por *ideal*, entenda-se o tempo de resposta da aplicação sob condições normais (i.e., com a ausência de anomalias). Para observar DI é necessário conduzir experimentos em um ambiente controlado e sem interferências.

Uma abordagem para a descoberta do DI está presente na Seção 4.2. O algoritmo proposto decide com base no tempo de resposta médio de mensagem, que representa o tempo médio necessário para o *broker* MQTT responder a uma solicitação. Essa métrica foi escolhida por ser altamente sensível à sobrecarga do sistema, característica comum em ataques MemDoS [Torquato and Vieira 2021][Torquato et al. 2021]. A detecção de anomalia é embasada pelo grau de degradação relativo ao DI. Estima-se o **tempo médio de mensagem** (i.e., tempo médio de resposta dos tratamentos de mensagem) (μ_{DI}) e o seu desvio padrão (σ_{DI}) em condições normais de funcionamento das aplicações IoT (i.e., livre de ataques). O desvio padrão indica a variabilidade natural no tempo de resposta. Subentende-se que o perfil do desempenho da aplicação sob condições normais é conhecido, de modo que a detecção de anomalia será sensível à variação configurada, independente de sua origem. A descoberta da origem da anomalia detectada está fora do escopo desse artigo.

O *Bucket Algorithm (BA)* utiliza os parâmetros do DI como referência (μ_{DI} e σ_{DI}) para comparar continuamente com o tempo de resposta corrente observado no sis-

tema $\hat{x}$ e estabelecer o grau de degradação atual do sistema com base no teorema do limite central [Gonçalves et al. 2023]. Considere que *BA* gerencia uma quantidade B de reservatórios, cada um com a capacidade máxima de D fichas. A cada amostra $\hat{x}$ avaliada, uma ficha pode ser armazenada ou removida dos reservatórios. BA utiliza um ponteiro e um contador para gerenciar os reservatórios: b é um ponteiro que indica qual reservatório está sendo utilizado, enquanto d é um contador que representa o número de fichas presentes no reservatório corrente. A Figura 1 ilustra o funcionamento do BA.

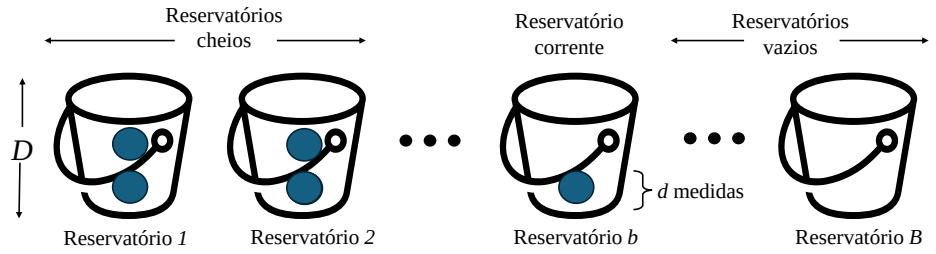


Figura 1. *Bucket Algorithm* - exemplo com $D = 2$.

O fluxo de preenchimento ou esvaziamento dos reservatórios em *BA* funciona da seguinte maneira. *BA* estima o grau de degradação atual do sistema ($\tilde{x}$) pela seguinte fórmula $\tilde{x} = \mu_{DI} + (b-1)\sigma_{DI}$. Assim, o reservatório corrente recebe uma ficha quando $\hat{x} \geq \tilde{x}$ (tempo de tratamento de mensagem atual maior que o grau de degradação do sistema). De modo oposto, o reservatório corrente remove uma ficha quando $\hat{x} < \tilde{x}$. Neste fluxo, a dinâmica dos reservatórios sendo preenchidos e esvaziados a depender da medida observada contempla variações esperadas na carga do sistema. Quando o reservatório corrente estiver completamente preenchido (i.e. $d = D$), utiliza-se o próximo reservatório, (aumentando o nível de degradação atual em mais um desvio padrão). Quando todos os reservatórios estiverem cheios, o alarme é disparado. Por outro lado, se um reservatório for esvaziado (i.e., $d < 0$), retorna-se para o reservatório anterior. A Figura 2 apresenta um fluxograma de *BA*.

Em síntese, na inicialização todos os reservatórios estão vazios e os ponteiros estão em suas posições iniciais. O laço principal de execução obedece o seguinte fluxo.

1. Se o a medida observada ($\hat{x}$) superar o nível atual de degradação alvo ($\tilde{x}$), o reservatório recebe uma ficha. Caso contrário, uma ficha é retirada do reservatório;
2. Se for necessário armazenar uma ficha num reservatório já preenchido completamente, utiliza-se o próximo reservatório, aumentando o nível de degradação alvo por mais um desvio padrão (σ_{DI});
3. Se for necessário retirar uma ficha de um reservatório já vazio, retorna-se ao reservatório anterior;
4. Se for necessário retirar uma ficha e o reservatório já for o inicial, mantém-se o ponteiro *d* na posição 0 do reservatório 1 (inicial);
5. Se for necessário armazenar uma medida e todos os reservatórios estiverem cheios, dispara-se o alarme (i.e., anomalia detectada).

O fluxo de execução do *BA* considera uma avaliação sequencial de desempenho da aplicação observada. A configuração dos reservatórios em um sistema de detecção

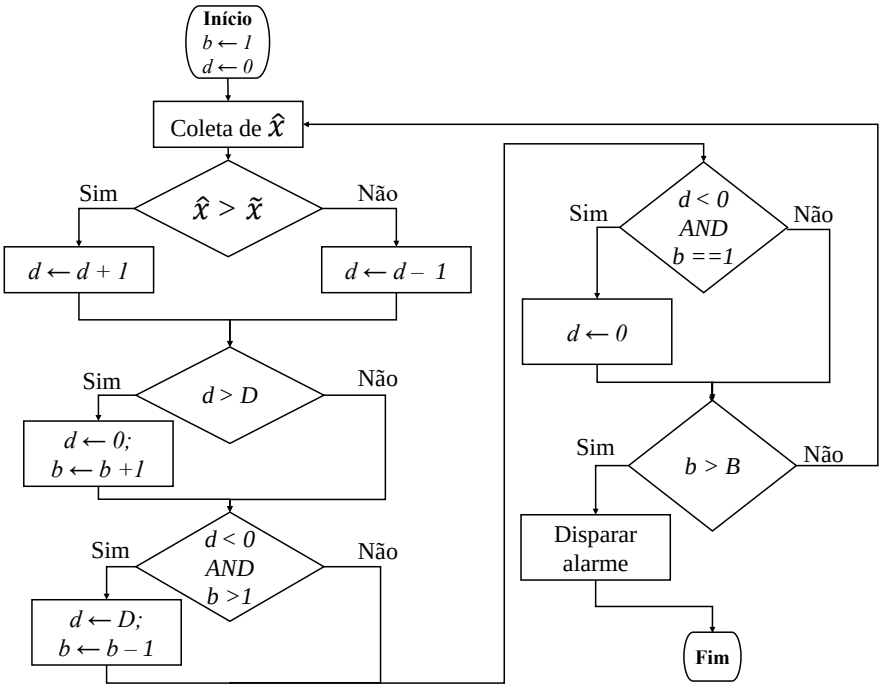


Figura 2. *Bucket Algorithm* - Fluxograma

de anomalias envolve um trade-off crucial entre robustez e sensibilidade. Os reservatórios maiores e mais numerosos conferem maior robustez contra sobre cargas transientes, mas atrasam a detecção de ataques (tempo mínimo de detecção é o tempo de processamento de $B \times D$ mensagens). Por outro lado, os reservatórios menores e menos numerosos aumentam a sensibilidade, mas aumentam a chance de falsos positivos. Em [Gonçalves et al. 2023] há uma discussão detalhada a respeito dos compromissos envolvidos na parametrização do BA. No contexto deste artigo, BA foi utilizado no contexto de defesa de *brokers MQTT*, no entanto, salienta-se que o algoritmo pode ser adaptado/utilizado em outros contextos, uma vez que considera métricas genéricas de desempenho.

4. Avaliação

Esta seção apresenta a metodologia de avaliação e os resultados de três experimentos. O primeiro experimento (Subseção 4.1) consiste no planejamento de capacidade. O estudo apresentado busca encontrar a carga de trabalho adequada para o *broker* MQTT hospedado no ambiente. O segundo experimento (Subseção 4.2) é para obtenção do *desempenho ideal* (DI). Nele, a experimentação consiste em estressar o ambiente com a carga de trabalho obtida no experimento anterior para observar o desempenho do *broker* MQTT na ausência de anomalias. O terceiro experimento (Subseção 4.3) exercita o *BA* como mecanismo de disparo de migração automatizada de VMs como mecanismo de defesa. A Figura 3 ilustra a arquitetura do sistema. Ela é composta por três PMs denominadas STRESSER, MÁQUINA FONTE e MÁQUINA ALVO, além de duas VMs: VM VÍTIMA e VM ATACANTE. A MÁQUINA FONTE (host) hospeda inicialmente todas as VMs. A VM VÍTIMA executa o *broker* MQTT, e a VM ATACANTE executa o ataque MemDoS. A MÁQUINA ALVO é o destino para a migração da VM VÍTIMA. O STRESSER

gera carga de trabalho para o broker MQTT. A Tabela 1 detalha as configurações.

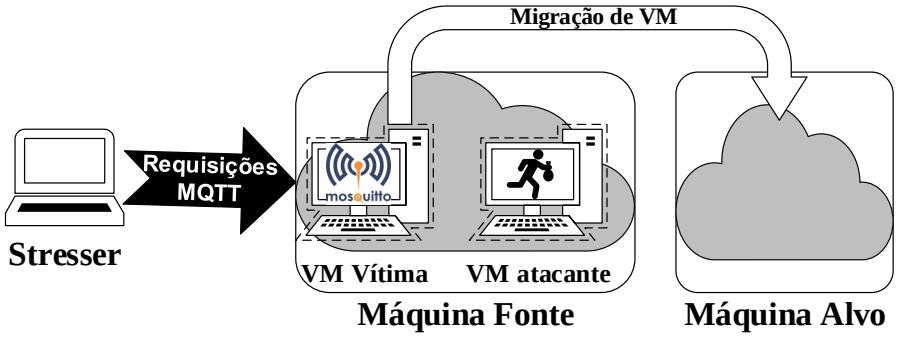


Figura 3. Arquitetura do sistema

Tabela 1. Configuração dos componentes da arquitetura

Componente	Hardware	Software
STRESSER	Intel Core i5 1035G1 1.0GHz, 8GB RAM	Fedora Linux 39 (kernel 6.11.9), *mqtt-benchmark*
MÁQUINA FONTE	Intel Core Xeon E5-2620 2.0GHz, 16GB RAM	Ubuntu Server 20.04 (kernel 5.4.0), Kernel Virtual Machine (KVM) 4.2.1
MÁQUINA ALVO	Intel Core i7-9700 3.0GHz, 16GB RAM	Ubuntu Server 20.04 (kernel 5.4.0), Kernel Virtual Machine (KVM) 4.2.1
VM VÍTIMA	Single-core 1.0GHz, 3GB RAM	Ubuntu Server 20.04 (kernel 5.4.0), Mosquitto MQTT Broker [Light 2017] versão 1.6.9. A versão do MQTT é 3.1.1.
VM ATACANTE	Single-core 1.0GHz, 3GB RAM	Ubuntu Server 20.04 (kernel 5.4.0), *Memory Denial of Service* (MemDoS)

O atacante, controlando a VM ATACANTE, executa um ataque MemDoS com o objetivo de sobrecarregar a memória da MÁQUINA FONTE, onde a VM VÍTIMA está hospedada. Para mitigar o ataque, é realizada a migração da VM VÍTIMA para a MÁQUINA ALVO. Essa ação remove a VM VÍTIMA do alcance direto do atacante, que está confinado à MÁQUINA FONTE. Um dos principais problemas para a implantação do mecanismo de defesa é o agendamento da migração. No cenário com migração **oportuna**, a migração ocorre antes dos efeitos do ataque causarem uma negação no serviço do *broker MQTT*. No cenário com migração **atrasada**, a migração torna-se inócua pois o atacante atingiu o seu objetivo. Desse modo, o mecanismo decide o momento de disparo da migração da VM para proteger o sistema. A Figura 4 apresenta um exemplo de fluxo com as duas situações: migração **oportuna** e migração **atrasada**. MF significa MÁQUINA FONTE e MA, MÁQUINA ALVO.

4.1. Planejamento de capacidade

O experimento de planejamento de capacidade consiste no envio crescente de carga de trabalho a partir da máquina STRESSER para o *broker* MQTT hospedado na VM VÍTIMA. Desse modo, a máquina STRESSER executa o *mqtt-benchmark* variando o tamanho da mensagem para cada rodada de experimentos. De modo arbitrário, o experimento considera um total de 100 clientes com 100 requisições cada um. O intuito é observar em qual

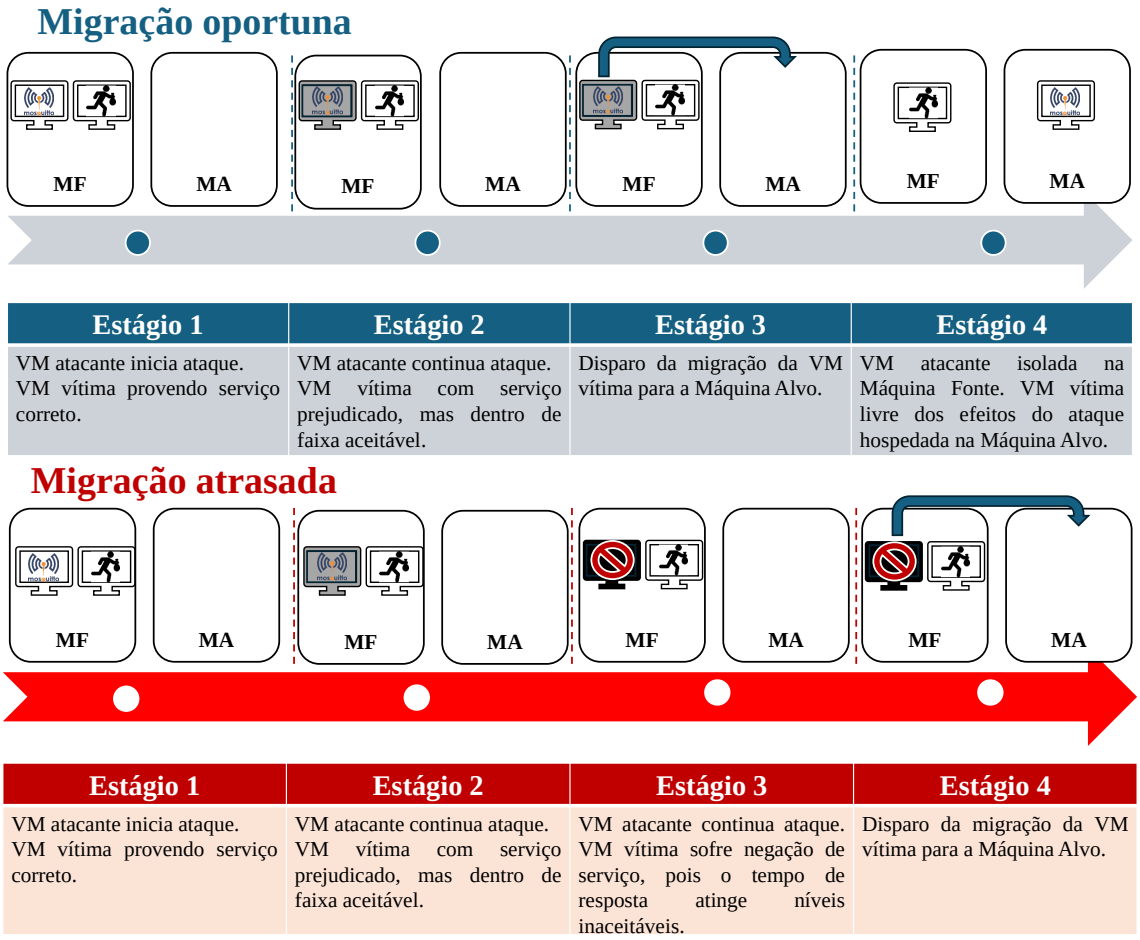


Figura 4. Cenário de ataque e defesa

configuração a qualidade do serviço baixa drasticamente, indicando um ponto limite de capacidade.

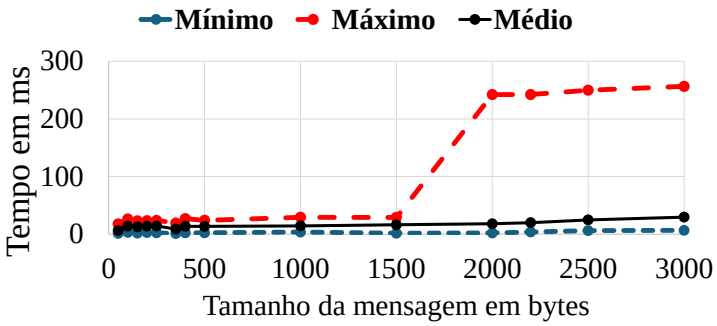


Figura 5. Resultados do experimento de planejamento de capacidade

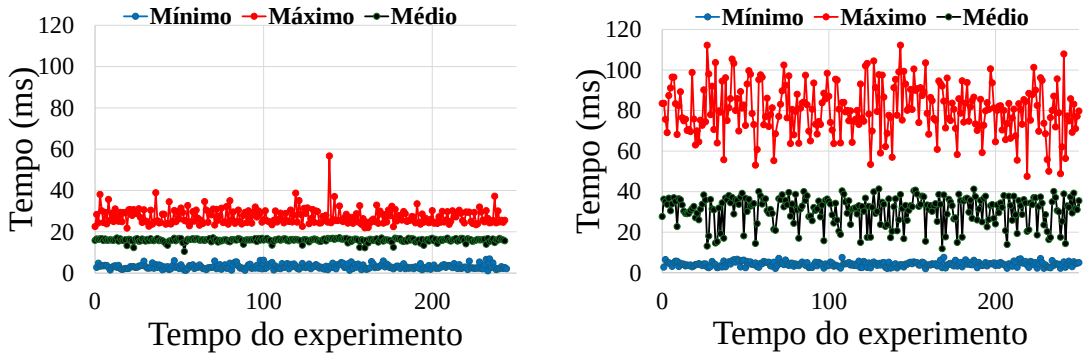
A Figura 5 mostra os resultados obtidos. Apesar dos tempos médios e mínimos permanecerem estáveis com o incremento do tamanho da mensagem, o tempo máximo apresenta um aumento substancial. Ao alterar o tamanho da mensagem de 1500 para 2000 bytes, o tempo máximo aumenta de 29 ms para 242 ms (mais de oito vezes maior). Este resultado indica que a carga de mensagens com mais de 2000 bytes de tamanho superam a capacidade do sistema em questão. Por este motivo, a carga com mensagens

de tamanho **1500 bytes** foi escolhida para as experimentações das próximas seções.

4.2. Experimentos para descoberta do *desempenho ideal*

Definida a carga de trabalho apropriada para o *broker* MQTT (vide experimento anterior), procede-se à avaliação do desempenho do sistema em um ambiente controlado, submetido a essa carga. O experimento apresentado nessa seção utiliza os parâmetros obtidos do planejamento de capacidade apresentado anteriormente. Desse modo, o *broker MQTT* recebe uma carga constante de 100 mensagens de 100 clientes com cada uma com o tamanho de 1500 bytes. As VMs estão hospedadas na MÁQUINA FONTE.

A Figura 6a apresenta o resultado do experimento para descoberta do desempenho ideal. É possível notar que o tempo médio das mensagens é controlado com valores próximos de **16 ms** com o desvio padrão médio de **1 ms**. Desse modo, os parâmetros utilizados nos experimentos de ataque-defesa são os seguintes: $\mu_{DI} = 16$ e $\sigma_{DI} = 1$. Com o intuito de comparação, e para entender melhor os efeitos do ataque em questão, uma rodada adicional de experimentos executa o ambiente sob influência do ataque, porém sem defesas (*sistema indefeso*). A Figura 6b apresenta o resultado do experimento e a Tabela 2 apresenta a comparação entre o desempenho ideal e o desempenho do sistema indefeso sob ataque.



(a) Desempenho ideal

(b) Sistema indefeso

Figura 6. Resultado dos experimentos

Tabela 2. Desempenho ideal vs. desempenho do sistema indefeso

Métrica	Desempenho ideal	Desempenho Sistema Indefeso	Comparação
Mínimo	3 ms	4 ms	1 ms (33%)
Máximo	27 ms	79 ms	52 ms (193%)
Médio	16 ms	30 ms	14 ms (87%)
Desvio padrão (médio)	1 ms	6 ms	5 ms (500%)

A Tabela 2 indica que o impacto do ataque pode superar 80% de degradação na qualidade do serviço. Nos casos extremos (valores máximos), a degradação ultrapassa 190%. Além disso, o desempenho do sistema sob ataque revela que há instabilidade nos resultados observados onde o desvio padrão chega a ser 500% maior que o desvio padrão do desempenho ideal. Essa análise enfatiza a necessidade de aplicar mecanismos de defesa robustos para assegurar níveis de qualidade de serviço adequados.

4.3. Experimentos de ataque-defesa

O experimento de ataque-defesa consiste em submeter o *broker MQTT* ao MemDoS enquanto ele executa *BA* para disparar as migrações. Desse modo, o experimento a seguir exercita diversas configurações do algoritmo, visando entender quais os impactos de escolher diferentes parâmetros. Baseando-se no trabalhos anteriores [Torquato et al. 2021][Distefano et al. 2020] que sugerem o progresso de ataque com quatro etapas, o experimento considera um total de quatro reservatórios. Para cada rodada, altera-se a profundidade dos reservatórios para verificação do impacto.

O experimento dura o tempo de enchimento e esvaziamento dos reservatórios. Esta estratégia foi adotada para observar detalhadamente o comportamento do sistema antes e depois da migração. O tempo de execução que cada rodada do benchmark (i.e., cada ponto no gráfico) é de 130 segundos[4]. Cada um dos gráficos a seguir possui uma marcação para indicar o momento exato da migração (linha azul tracejada). Os resultados mostram o **tempo médio da mensagem** e o **nível dos reservatórios**. Para facilitar a visualização, o gráfico do nível do reservatório possui as faixas indicando cada um dos reservatórios utilizados. A Figura 7 apresenta os resultados, onde no lado esquerdo da figura estão os resultado do tempo médio da mensagem (i.e., o eixo Y corresponde ao tempo médio em milissegundos). No lado direito está o nível dos reservatórios (i.e., eixo Y corresponde ao nível dos reservatórios - R_n corresponde ao *n*-ésimo reservatório). O eixo X nos ambos os casos corresponde ao tempo do experimento. Os experimentos obedecem ao seguinte fluxo: *(i)* No momento inicial, há um período de aquecimento, onde o sistema ainda não está sob ataque (i.e., VM VÍTIMA fornecendo serviço e VM ATACANTE ociosa). O tempo de aquecimento para os experimentos foi de 30 minutos. Este tempo foi obtido de observações empíricas realizados com este propósito. Elas mostraram que é o ponto onde o sistema atinge estabilidade. Passado o aquecimento, no segundo momento *(ii)*, o atacante inicia o ataque. Os gráficos possuem uma linha vermelha pontilhada para indicar o início do segundo momento. Finalmente, o terceiro momento *(iii)* é iniciado a partir do disparo da migração.

Os resultados do níveis do reservatório seguem o fluxo esperado. Após o transbordamento de R_4, os níveis dos reservatórios começam a baixar, uma vez que migração foi realizada. O processo de esvaziamento segue um ritmo similar ao de enchimento, resultando assim em uma curva similar a um triângulo com o pico no momento da migração. Também os resultados do tempo de resposta possuem características similares. Como esperado, os efeitos do ataque são (no geral) removidos após a migração, fazendo o sistema se aproximar novamente do desempenho ideal.

Os resultados do tempo médio da mensagem com reservatórios com $D = 100$, mostram que o desempenho após a migração possui oscilações severas. Apesar do *broker* MQTT recuperar significativamente o desempenho após a migração, os resultados sugerem que há resíduos do ataque que permanecem no sistema. Os resultados das outras profundidades reforçam a sugestão ao passo que apresentam um comportamento mais estável. No cenário com $D = 100$, a média dos tempos de mensagem ficam em torno de 16,5 ms, enquanto com $D = 50$, 13,4 ms (redução de aproximadamente 18%) e com D

[4]Esse é o tempo padrão para a execução do benchmark. A carga executa por 100 segundos e há uma pausa de 30 segundos entre os ciclos

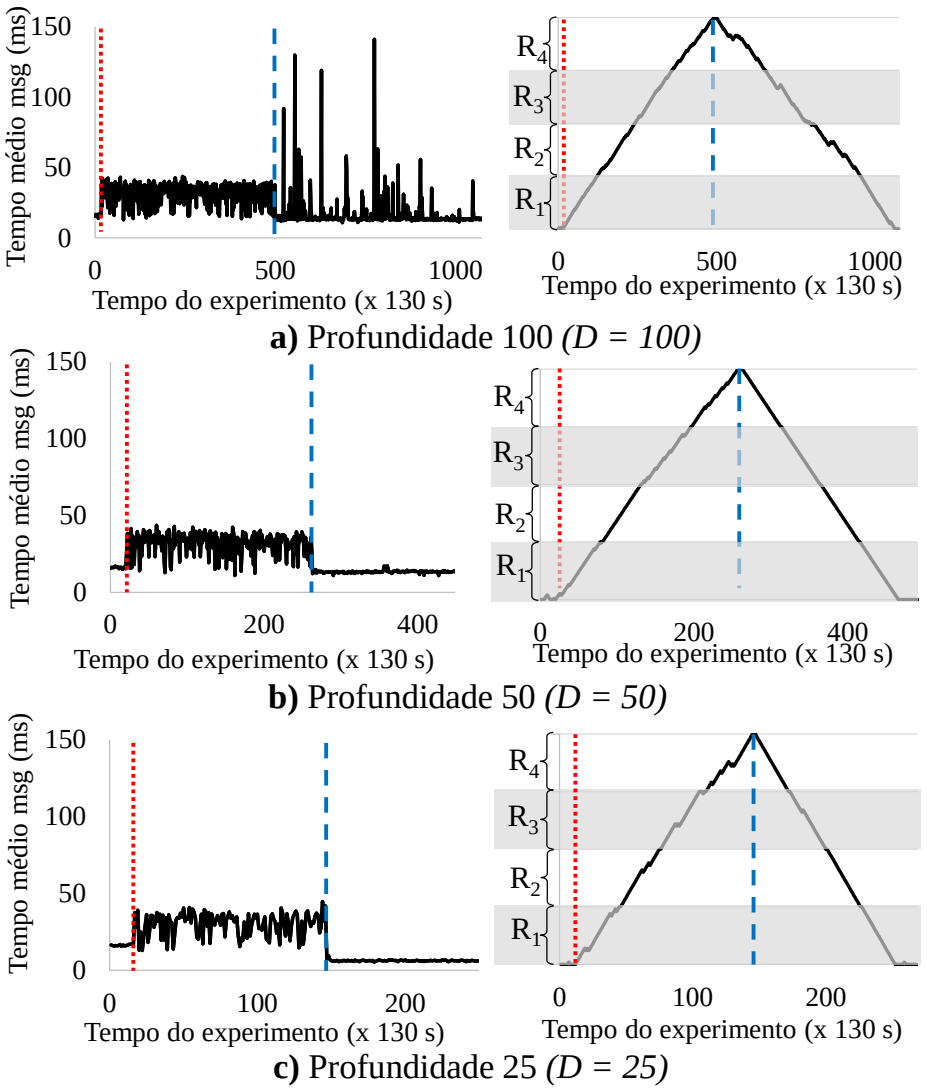


Figura 7. Resultados do experimento de ataque-defesa

= 25, 7,3 ms (redução de 56%)[5]. Assim, os resultados revelam a necessidade de aprofundar as pesquisas para avaliar a relação entre os possíveis resíduos do ataque e o tempo que o sistema passou sob ataque.

4.4. Discussão

Os resultados dos experimentos apresentados nas seções anteriores revelam que os parâmetros do *BA* são a chave para determinar o desempenho do *broker* MQTT sob ataque. De fato, no cenário considerado, utilizar reservatórios mais rasos favorece a qualidade de serviço hospedado na VM VÍTIMA. Os experimentos sugerem que o *broker* MQTT tende a acumular resíduos oriundos do ataque mesmo após a migração da VM. Superficialmente, o impacto de tal resíduo aparenta estar associado com o tempo que o sistema passou sob ataque. Esse efeito tem similaridades com efeitos de *software ag-*

[5]Os resultados são menores que os que foram observados no experimento do *desempenho ideal* (Seção 4.2) porque foram executados na MÁQUINA ALVO que possui configuração ligeiramente superior à MÁQUINA FONTE (onde foram executados os experimentos do *desempenho ideal*)

ing (envelhecimento de software)[Dohi et al. 2020]. Porém, é necessário aprofundar as pesquisas para confirmar essa afirmativa.

Por fim, respondendo a Pergunta de Pesquisa deste artigo, os resultados mostram que a migração é eficaz para defender o *broker MQTT* dos ataques baseados em *(host)*. De modo específico, os resultados sugerem que tempo para disparo da migração influencia na qualidade do serviço após a aplicação da técnica. A Tabela 3 traz um resumo destes efeitos. A última coluna (**Redução**) mostra a diferença entre o desempenho após a migração e o desempenho do sistema indefeso, i.e, 30 ms (Figura 6b). Nota-se que no melhor caso, a técnica utilizada é capaz de reduzir o impacto do ataque em 75%. Os valores também mostram o tempo que o sistema passou sob ataque. Para fins de reprodutibilidade, os scripts de monitoramento, bem como a implementação do *BA*, estão disponíveis no repositório GitHub: https://github.com/matheustor4/automatedVMMigration.

Tabela 3. Efeitos das diferentes políticas de *BA* no desempenho após a migração

Profundidade	**Tempo do início do ataque até a migração**	**Desempenho após a migração**	**Redução**
100	17h 28m	16,5 ms	45%
50	8h 46m	13,4 ms	55%
25	4h 56m	7,3 ms	75%

A seguir são apresentadas limitações identificadas neste artigo, bem como uma proposta de superar os desafios encontrados. Entretanto, devido às restrições de espaço, esta lista é não-exaustiva, de modo que podem existir outros pontos não listados aqui.

Generalização do ambiente IoT. Como os valores usados para aferir e calibrar o ambiente IoT deste artigo foram coletados em um *test-bed* IoT implantado com este propósito e descrito na Tabela 1, os valores podem não representar os parâmetros observados em outros ecossistemas IoT e não é possível generalizar os resultados deste artigo. Como mitigação, este artigo preocupa-se não apenas em apresentar os resultados principais (Seção 4.3), mas também os experimentos prévios (Seções 4.1 e 4.2) utilizados para obtenção dos parâmetros utilizados.

Problemas inerentes ao disparo de alarmes pelo *BA*. Os problemas de disparo de alarme tardio ou falsos alarmes foram discutidos previamente na Seção 3. Estes problemas são inerentes ao algoritmo e dependem apenas da escolha adequada da quantidade e profundidade dos reservatórios. É possível mitigar esse problema após análises de desempenho do ambiente, bem como a após a definição dos acordos de níveis de serviço. Outro ponto importante são as ações tomadas após a recuperação do sistema. Por exemplo, uma vez que o sistema sai do estado de alerta é possível tomar alguma ação (e.g., migrar a VM de volta ou aplicar outra abordagem) para reduzir custos ou melhorar o gerenciamento do sistema. Tais ações, após o cessar do estado de alerta, serão tratadas em trabalhos futuros.

5. Conclusão

Este artigo apresenta uma abordagem para defesa de *brokers* MQTT virtualizados contra ataques MemDoS. Utiliza-se um algoritmo para disparo automatizado de migração de máquinas virtuais. O algoritmo submete as migrações baseando-se na observação de anomalias no tempo de resposta do *broker* MQTT. A proposta traz novas perspectivas para

a defesa de ambientes IoT, que estão presentes em diversos setores. Ele também apresenta um conjunto extenso de resultados de desempenho visando apresentar os potenciais impactos de aplicar a técnica proposta. A solução proposta supera as anteriores em dois pontos principais: 1) verificação de anomalia antes da migração, para evitar migração desnecessária; 2) a migração remove a vítima do ambiente compartilhado com o atacante, conferindo assim maior proteção. Os algoritmos utilizados nos experimentos estão publicamente disponíveis. Por fim, os resultados apresentados sugerem uma melhoria de desempenho de mais de 75% nos melhores cenários estudados.

Os trabalhos futuros aplicarão políticas adicionais de segurança após a saída do estado de alerta do sistema. Outra estratégia para o futuro é manter o monitoramento do *BA* e disparar ações adicionais tão logo o estado de alerta seja superado. Além disso, é importante salientar que outras linhas de pesquisa podem ser desdobradas no futuro ao considerar outras ameaças baseadas em host. Note-se que o ataque *memDoS* é uma instância de *host-based attack*. A inclusão e adaptação para outros *host-based attacks* é possível, uma vez que a migração de VMs pode atuar como defesa competente.

Disponibilidade de artefatos

mqtt-benchmark - https://github.com/krylovsk/mqtt-benchmark[Krylovsk 2024]. A ferramenta foi utilizada para enviar carga de trabalho para o *broker MQTT*.

Memory Denial of Service (MemDoS) - O código-fonte do ataque está disponível no artigo original na referência [Zhang et al. 2017].

Repositório Github - Códigos adicionais para execução dos experimentos. É necessário atentar aos comentários no código para fazer alterações nos caminhos e diretórios utilizados. Disponível em https://github.com/matheustor4/automatedVMMigration

Agradecimentos

Esta pesquisa é parte do INCT de Redes de Comunicação e Internet das Coisas Inteligentes (ICoNIoT), financiado por CNPq (proc. `405940/2022-0`), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Financiamento `88887.954253/2024-00`. Também é financiado pelo CNPq (proc. `403979/2023-4`), e por fundos nacionais portugueses através da FCT – Fundação para a Ciência e a Tecnologia, I.P., no âmbito do projeto UIDB/00326/2025 e UIDP/00326/2025. *Content produced within the scope of the Agenda "NEXUS - Pacto de Inovação - Transição Verde e Digital para Transportes, Logística e Mobilidade", financed by the Portuguese Recovery and Resilience Plan (PRR), with no. C645112083-00000059 (investment project no. .º 53)*

References

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*, pages 1988–1995.

Dohi, T., Trivedi, K. S., and Avritzer, A. (2020). *Handbook of software aging and rejuvenation: fundamentals, methods, applications, and future directions*. World scientific.

Fernandes, G., Rodrigues, J. J., Carvalho, L. F., Al-Muhtadi, J. F., and Proença, M. L. (2019). A comprehensive survey on network anomaly detection. *Telecommunication Systems*, 70:447–489.

Gonçalves, C. F., Menasché, D. S., Avritzer, A., Antunes, N., and Vieira, M. (2023). Detecting anomalies through sequential performance analysis in virtualized environments. *IEEE Access*, 11:70716–70740.

Islam, U., Al-Atawi, A., Alwageed, H. S., Ahsan, M., Awwad, F. A., and Abonazel, M. R. (2023). Real-time detection schemes for memory dos (m-dos) attacks on cloud computing applications. *IEEE Access*.

Krylovsk (2024). Krylovsk/mqtt-benchmark: Mqtt broker benchmarking tool, disponível em https://github.com/krylovsk/mqtt-benchmark, acesso em dezembro/2024.

Kusumi, K. and Koide, H. (2024). Mqtt-mtd: Integrating moving target defense into mqtt protocol as an alternative to tls. In *2024 7th International Conference on Advanced Communication Technologies and Networking (CommNet)*, pages 1–8. IEEE.

Li, W., Manickam, S., Nanda, P., Al-Ani, A. K., Karuppayah, S., et al. (2024). Securing mqtt ecosystem: Exploring vulnerabilities, mitigations, and future trajectories. *IEEE Access*.

Light, R. A. (2017). Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13):265.

Saputro, N., Tonyali, S., Aydeger, A., Akkaya, K., Rahman, M. A., and Uluagac, S. (2020). A review of moving target defense mechanisms for internet of things applications. *Modeling and Design of Secure Internet of Things*, pages 563–614.

Siddharthan, H., Deepa, T., and Chandhar, P. (2022). Senmqtt-set: An intelligent intrusion detection in iot-mqtt networks using ensemble multi cascade features. *IEEE Access*, 10:33095–33110.

Torquato, M., Jesus, B., Silva, F. A., and Cerqueira, E. (2024). Empirical observation of execution throttling as mqtt broker defense against memory denial of service attacks. In *Proceedings of the 13th Latin-American Symposium on Dependable and Secure Computing*, LADC '24, page 184–187, New York, NY, USA. Association for Computing Machinery.

Torquato, M., Maciel, P., and Vieira, M. (2021). Pymtdevaluator: A tool for time-based moving target defense evaluation: Tool description paper. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 357–366. 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)*, pages 1–6. IEEE.

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*, pages 253–265.