O uso de Security Chaos Engineering para ambientes serverless

Rafael Mendes Pinto

Programa de Pós-Graduação em Ciência da Computação
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil

rafael.mendes.negocios@gmail.com

Abstract. *Serverless architectures shift infrastructure management to the cloud provider, but customers remain responsible for permissions, event validation, and cost control. This paper applies Security Chaos Engineering to test whether these controls work. Three experiments were run with AWS Fault Injection Service, one per pillar of the CIA triad: over-privileged permissions, tampered payloads, and Denial of Wallet. Despite best-practice controls, the experiments revealed detection without containment, an alarm that stopped notifying after the first attack, and a concurrency limit that delayed cost alerts.*

Resumo. *A adoção de arquiteturas serverless transfere ao provedor de nuvem a gestão da infraestrutura, mas mantém com o cliente a responsabilidade por permissões, validação de eventos e controle de custos. Este artigo aplica a Engenharia do Caos de Segurança para testar se esses controles funcionam. Com o AWS Fault Injection Service, foram executados três experimentos, um por pilar da tríade CID: permissão excessiva, payload adulterado e Denial of Wallet. Mesmo com boas práticas, os experimentos revelaram detecção sem contenção, um alarme que deixava de notificar após o primeiro ataque e um limite de concorrência que atrasava os alertas de custo.*

1. Introdução

A computação serverless consolidou-se como um dos principais paradigmas de desenvolvimento em nuvem na última década [Kumari et al. 2026]. Nesse modelo, o desenvolvedor escreve funções acionadas por eventos, e o provedor assume os servidores, a escalabilidade e a cobrança por uso. Todos os grandes provedores ampliaram suas ofertas serverless, e a redução do custo de hospedagem é o principal motivo de adoção [Eismann et al. 2022]. A segurança, porém, passa a ser uma responsabilidade compartilhada entre o provedor e quem desenvolve as funções [Marin et al. 2022].

Essa divisão cria superfícies de ataque próprias. Funções com permissões excessivas, eventos que chegam por fontes não-HTTP sem validação e o Denial of Wallet, que explora a cobrança por uso para inflar a conta da vítima, estão entre os riscos destacados pela indústria [OWASP 2018, CSA 2019] e pela academia [Kelly et al. 2021]. A literatura, no entanto, concentra-se em propor mecanismos de proteção e raramente verifica se eles funcionam: em uma revisão de 51 trabalhos, apenas três fazem algum tipo de avaliação de segurança [Escaleira et al. 2025].

Este trabalho aplica a Engenharia do Caos de Segurança para verificar, na prática, se os controles de um ambiente serverless detectam, contêm e respondem a falhas de

segurança. A técnica serve para verificar como os controles que protegem a confidencialidade, a integridade e a disponibilidade (CID) reagem diante de uma falha.

Assim, as principais contribuições deste trabalho são:

- um modelo de ameaças para uma aplicação serverless orientada a eventos na AWS;
- três experimentos de caos de segurança, um por pilar da tríade CID, orquestrados pelo AWS Fault Injection Service e declarados como código;
- evidências de que controles configurados conforme as boas práticas ainda falham na prática: um detecta sem conter, outro deixa de alertar após o primeiro ataque e dois interferem entre si.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta os conceitos de computação serverless e de Engenharia do Caos. A Seção 3 discute os trabalhos relacionados. A Seção 4 descreve a abordagem proposta, e a Seção 5 detalha os experimentos. A Seção 6 apresenta e discute os resultados, e a Seção 7 traz as considerações finais e os trabalhos futuros.

2. Fundamentação Teórica

2.1. Computação Serverless

Na computação serverless, a aplicação é dividida em funções pequenas e sem estado, executadas sob demanda em resposta a eventos, com cobrança apenas pelos recursos consumidos [Marin et al. 2022]. Os servidores continuam existindo; quem os configura e gerencia é o provedor. As funções costumam ser combinadas com serviços gerenciados: armazenamento, bancos de dados e mensageria em nuvem aparecem em 61%, 47% e 38% das 89 aplicações analisadas por [Eismann et al. 2022], das quais 80% rodam na AWS.

Esse modelo desloca a fronteira de responsabilidade. O provedor protege a infraestrutura e o ambiente de execução, enquanto cabem ao cliente as permissões de cada função, a validação dos eventos recebidos, a configuração de limites como timeout e concorrência e o monitoramento [Marin et al. 2022, OWASP 2018].

2.2. Superfícies de Ataque em Ambientes Serverless

Este trabalho se concentra em três ameaças, resumidas na Tabela 1. A primeira são as permissões excessivas: como cada função tem seu próprio papel, é possível aplicar o privilégio mínimo, mas funções com privilégios acima do necessário tornam-se o alvo principal para o acesso indevido a recursos da conta [OWASP 2018, CSA 2019]. A segunda é a injeção de dados por eventos: as funções podem ser acionadas por armazenamento, filas, streams e notificações, e para esses eventos não há um perímetro onde posicionar um firewall [OWASP 2018]. A terceira é o Denial of Wallet, a invocação intencional, em massa e contínua de funções para esgotar financeiramente a vítima por meio de contas infladas [Kelly et al. 2021]. Como a plataforma escala para absorver a carga, o prejuízo aparece no custo dessa escala, ao contrário do DoS, que busca derrubar o ser-

viço. Os limites que conteriam o ataque, por sua vez, tendem a convertê-lo em negação de serviço [Kelly et al. 2021, OWASP 2018].

Tabela 1. Principais ameaças em ambientes serverless e pilares afetados

Ameaça	Descrição	Pilar CID
Permissões excessivas	Função com privilégios além do necessário acessa recursos fora do seu escopo (SAS-4)	Confidencialidade
Injeção de eventos	Eventos de fontes não-HTTP chegam à função sem um perímetro que os valide (SAS-1)	Integridade
Denial of Wallet	Invocações em massa ou prolongadas inflam a cobrança por uso (SAS-8)	Disponibilidade

2.3. Engenharia do Caos e Engenharia do Caos de Segurança

A Engenharia do Caos surgiu na Netflix e se apoia em quatro princípios: definir o estado estável a partir de métricas observáveis, formular hipóteses sobre esse estado, injetar eventos que reproduzam situações reais e automatizar os experimentos [Basiri et al. 2016]. A AWS incorpora a prática ao seu framework de boas práticas e recomenda estado estável mensurável, hipóteses explícitas, condições de parada, rollback para o estado íntegro original e testes primeiro em ambiente de não produção [AWS 2026b].

A Engenharia do Caos de Segurança (SCE) aplica esses princípios aos controles de segurança e é definida como a disciplina de instrumentação, identificação e remediação de falhas nos controles de segurança [Jolak et al. 2026]. A diferença para o teste de intrusão está no objetivo: o pentest procura vulnerabilidades a explorar, enquanto a SCE parte de uma falha injetada de propósito e verifica se os controles a detectam, contêm e respondem a ela. Cada experimento termina com a confirmação ou a refutação de uma hipótese sobre o comportamento dos controles.

3. Trabalhos Relacionados

A Engenharia do Caos ganhou forma como disciplina a partir do trabalho de [Basiri et al. 2016], engenheiros da Netflix que descrevem a prática de provocar falhas de propósito em produção para entender como o sistema reage. A ideia nasceu com o Chaos Monkey, que desligava servidores aleatoriamente, e evoluiu para experimentos mais amplos, como simular a queda de uma região inteira da AWS. Os autores organizam essa experiência em quatro princípios: definir o comportamento normal do sistema a partir de métricas observáveis, formular hipóteses sobre esse comportamento, injetar eventos que reproduzam situações reais e automatizar os experimentos para que rodem de forma contínua. O foco do trabalho, porém, é a disponibilidade. As falhas estudadas são de infraestrutura e de comunicação entre serviços, e a segurança não aparece como objeto dos experimentos.

Do lado da segurança em serverless, [Escaleira et al. 2025] fazem uma revisão sistemática de 51 trabalhos publicados entre 2018 e 2024 e mostram que a pesquisa se concentrou em propor novos mecanismos de proteção, como isolamento de funções e controle de fluxo entre elas. O que chama atenção é a pouca preocupação em verificar se esses mecanismos funcionam de fato: quase metade dos trabalhos não define um modelo de ameaças, e apenas três fazem algum tipo de avaliação de segurança. Os autores defendem que estudos futuros testem as proteções contra ataques realistas, em cenários próximos dos ambientes de produção.

No campo da SCE, [Torkura et al. 2020] propõem a injeção de falhas guiada por risco e a ferramenta CloudStrike, que injeta falhas que afetam a confidencialidade, a integridade e a disponibilidade de infraestruturas na AWS e no Google Cloud, com foco em erros de configuração. Em revisão sistemática com 19 estudos de SCE, [Jolak et al. 2026] mostram que a nuvem é o domínio predominante (47,4% dos estudos), que a elevação de privilégio é pouco explorada (26,3%) e não abordam ambientes serverless.

É justamente nesse ponto que a Engenharia do Caos pode contribuir. Se a revisão de Escaleira et al. aponta a falta de validação prática dos controles de segurança em serverless, o método proposto por Basiri et al. oferece uma forma estruturada de fazer essa validação, com hipóteses claras e experimentos controlados. Este artigo leva a Engenharia do Caos da disponibilidade para a segurança em ambientes serverless, usando a injeção deliberada de falhas para verificar se os mecanismos de detecção, contenção e resposta de um ambiente serverless se comportam como esperado.

4. Abordagem Proposta

A abordagem consiste em submeter uma aplicação serverless, configurada conforme as boas práticas, a falhas de segurança injetadas de forma controlada e medir se os controles reagem como esperado. As ameaças são delimitadas por um modelo de ameaças (Seção 4.1), cada experimento segue um ciclo fixo (Seção 4.2) e todos são executados no mesmo ambiente (Seção 4.3).

4.1. Modelo de Ameaças

O modelo de ameaças é explicitado porque quase metade dos trabalhos revisados por [Escaleira et al. 2025] não o define. Os ativos protegidos são os dados gravados pela aplicação no S3, um objeto sensível armazenado em um bucket fora do escopo da função B e o orçamento da conta. Assume-se um atacante que já obteve uma de três capacidades, cada uma ligada a um pilar da CID: (i) execução de código em B combinada a uma alteração indevida de permissão, como a causada por um erro de implantação ou por credenciais vazadas (confidencialidade); (ii) controle sobre o conteúdo dos eventos enviados a B, como se a função A estivesse comprometida (integridade); e (iii) capacidade de provocar falhas ou lentidão em B sob carga, que, somada às novas tentativas automáticas, ameaça o custo e a disponibilidade (disponibilidade).

O provedor de nuvem é considerado confiável: ataques ao isolamento entre funções, canais laterais e vulnerabilidades do ambiente de execução estão fora do escopo, assim como o comprometimento das credenciais administrativas da conta. O modelo

parte do pressuposto de que o atacante já obteve essas capacidades e se concentra em saber se os controles do cliente percebem e limitam o dano.

4.2. Ciclo de um Experimento de Caos de Segurança

Cada experimento segue o ciclo da Figura 1, adaptado dos princípios de [Basiri et al. 2016]. Primeiro, o **estado estável** é medido sem falhas. Em seguida, formula-se uma **hipótese** sobre como os controles de segurança devem reagir, com um critério verificável. A falha é então **injetada** de forma controlada pelo AWS Fault Injection Service (FIS) [AWS 2026a], enquanto métricas e logs são **observados**. Por fim, os resultados são **analisados** contra a hipótese, e as falhas encontradas nos controles são corrigidas antes da próxima rodada.

O experimento é mantido sob controle por uma **condição de parada** (o FIS interrompe a injeção se a taxa de falha da função de entrada passar de 5%) e por um **rollback garantido**, executado mesmo se o experimento falhar ou for interrompido. Esses mecanismos seguem as recomendações da AWS para experimentos de caos, que desaconselham scripts sem condição de parada e rollback [AWS 2026b].

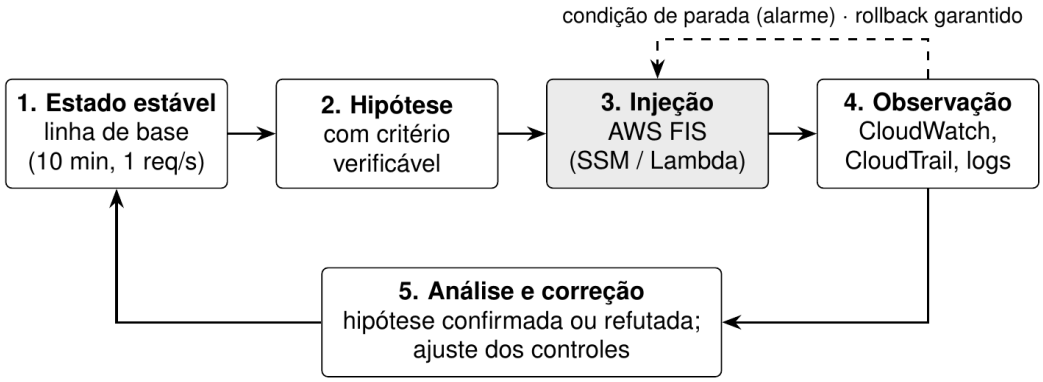


Figura 1. Ciclo de um experimento de Engenharia do Caos de Segurança

4.3. Ambiente Experimental

O alvo é um fluxo assíncrono mínimo, típico de arquiteturas orientadas a eventos: um gerador de tráfego invoca a função **A** (produtor), que encaminha cada pedido de forma assíncrona à função **B** (consumidor); B valida o evento e o grava em um bucket S3. As funções usam Node.js 22 em arm64, com 128 MB e *timeout* de 5 s, na região us-east-1. Toda a infraestrutura é declarada como código (AWS SAM) em um único *stack*, o que torna o ambiente reproduzível e descartável.

O ambiente parte de uma configuração considerada correta, e os experimentos verificam se ela se sustenta. Cada função tem um papel IAM próprio de privilégio mínimo: A só pode invocar B, e B só pode ler e gravar no prefixo de dados do bucket. B aplica um *schema* estrito (campos obrigatórios, tipos, faixa de valores e rejeição de campos extras). A invocação assíncrona de B tem até 2 novas tentativas e idade máxima de evento de 600 s. A detecção combina uma trilha do CloudTrail, uma regra do Event-

Bridge para alterações de política no papel de B, alarmes do CloudWatch (rejeições, invocações por minuto e amplificação B/A) e notificação por SNS.

O estado estável foi medido em 10 min de tráfego a 1 requisição/s: 600 invocações em cada função, nenhum erro ou *throttle*, duração média de 35,8 ms (A) e 59,9 ms (B) e fator de amplificação B/A igual a 1,00. Durante os experimentos, um tráfego legítimo de fundo (1 req/s em E1 e E2; rajada de 5 req/s em E3) começa 30 s antes da injeção e termina 60 s depois dela.

5. Experimentos

Foram definidos três experimentos, um por pilar da tríade CID, resumidos na Tabela 2. Todos são orquestrados pelo FIS: E1 e E2 executam *runbooks* do SSM Automation por meio da ação `aws:ssm:start-automation-execution`, e E3 usa as ações nativas do FIS para Lambda [AWS 2026a].

Tabela 2. Desenho dos experimentos segundo os pilares CID

Id	Pilar	Falha injetada	Controle avaliado	Critério da hipótese
E1	Confidencialidade	Política `s3:*` em `*` anexada ao papel de B	Detecção de mudança de IAM (CloudTrail, EventBridge)	Detecção em até 300 s
E2	Integridade	Eventos adulterados enviados a B (30 por rodada)	Validação de *schema* e alarme de rejeição	100% rejeitados, 0 gravações, alarme em até 300 s
E3	Disponibilidade (custo)	Erro ou atraso nas invocações de B por 180 s	Novas tentativas limitadas, concorrência e alarmes de custo	Custo/pedido de até 2× o estável e alarme antes de 180 s

5.1. E1 — Confidencialidade: permissão excessiva

Hipótese. Se uma política com curinga for anexada ao papel de B, a alteração é detectada em até 300 s, antes de ser revertida.

Injeção. O *runbook* executa cinco passos: (i) uma sondagem de controle, em que B tenta ler um objeto de um segundo bucket, fora do seu escopo; (ii) a anexação de uma política inline `s3:*` em `*` ao papel de B; (iii) uma espera de 20 s para a propagação do IAM; (iv) uma nova sondagem, que simula um atacante com execução de código em B; e (v) uma janela de exposição de 1 min. Ao final, a política é removida e uma sondagem confirma que o acesso voltou a ser negado.

Observação. Tempo médio de detecção (MTTD), do fim da injeção até o registro do evento no *log group* de detecções; ocorrência do acesso indevido; e sua visibilidade nos logs de B e no CloudTrail.

5.2. E2 — Integridade: payload adulterado

Hipótese. Se B receber eventos adulterados, como se A estivesse comprometida, B rejeita todos, não grava nada no S3 e o alarme de rejeição dispara em até 300 s.

Injeção. O *runbook* invoca B de forma assíncrona, pelo mesmo caminho usado por A, com 10 eventos de cada tipo: campo extra malicioso (`is_admin` e um destino de exfiltração), tipo errado (valor como texto) e valor fora da faixa. Cada evento carrega uma marca da rodada, o que permite contar rejeições e localizar objetos indevidos.

Observação. Taxa de rejeição por tipo, objetos indevidos no S3 e tempo até o alarme.

5.3. E3 — Disponibilidade: Denial of Wallet

Hipótese. Com B falhando, as novas tentativas multiplicam as invocações, mas os controles mantêm o custo por pedido em até 2× o do estado estável, e um alarme de custo dispara antes do fim da falha (180 s).

Injeção. Sob rajada de 5 req/s, três variantes de 180 s: (i) `aws:lambda:invocation-error` em 100% das invocações (3 rodadas) e em 50% (1 rodada), o que simula uma falha rápida; (ii) `aws:lambda:invocation-add-delay` de 6 s, acima do *timeout*, o que simula uma falha lenta (1 rodada); e (iii) a mesma falha lenta com concorrência reservada de 5 execuções em B (1 rodada).

Observação. Fator de amplificação (invocações de B por invocação de A), pico de concorrência, *throttles*, idade máxima dos eventos na fila assíncrona, custo estimado por pedido (Lambda e S3, sem *free tier*) e tempo até o primeiro alarme de custo.

6. Resultados e Discussão

A Tabela 3 consolida os resultados; com 3 rodadas, os valores são média ± desvio-padrão. Nenhum experimento acionou a condição de parada, e o rollback funcionou em todas as rodadas.

Tabela 3. Resultados dos experimentos

Id	n	Hipótese	Tempo até detecção	Resultado principal
E1	3	3/3	2,5 ± 0,5 s	Acesso indevido em 3/3; detecção sem contenção
E2	3	3/3	77,1 ± 8,8 s	100% rejeitado; 0 gravações; alarme falhava sem `DefaultValue`
E3 erro 100%	3	3/3	61,7 ± 11,9 s	Fator B/A 1,96 ± 0,03; custo/pedido 1,03×
E3 erro 50%	1	1/1	60,5 s	Fator B/A 1,37; custo/pedido 1,01×
E3 atraso	1	0/1	47,6 s	Custo/pedido 2,44×; 87 execuções simultâneas

E3 atraso + reserva	1	0/1	241,6 s	Custo/pedido 1,12×; fila com eventos de 492 s

E1: detectar não é conter. A alteração de permissão foi detectada em 2,5 ± 0,5 s em todas as rodadas, bem abaixo do limiar. Ainda assim, o objeto sensível foi lido nas três rodadas. O acesso foi confirmado pela comparação das três sondagens de cada rodada: negado antes da injeção, permitido durante a exposição e negado de novo após o rollback, sequência visível tanto nos logs de B quanto no CloudTrail. O controle avaliado é apenas detectivo; sem uma resposta automática que reverta a política, a exposição dura até alguém reagir ao alerta.

E2: o experimento revelou uma falha no próprio controle. A validação rejeitou 100% dos 90 eventos adulterados, com o motivo correto em cada tipo, e nenhum objeto indevido foi gravado. O alarme disparou em 77,1 ± 8,8 s, tempo dominado pelo período de 1 min da métrica: a primeira rejeição ocorreu em cerca de 2 s. O problema apareceu na comparação entre as rodadas da primeira bateria: o alarme disparou na rodada 1 e permaneceu em estado de alerta nas seguintes, sem gerar novas notificações. Sem `DefaultValue`, o filtro de métrica só publicava pontos quando havia rejeição, e o CloudWatch manteve o último estado. Em produção, apenas o primeiro ataque seria notificado. A correção foi trivial, e a falha só se tornou visível porque o controle foi exercitado repetidamente. Trata-se de um caso de monitoramento inadequado, risco listado entre os mais críticos para aplicações serverless [CSA 2019].

E3: o custo vem da falha lenta, e os controles entram em conflito. A falha rápida amplificou as invocações (fator 1,96, com pico de 2,74 por minuto, contra 3× teóricos para duas novas tentativas), mas quase não alterou o custo por pedido, pois invocações que falham logo duram milissegundos. A falha lenta é o verdadeiro *Denial of Wallet*: cada invocação é cobrada até o *timeout*, B escalou até 87 execuções simultâneas e o custo por pedido chegou a 2,44×, o que refuta a hipótese. Com concorrência reservada, o consumo caiu de 889 para 90 GB-s (−90%) e nenhum pedido foi perdido. Em troca, B sofreu 2.366 *throttles*, que não contam como invocações: os alarmes de custo só dispararam em 241,6 s, depois do fim da falha, e os eventos represados atingiram 492 s de idade, perto do limite de 600 s a partir do qual seriam descartados sem aviso. O resultado confirma empiricamente a observação de [OWASP 2018] e [Kelly et al. 2021] de que as proteções contra o Denial of Wallet tendem a se converter em negação de serviço.

Os três casos mostram o que a Engenharia do Caos de Segurança acrescenta à revisão de configuração: todos os controles estavam corretamente declarados, e ainda assim um não continha o ataque (E1), outro falhava na repetição (E2) e dois interferiam entre si (E3). Os resultados apontam ajustes concretos: resposta automática acionada pelo EventBridge para reverter políticas, `DefaultValue` em filtros de métrica e alarmes

sobre *throttles* e idade dos eventos assíncronos, com um destino para eventos descartados.

6.1. Limitações e Ameaças à Validade

O tráfego é sintético e de baixa taxa, as variantes secundárias do E3 têm uma única rodada e os limiares das hipóteses foram explicitados na análise, não antes da execução. Os custos são estimativas a partir de métricas e preços públicos, e os resultados vêm de uma única conta, região e provedor.

7. Considerações Finais

Este trabalho aplicou a Engenharia do Caos de Segurança a uma aplicação serverless na AWS, com três experimentos orquestrados pelo AWS FIS, um para cada pilar da tríade CID. Mesmo com os controles configurados conforme as boas práticas, os experimentos revelaram lacunas que a revisão da configuração não mostrava: detecção sem contenção, um alarme que silenciava após o primeiro ataque e um controle de custo que atrasava os alertas. O caso mais revelador foi o do E3: limitar a concorrência parecia a defesa óbvia contra o Denial of Wallet e reduziu o consumo em 90%, mas, como só a execução do experimento mostrou, também empurrou os alarmes de custo para depois do fim da falha. Como trabalho futuro, pretende-se executar os experimentos de forma contínua no pipeline de implantação e automatizar também a resposta aos alertas.

Referências

Amazon Web Services (2026a). *AWS Fault Injection Service: User Guide*. Disponível em: https://docs.aws.amazon.com/fis/latest/userguide/. Acesso em: 9 nov. 2026.

Amazon Web Services (2026b). REL12-BP04 Testar a resiliência usando engenharia do caos. In *AWS Well-Architected Framework*. Disponível em: https://docs.aws.amazon.com/wellarchitected/latest/framework/rel_testing_resiliency_failure_injection_resiliency.html. Acesso em: 10 nov. 2026.

Basiri, A., Behnam, N., de Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., and Rosenthal, C. (2016). Chaos engineering. *IEEE Software*, 33(3):35–41.

CSA – Cloud Security Alliance (2019). *The 12 Most Critical Risks for Serverless Applications*. Cloud Security Alliance, Serverless Working Group.

Eismann, S., Scheuner, J., van Eyk, E., Schwinger, M., Grohmann, J., Herbst, N., Abad, C. L., and Iosup, A. (2022). The state of serverless applications: Collection, characterization, and community consensus. *IEEE Transactions on Software Engineering*, 48(10):4152–4166.

Escaleira, P., Cunha, V. A., Barraca, J. P., Gomes, D., and Aguiar, R. L. (2025). A systematic review on security mechanisms for serverless computing. *Cluster Computing*, 28:465.

Jolak, R., Mohamad, M., Avula, R., Meek, J., and Åström, A. (2026). SCENE: Guidelines for security chaos engineering based on a systematic literature review. *Journal of Systems and Software*, 239:112896.

Kelly, D., Glavin, F. G., and Barrett, E. (2021). Denial of wallet – defining a looming threat to serverless computing. *Journal of Information Security and Applications*, 60:102843.

Kumari, A., Behera, R. K., Sahoo, K. S., Sahoo, B., and Gandomi, A. H. (2026). Emergence of serverless computing in cloud: A state-of-the-art review of challenges and opportunities. *Machine Learning with Applications*, 24:100862.

Marin, E., Perino, D., and Di Pietro, R. (2022). Serverless computing: a security perspective. *Journal of Cloud Computing*, 11. DOI: 10.1186/s13677-022-00347-w.

OWASP (2018). *OWASP Top 10 (2017) Interpretation for Serverless*. Relatório provisório, OWASP Serverless Top 10 Project (líder: T. Melamed).

Torkura, K. A., Sukmana, M. I. H., Cheng, F., and Meinel, C. (2020). CloudStrike: Chaos engineering for security and resiliency in cloud infrastructure. *IEEE Access*, 8:123044–123060.