Kumo: A Security-Focused Serverless Cloud Simulator

Wei Shao*, Khaled N. Khasawneh†, Setareh Rafatirad*, Houman Homayoun*, and Chongzhou Fang‡
*University of California, Davis, USA
Email: {wayshao, srafatirad, hhomayoun}@ucdavis.edu
†George Mason University, USA
Email: kkhasawn@gmu.edu
‡Rochester Institute of Technology, USA
Email: cxfeec@rit.edu

***Abstract*—Serverless computing abstracts infrastructure management but also obscures system-level behaviors that can introduce security risks. Prior work has shown that serverless platforms are vulnerable to attacks exploiting shared execution environments, including attacker–victim co-location and denial-of-service through resource contention, yet analyzing these risks on production platforms is difficult due to limited observability, high cost, and lack of experimental control, while existing simulators primarily focus on performance and cost rather than security. We present Kumo, a security-focused simulator for serverless platforms that enables controlled, reproducible analysis of security risks arising from scheduling and resource sharing decisions. Kumo models invocation arrivals, scheduler placement, container reuse, resource contention, and queuing within a discrete-event framework, explicitly representing attackers and victims as first-class entities and providing metrics such as co-location probability, time to first co-location, invocation drop rate, and tail latency. Through two case studies, we show that scheduler choice is a first-order factor for co-location attacks, inducing orders-of-magnitude differences under identical workloads, while Denial-of-Service behavior is largely governed by system-level factors such as service time, queuing policy, and cluster capacity once contention dominates. These results highlight the need to distinguish scheduler-driven isolation risks from broader resource exhaustion vulnerabilities and position Kumo as a flexible foundation for systematic, security-aware exploration of serverless platforms.**

***Index Terms*—Serverless computing, cloud security, multi-tenant isolation, scheduling, resource contention, denial-of-service attacks, simulation, cloud platforms.**

I. Introduction

Serverless computing is widely adopted for its elasticity, simplified deployment, and fine-grained billing. By abstracting away infrastructure management, however, serverless platforms also obscure scheduling, resource sharing, and container reuse decisions that can have important security implications [1]–[4].

Recent work shows that serverless platforms are vulnerable to attacks that exploit shared execution environments rather than software bugs [5]–[10]. Co-location attacks exploit placement decisions to place attacker and victim functions on the same worker, while availability attacks exploit contention and queuing to degrade victim performance or cause drops. Studying such attacks on production platforms is difficult due to limited observability, high cost, and poor controllability.

Existing serverless simulators and performance models primarily focus on cost estimation, latency optimization, or capacity planning. While valuable, these tools are not designed to analyze security risks, and typically lack explicit attacker modeling, isolation metrics, or the ability to study scheduler behavior as a security mechanism [11]–[13]. As a result, there is currently no principled way to explore how serverless design choices influence security outcomes under controlled and reproducible conditions.

In this paper, we present **Kumo**, a security-focused simulator for serverless platforms. Rather than replicating any specific commercial platform in full detail, Kumo captures the mechanisms most relevant to serverless security, including scheduler placement, container reuse, resource contention, and queuing. It explicitly models attackers and victims and reports security-relevant metrics such as co-location probability, time to first co-location, invocation drop rate, and tail latency.

We demonstrate Kumo through two case studies. The first analyzes attacker–victim co-location under different scheduling policies and shows that scheduler choice can lead to orders-of-magnitude differences in co-location probability. The second examines Denial-of-Service (DoS) behavior and shows how availability degradation depends on service time, queuing policy, and cluster capacity. Together, these studies distinguish scheduler-driven isolation risks from broader resource exhaustion vulnerabilities.

This paper makes the following contributions:

- We design and implement **Kumo**, a configurable, event-driven simulator for security analysis of serverless platforms, with explicit modeling of scheduling, container reuse, resource contention, and queuing.
- We introduce a flexible framework for modeling attackers and victims at the workload level, enabling direct measurement of security-relevant outcomes such as co-location probability, time to first co-location, and availability degradation.
- We present two in-depth case studies that demonstrate how scheduler design and system-level resource management influence distinct classes of serverless security risks.

By enabling controlled, reproducible exploration of serverless security tradeoffs, Kumo complements empirical studies

on production systems and provides a foundation for principled analysis of serverless platform design.

II. Background and Threat Model

This section provides brief background on the serverless execution model and defines the threat models considered in this work.

A. Serverless Execution Model

In serverless computing, applications are deployed as functions that are executed on demand in response to events. A cloud provider manages a pool of workers that host function containers and allocates resources dynamically. When a function is invoked, the platform selects a worker, initializes or reuses a container, and executes the function. Containers may be reused across invocations, leading to *warm starts*, or freshly initialized, resulting in *cold starts* that incur additional latency.

Serverless platforms are inherently multi-tenant: functions from different users may execute on the same worker and share underlying hardware and software resources. Placement decisions are typically handled by platform schedulers, which are opaque to tenants and optimized primarily for performance and resource efficiency. These characteristics make serverless platforms susceptible to security risks that arise from shared execution environments, even in the absence of software vulnerabilities.

B. Threat Model

We consider adversaries that exploit legitimate serverless execution mechanisms rather than implementation bugs. In all cases, attackers issue only valid function invocations and do not compromise the underlying platform or worker nodes.

1) Co-location Attacks: In a co-location attack, an adversary attempts to place its function execution on the same worker as a victim tenant via exploiting scheduling features [14]–[16]. Successful co-location enables a range of potential side-channel and information leakage attacks by exploiting shared resources [17]–[20]. The attacker does not observe placement decisions directly, but can repeatedly invoke functions to increase the probability of sharing a worker with the victim. An attack is considered successful if attacker and victim functions are co-located on the same worker at any point during execution.

2) Availability (Denial-of-Service) Attacks: In an availability attack, an adversary seeks to degrade victim performance by inducing resource contention [21], [22]. The attacker injects a high volume of function invocations that compete for worker resources, leading to queuing delays, invocation drops, or increased tail latency for victim functions. These attacks do not overwhelm the network or exploit crashes; instead, they rely solely on the platform's resource management and admission control behavior.

C. Out-of-Scope Attacks

Kumo does not model attacks that require low-level microarchitectural leakage, such as cache or speculative execution side channels, nor does it consider network-level distributed DoS attacks. This scope allows Kumo to focus on system-level security risks that emerge from shared execution environments and resource management decisions.

III. Design

A. Design Goals and Scope

Kumo is designed as a security-focused simulator for serverless platforms. Its primary goal is not to replicate any specific commercial platform in full detail, but to enable controlled, reproducible analysis of security risks that arise from cloud control plane decisions, including scheduling and resource sharing. To this end, Kumo is guided by the following design goals.

a) G1: Security-Oriented Modeling: Kumo prioritizes modeling features that are essential for analyzing serverless security risks, including scheduler placement decisions, container reuse (cold vs. warm starts), resource contention, and queuing behavior. Rather than targeting microarchitectural accuracy, Kumo focuses on capturing the system-level mechanisms that enable attacks such as co-location and DoS.

b) G2: Explicit Attacker and Victim Modeling: Unlike performance-oriented simulators, Kumo treats attackers and victims as explicit modeling primitive entities whose identities, behaviors, and outcomes are explicitly modeled. The simulator explicitly supports attacker-controlled workloads that interact with benign tenants through shared platform resources, enabling direct measurement of security-relevant outcomes such as co-location probability, time to first co-location, invocation drops, and tail latency.

c) G3: Reproducible and Controlled Experiments: Kumo is designed to support reproducible experimentation through deterministic event-driven execution and configurable random seeds. Platform parameters, workloads, schedulers, and attacker behavior are all specified explicitly, allowing systematic exploration of security tradeoffs under controlled conditions.

d) G4: Scheduler Pluggability: Scheduling policies are implemented through a clean abstraction that decouples scheduler logic from the execution engine. This allows researchers to evaluate existing serverless schedulers, prototype new policies, and study scheduler behavior as a security mechanism without modifying the core simulator.

e) G5: Low Barrier to Extension: Kumo is intended to be easily extensible. New workloads, schedulers, attack strategies, and metrics can be added with minimal changes to the core engine, enabling rapid prototyping and iterative security analysis.

B. Architecture Overview

Kumo is a discrete-event simulator that models serverless execution as a sequence of invocation arrivals, scheduling decisions, execution events, and resource reclamation. Its architecture cleanly separates workload generation, scheduling

policy, platform state, and metric collection, enabling security-focused analysis under controlled and reproducible conditions. Figure 1 provides an overview of Kumo's architecture and execution flow.

At the core of Kumo is an event-driven simulation engine that advances logical time by processing a priority queue of events. Each function invocation is represented as a series of events, including invocation arrival, scheduling, execution start, and execution completion. This design allows Kumo to efficiently simulate large-scale workloads while preserving precise ordering of security-relevant events such as placement decisions, container reuse, and resource contention.

Invocation arrivals are produced by pluggable workload generators that emit batches of invocations over simulated time. Kumo supports multiple workload models, currently including uniform, Poisson, and bursty arrival processes. To enable security analysis, workloads may be wrapped by attacker modules that inject adversarial invocations alongside benign traffic. Generated invocations are timestamped and submitted to the engine without embedding any scheduling assumptions, ensuring that placement behavior is determined solely by the scheduler under study.

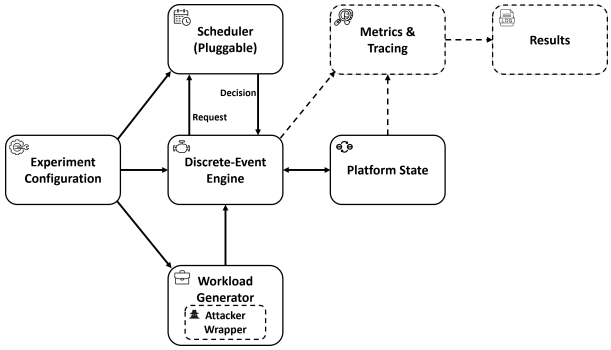

Fig. 1. Architecture of Kumo.

Scheduling decisions are handled through a dedicated scheduler abstraction. Upon invocation arrival, the engine queries the scheduler with the invocation metadata and the current platform state. The scheduler returns a placement decision indicating a selected worker or a failure if no placement is possible. By decoupling scheduling logic from execution, Kumo enables direct comparison of different scheduling policies under identical workloads and platform configurations, isolating the security implications of scheduler behavior.

Kumo maintains an explicit platform state that models workers, containers, and resource usage. Each worker tracks its available CPU, memory, and storage resources, as well as the set of active and idle containers. Containers are associated with specific functions and may be reused across invocations, capturing instance cold and warm start behavior. Resource allocation and reclamation are modeled at invocation start and completion, allowing contention, queuing, and capacity exhaustion effects to emerge naturally during simulation.

When worker resources are insufficient to immediately execute an invocation, Kumo may enqueue the invocation in a per-worker waiting queue, subject to a configurable queue length limit. If the queue is full, the invocation is dropped. This explicit modeling of backpressure is essential for analyzing availability degradation and DoS behavior under overload.

Security- and performance-relevant metrics are collected through a centralized metrics module that observes key events during simulation. Metrics include performance metrics like cold and warm start rates, invocation drop rates, end-to-end latency distributions, and security-related metrics like co-location counts and time to first co-location which are important to micro-architectural side-channel attackers [5], [7], [16]. Metrics collection is decoupled from the execution engine, ensuring that measurement does not influence scheduling or execution behavior.

In a typical execution, workloads generate invocation arrivals that are submitted to the engine. For each invocation, the engine queries the scheduler for placement, updates the platform state accordingly, and schedules execution and completion events. Throughout this process, the metrics module records security-relevant outcomes. This modular execution flow allows Kumo to flexibly combine different workloads, schedulers, and attack strategies within a unified simulation framework.

C. Workloads, Schedulers, and Attack Modeling

Kumo is designed to flexibly model serverless workloads, scheduling policies, and adversarial behavior within a unified simulation framework. These abstractions allow Kumo to express a wide range of security scenarios while maintaining a clear separation between workload generation, scheduling decisions, and attack logic.

a) Workloads: Kumo provides multiple workload generators that capture common serverless invocation arrival patterns. Uniform workloads generate invocations by sampling tenants and functions evenly, and are primarily used for controlled stress testing and debugging. Poisson workloads model more realistic serverless traffic using exponentially distributed inter-arrival times, reflecting production environments. Kumo also supports bursty workloads that alternate between low- and high-intensity phases to capture flash crowds and workload spikes. All workload generators emit timestamped invocation arrivals without embedding any assumptions about placement or execution.

b) Schedulers: Scheduling behavior is modeled through a unified scheduler abstraction that maps invocation requests to workers based on the current platform state. This abstraction allows different scheduling policies to be evaluated under identical workloads and platform configurations, isolating the security implications of scheduler behavior from other system components. In this work, we model several representative schedulers with distinct design objectives:

- *Random* selects a worker uniformly at random from the set of workers with sufficient available resources. It does not consider container locality, tenant identity, or prior placements, and serves as a baseline with no explicit isolation or performance optimization.

- *DoubleDip* [6] prioritizes spreading invocations across workers to reduce repeated co-location. It avoids placing invocations on workers that have recently hosted the same tenant when alternatives exist, and breaks ties by selecting less-loaded workers. This design captures schedulers that seek to improve isolation through placement diversity rather than strict partitioning.
- *Helper* [7] prioritizes container reuse by favoring workers that already host warm containers for the invoked function. This locality-aware policy reduces cold-start overhead but increases the likelihood of repeated co-location, reflecting schedulers that optimize latency at the expense of isolation.
- *OpenWhisk* [23] schedulers emphasize aggressive container reuse and opportunistic instance sharing. They preferentially place invocations on workers with existing containers for the same function, falling back to other workers only when necessary. While effective for performance, this behavior can amplify attacker–victim co-location.

These scheduler models are intentionally simplified to capture dominant placement behaviors relevant to security analysis, rather than to replicate proprietary production heuristics in full detail.

c) Attack Modeling: To support security analysis, Kumo explicitly models attacker behavior at the workload level. Attacker modules wrap benign workloads and inject adversarial invocations that compete for shared platform resources. For co-location attacks, attackers issue invocations intended to increase the likelihood of sharing a worker with a victim tenant. For availability attacks, attackers inject invocations at configurable intensities to induce resource contention, queuing, and invocation drops. Attack parameters such as injection rate, arrival pattern, and service time are explicitly controlled, enabling systematic exploration of attacker capabilities.

Kumo allows experiments to designate specific tenants and functions as victims and tracks victim-specific metrics throughout execution. These metrics include co-location counts, time to first co-location, invocation drop rates, and tail latency. By explicitly labeling victims, Kumo enables direct measurement of security-relevant outcomes rather than relying on aggregate system metrics that may obscure attack impact.

Workloads, schedulers, and attacker models are fully configurable and composable. Any workload can be paired with any scheduler and attacker configuration without modifying the core engine, allowing Kumo to express a wide range of security scenarios using a small set of orthogonal building blocks.

D. Experiment Configuration and Metrics

Kumo is driven by explicit experiment configuration files that fully specify platform parameters, workloads, schedulers, and attacker behavior. This design enables controlled, reproducible security experiments without modifying simulator code.

a) Configuration Parameters: Each experiment configuration defines the serverless platform, workload characteristics, and security scenario under study. Platform parameters include the number of workers, per-worker CPU, memory, and storage resources, optional heterogeneity, container idle timeout, prewarming policy, and per-worker queue limits. Workloads are configurable as uniform, Poisson, or bursty arrival processes, with explicit control over arrival rates, batch sizes, and total invocation volume.

Scheduling policies are selected via a pluggable scheduler interface, and experiments may sweep across multiple schedulers and random seeds to capture stochastic effects. Attacker behavior is explicitly specified, including attacker tenant identity, attack intensity (expressed as attacker invocations per victim invocation), arrival pattern, and victim set. Service-time distributions are configurable (fixed or exponential), enabling controlled stress of system capacity.

Kumo supports parameter sweeps over schedulers, attacker intensity, queue length, cluster size, and other dimensions, allowing systematic exploration of security and performance tradeoffs.

b) Metrics Collection: Kumo outputs a structured CSV trace summarizing both security- and performance-relevant outcomes for each experiment run. Reported metrics include: (i) cold and warm start counts, (ii) attacker–victim co-location count and time to first co-location, (iii) total and victim-specific invocation arrivals and drops, (iv) victim invocation drop rate, (v) victim mean and tail (p95) latency, and (vi) attacker drop rate. Metrics are tracked per tenant, enabling direct measurement of victim impact rather than aggregate system behavior. These metrics directly support the analyses in Case Studies A and B by quantifying isolation risk (co-location metrics) and availability degradation (drop rate and tail latency).

c) Tracing Support: For debugging and validation, Kumo optionally records fine-grained execution traces capturing invocation arrivals, scheduling decisions, execution start and completion events, container reuse, and queuing behavior. Tracing is disabled by default and does not affect scheduling or execution semantics when enabled.

By combining explicit configuration, parameter sweeps, and security-focused metrics, Kumo enables reproducible and transparent evaluation of serverless security risks across a wide range of adversarial scenarios. These configuration and measurement capabilities are exercised in the case studies presented in Section IV.

IV. Evaluation

We evaluate Kumo through two complementary case studies that capture distinct classes of security risks in serverless platforms. Case Study A examines *co-location security*, where attackers attempt to achieve physical co-residency with a victim through scheduler placement decisions. Case Study B focuses on *availability (denial-of-service) attacks*, where attackers degrade victim performance through resource contention and queuing effects without exploiting scheduler logic.

Together, these case studies demonstrate Kumo's ability to disentangle scheduler-driven isolation risks from system-level resource exhaustion vulnerabilities, and to support security analysis across multiple attack surfaces using a unified simulation framework.

A. Case Study A: Attacker–Victim Co-location Security

We first evaluate Kumo's ability to analyze *co-location security*, a prerequisite for many serverless side-channel and information leakage attacks. This case study examines how different serverless scheduling policies influence an attacker's ability to become co-located with a victim tenant under identical workloads and platform configurations.

1) Threat Model: We consider a malicious tenant that repeatedly invokes an attack function in order to increase the likelihood of being scheduled on the same worker as a victim tenant. The attacker does not control the scheduler and cannot observe placement decisions directly. An attack is considered successful if an attacker invocation executes concurrently with a victim invocation on the same worker. This threat model is consistent with assumptions made by prior serverless co-location attacks.

2) Experimental Setting: We simulate a large-scale homogeneous serverless cluster consisting of 512 workers, each provisioned with identical CPU, memory, and storage resources. The platform hosts 200 benign tenants, each owning 20 functions, for a total of 4,000 distinct functions and 20,000 benign invocations. One benign tenant is designated as the victim, while a separate attacker tenant repeatedly invokes a dedicated attack function in an attempt to achieve co-location with the victim. All remaining tenants generate background traffic.

Invocation arrivals follow a Poisson process. Containers are subject to an idle timeout of 60 time units, and no pre-warming is enabled, reflecting a cost-efficient serverless configuration. The attacker injects invocations at a rate proportional to the victim's workload, modeling an adversary that blends into normal system activity rather than overwhelming the platform.

We compare four schedulers—*DoubleDip*, *Random*, *Helper*, and *OpenWhisk*—under identical workload and platform conditions. Each experiment is repeated across 20 random seeds to capture stochastic effects, and all results are reported as means.

Figure 2 summarizes the results of this case study.

a) Co-location Probability: Figure 2(a) reports the probability that an attacker invocation is co-located with the victim per victim invocation. Despite the large cluster size and high tenant diversity, co-location behavior differs substantially across schedulers. *DoubleDip* effectively eliminates co-location, achieving zero probability across runs. In contrast, the *Random* scheduler exhibits high co-location probability, while *Helper* and *OpenWhisk* fall between these extremes. These results illustrate how scheduler placement logic directly shapes co-location outcomes, even at large scale.

b) Cold-Start Overhead: Figure 2(b) shows the cold-start rate of the victim function under each scheduler. The *Random* scheduler incurs a significantly higher cold-start rate, reflecting frequent container churn. In contrast, *DoubleDip*, *Helper*, and *OpenWhisk* exhibit similarly low cold-start rates. This indicates that reduced co-location does not inherently require high cold-start overhead.

c) Security–Performance Tradeoff: Figure 2(c) visualizes the relationship between co-location probability and cold-start rate. *DoubleDip* occupies a favorable region of the design space, combining low co-location probability with low cold-start overhead. Other schedulers exhibit less favorable tradeoffs, either exposing higher co-location risk or incurring increased cold-start cost. This visualization highlights how Kumo enables direct comparison of scheduler-induced security and performance characteristics.

d) Attack Feasibility: Beyond steady-state co-location probability, we examine how quickly an attacker can succeed in practice. Figure 2(d) reports the mean time to first attacker–victim co-location for schedulers that ever observe co-location. The *Random* scheduler allows attackers to achieve co-location relatively quickly, while *Helper* and *OpenWhisk* delay initial co-location by several orders of magnitude in simulated time. Schedulers that never exhibit co-location are omitted from this plot. This time-based metric demonstrates that attack feasibility depends not only on probability but also on how rapidly co-location can occur.

This case study demonstrates that Kumo can capture systematic and persistent differences in co-location behavior across serverless schedulers, even in large-scale deployments with hundreds of workers and thousands of functions. The results show that scheduler placement policies strongly influence both the likelihood, which is consistent with existing literature [6] and also reflects the timing of attacker–victim co-location, while their performance implications can be evaluated simultaneously within the same experimental framework.

B. Case Study B: Denial-of-Service and Availability Degradation

We next demonstrate Kumo's ability to analyze *availability attacks*, focusing on denial-of-service (DoS) behavior in serverless platforms. Unlike co-location attacks, which exploit scheduler placement decisions, DoS attacks aim to degrade victim availability and latency through sustained resource contention and queuing effects. This case study examines how attacker intensity, queueing policies, and cluster scale influence DoS impact under realistic multi-tenant serverless workloads.

a) Threat Model: We consider an attacker tenant that issues a large volume of legitimate function invocations to contend for shared worker resources. The attacker does not crash workers, exploit implementation bugs, or generate network-level floods. Instead, the attack relies solely on invocation-level pressure to exhaust execution capacity and queue space. An attack is considered successful if it increases victim invocation drop rates or significantly inflates tail latency.

b) Experimental Setting: We evaluate DoS behavior using three controlled experiments that vary attacker intensity, queuing policy, and cluster scale. Unless otherwise stated, we

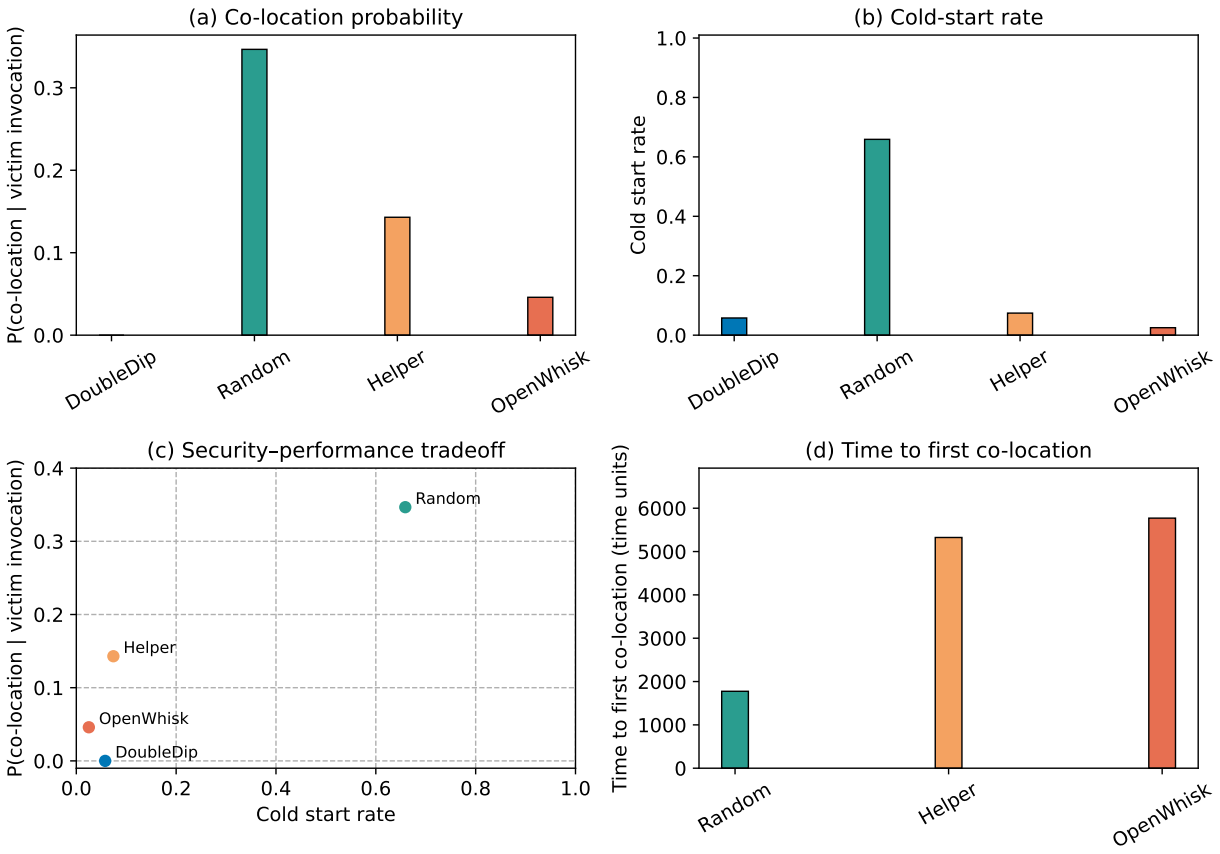

Fig. 2. Case Study A: Attacker–victim co-location under different schedulers in a large-scale serverless setting.

simulate a homogeneous serverless cluster with 512 workers, each provisioned with identical CPU, memory, and storage resources. The platform hosts 200 benign tenants, each owning 20 functions, and 20,000 benign invocations in total, generating background traffic representative of large multi-tenant deployments.

Workloads follow a Poisson arrival process, with arrival rates tuned such that the system operates near its capacity threshold even without an attacker. Service times follow an exponential distribution with a mean of 100 time units, capturing realistic execution variability. Containers are subject to a fixed idle timeout, and pre-warming is disabled. Each experiment is repeated across 20 random seeds, and results are reported as means.

Specifically, (i) to study attacker intensity, we sweep the scheduler and attacker injection rate while holding cluster size and queue limits fixed; (ii) to study queuing effects, we sweep the maximum per-worker queue length under a fixed attacker intensity and Helper scheduler; and (iii) to study scalability, we vary the number of workers while keeping workload characteristics constant and Helper scheduler. These experiments correspond to Figures 3, 4, and 5, respectively.

c) Impact of Attacker Intensity: Figure 3 sweeps attacker intensity from 0 to 10 attacks intensities and measures victim drop rate and p95 latency. Two observations stand out. First, DoS impact increases with attacker intensity as expected: higher adversarial load pushes the system closer to saturation, which increases both the probability of queue overflow (drops) and the waiting time of successfully served invocations (tail latency). Second, the scheduler policy does affect DoS outcomes in this near-capacity regime. In particular, the Helper scheduler exhibits noticeably higher victim drop rates as intensity grows, while DoubleDip remains zero drop rate and nearly flat p95 latency across the sweep. The Random scheduler behaves in-between: it largely avoids drops at low-to-moderate intensity, but its tail latency rises and drops appear at the highest intensity. These differences are consistent with the intuition that schedulers can either concentrate contention (creating hotspots that overflow queues) or spread load to avoid per-worker collapse, even when the total offered load is the same. Kumo makes this observation explicit by modeling resource contention and backpressure directly.

d) Queue Length Tradeoffs: Figure 4 sweeps the per-worker maximum queue length under a fixed attacker intensity (5.0). Increasing queue capacity reduces victim drop rate because fewer requests are rejected when workers are temporarily saturated. However, this mitigation comes with a clear tail-latency cost: as queue length grows, more requests wait longer before service, increasing victim p95 latency. The curve shows a visible “knee” (roughly around queue lengths in

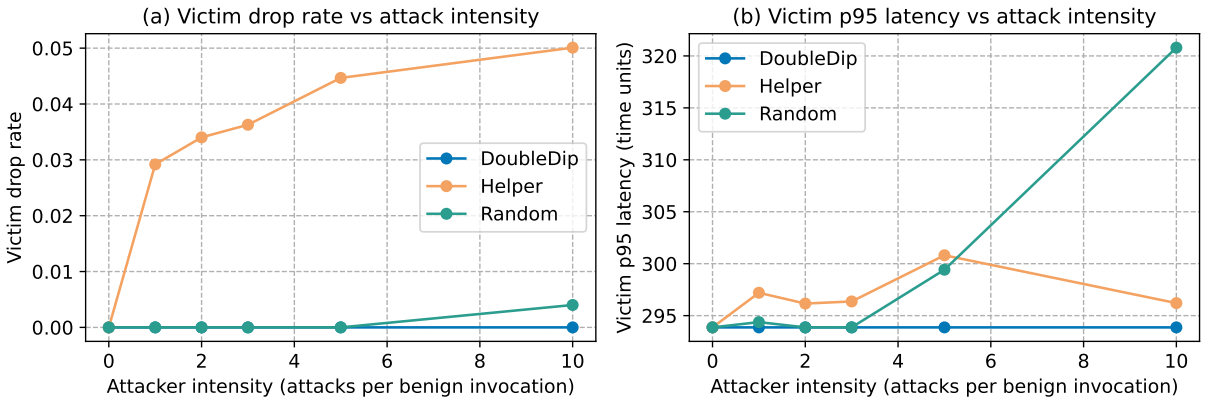

Fig. 3. Case Study B: Victim availability and tail latency under increasing attacker intensity.

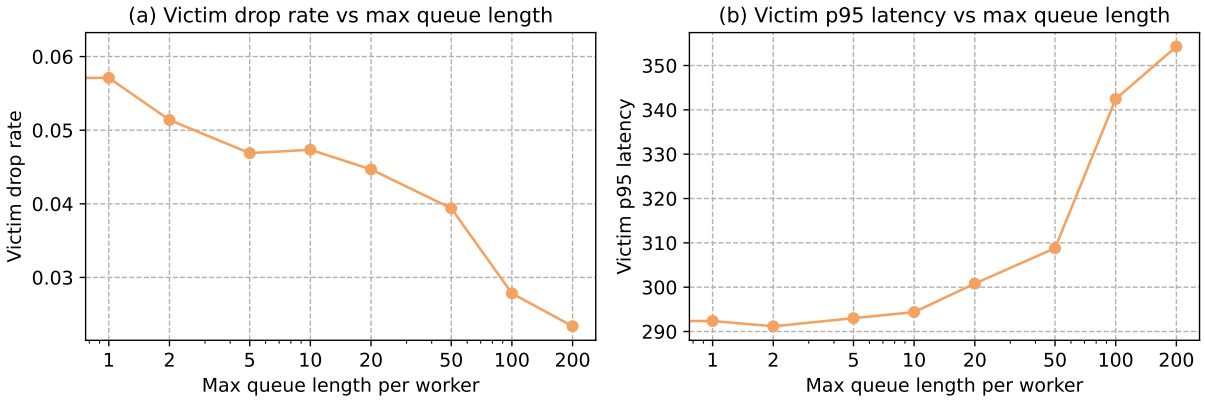

Fig. 4. Case Study B: Impact of per-worker queue length on availability and tail latency.

the tens to low hundreds), illustrating that modest queueing can reduce drops without severely harming p95, while aggressive queueing increasingly converts drops into long tail latency. By modeling queue limits and execution delays, Kumo enables systematic exploration of this tradeoff under adversarial load.

e) Effect of Cluster Scale: Figure 5 sweeps cluster size from 128 to 2048 workers under a fixed attacker intensity (5.0) and queue limit (50). Increasing the number of workers improves victim availability: drop rate decreases as additional capacity absorbs load, and tail latency trends downward overall. However, the improvement is not perfectly linear and exhibits diminishing returns, suggesting that beyond a certain point the remaining drops and tail latency are dominated by transient bursts, queueing effects, and stochastic imbalance rather than raw capacity alone. This behavior reflects a more realistic operating regime in which capacity expansion mitigates, but does not fully eliminate, contention-induced degradation. By allowing cluster size to be swept independently of workload and attacker behavior, Kumo enables precise analysis of how horizontal scaling shifts the availability–latency tradeoff under adversarial load.

This case study demonstrates that Kumo can model availability degradation driven by resource contention, queuing, and capacity limits, and can expose how design choices shape DoS behavior. In our scaled setting, DoS impact depends not only on attacker intensity and queue sizing, but also on scheduler behavior: some policies amplify hotspots and drops under overload, while others spread load and stabilize victim tail latency. Together with Case Study A, these results highlight why security analysis for serverless platforms must jointly consider scheduler policy, contention dynamics, and queueing limits.

C. Lessons Learned

Our evaluation highlights several lessons about analyzing security risks in serverless platforms.

First, *scheduler behavior can act as a security-relevant control surface*. Case Study A shows that different scheduling policies, when evaluated under identical workloads and platform configurations, can lead to substantial differences in attacker–victim co-location probability and time to first co-location. This demonstrates that placement decisions alone can meaningfully influence isolation risk, and that schedulers should be treated as configurable security mechanisms rather than fixed platform components.

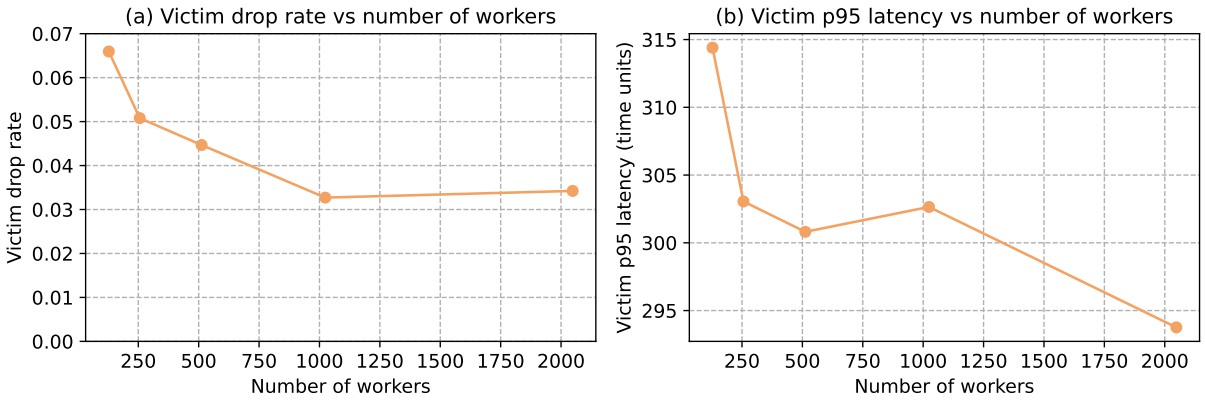

Fig. 5. Case Study B: Effect of cluster scale on victim availability and tail latency under fixed attacker intensity.

Second, *availability degradation emerges from the interaction between scheduling, queuing, and capacity*. As illustrated in Case Study B, denial-of-service behavior depends strongly on attacker intensity, service time, queue limits, and cluster scale. While schedulers may influence how contention manifests (e.g., through load spreading versus hotspot formation), availability outcomes are ultimately shaped by system-level resource dynamics once the platform approaches saturation.

Finally, *security and performance tradeoffs are inherently multi-dimensional*. Improving isolation may introduce moderate performance overheads, while mitigating availability attacks often shifts failures from drops to increased tail latency through deeper queuing or requires additional capacity through scaling. These tradeoffs cannot be captured by a single metric alone or configuration choice, underscoring the need for analysis tools that jointly model scheduler behavior, resource contention, and workload dynamics.

Together, these lessons reinforce the value of Kumo as a simulation framework for disentangling scheduler-driven security risks from broader resource exhaustion effects, enabling systematic exploration of security–performance tradeoffs in serverless platforms.

D. Simulation Performance

We report the execution time of our simulations to demonstrate the practicality of Kumo for large-scale security analysis. All experiments were run on a single commodity desktop machine, using a single core and no parallelization.

Case Study A consists of one experiment with 4 schedulers and 20 random seeds, corresponding to 80 simulation runs in total. Case Study B consists of three DoS experiments:

- **Attack intensity sweep (B1):** With 3 schedulers, 6 attacker intensities, and 20 random seeds, resulting in 360 runs.
- **Queue length sweep (B2):** With 9 per-worker queue length settings (including queue length $= 0$) and 20 random seeds, resulting in 180 runs.
- **Number of workers sweep (B3):** With 5 cluster sizes and 20 random seeds, resulting in 100 runs.

TABLE I
RUNTIME SUMMARY FOR CASE STUDIES A AND B.

Experiment	Total Time (s)	Total Runs	Time / Run (s)
Case Study A	394.1	80	4.93
Case Study B1	777.9	360	2.16
Case Study B2	353.8	180	1.97
Case Study B3	275.7	100	2.76

The runtime of each experiment is summarized in Table I. These results show that Kumo can simulate tens of thousands of invocations across hundreds of workers and tenants within minutes on a single machine. This performance makes Kumo suitable for iterative security analysis, large parameter sweeps, and comparative evaluation of scheduling and defense mechanisms.

V. DISCUSSION

A. Scheduler-Driven vs. System-Level Security

A key insight from our evaluation is the clear separation between scheduler-driven isolation risks and system-level availability vulnerabilities. Case Study A shows that scheduler choice is a first-order security mechanism for co-location attacks: under identical workloads and platform configurations, different schedulers lead to orders-of-magnitude differences in attacker–victim co-location probability and attack feasibility. In contrast, Case Study B demonstrates that DoS behavior is largely insensitive to scheduler design once resource contention dominates, and is instead governed by factors such as service time, queuing policy, and cluster capacity.

This distinction suggests that no single mechanism can address all serverless security risks. Scheduler design plays a critical role in mitigating isolation attacks, but availability attacks require system-level defenses that go beyond placement decisions. Kumo enables these distinctions to be studied explicitly within a unified framework.

B. Implications for Serverless Platform Design

Our results highlight several implications for serverless platform operators. First, schedulers should be viewed not only

as performance optimizers but also as security mechanisms. Policies that reduce co-location risk can substantially improve isolation with modest performance overhead. Second, defenses against DoS attacks cannot rely solely on scheduling. Queue management, admission control, and capacity provisioning all play a central role in determining availability under attack.

Importantly, several commonly used mitigation strategies involve tradeoffs. Increasing queue capacity reduces invocation drops but can severely degrade tail latency, while horizontal scaling improves availability at significant infrastructure cost. These tradeoffs underscore the need for principled analysis tools that expose the security consequences of design choices before deployment.

C. Limitations

Kumo is intentionally scoped to balance fidelity and tractability. It does not model low-level microarchitectural side channels, network-level DoS, or platform-specific implementation details such as proprietary placement heuristics. Instead, Kumo focuses on system-level behavior arising from shared execution environments, container reuse, and resource contention.

While this abstraction limits Kumo's ability to study certain classes of attacks, it enables scalable and reproducible analysis of dominant security mechanisms in serverless platforms. Future work could incorporate additional layers of fidelity where needed, without altering Kumo's core architecture.

D. Reproducibility and Availability

To support reproducibility and open-science practices, we release the complete Kumo artifact as open-source software. The artifact has been independently evaluated by the CCGRID Artifact Evaluation Committee and awarded the **Results Reproduced (ROR-R)** badge.

Additional details, including installation instructions and reproduction steps, are provided in Appendix.

E. Future Directions

Kumo opens several avenues for future research. One direction is integrating cost models to study the economic impact of security defenses, such as over-provisioning or aggressive isolation. Another is exploring scheduler synthesis, where security constraints are incorporated directly into scheduler design and evaluated using Kumo's simulation framework.

VI. Related Work

In contrast to prior work that focuses on either serverless performance optimization or empirical security attacks, Kumo bridges these domains by providing a security-focused simulation framework. We organize related work into four areas.

A. Serverless Platforms and Scheduling

Prior work has extensively studied the design and optimization of serverless platforms, focusing on scheduling, performance, and cost efficiency. Several studies propose scheduling policies to reduce cold starts, improve latency, or optimize resource utilization in serverless systems. Production platforms such as AWS Lambda, Azure Functions, and OpenWhisk employ sophisticated placement and reuse heuristics, though their exact implementations are largely opaque [23]–[25]. While these works provide important insights into serverless performance, they generally do not consider scheduler behavior as a security mechanism or analyze attacker-driven workloads.

B. Serverless Simulation and Modeling

A number of simulators and analytical models have been proposed for serverless computing, primarily targeting performance evaluation, cost modeling, and capacity planning [11]–[13]. These tools typically model invocation arrival processes, execution latency, and resource consumption, and are useful for understanding scalability and efficiency tradeoffs. However, existing simulators rarely include explicit attacker modeling, security-oriented metrics, or mechanisms to study isolation and availability under adversarial conditions. Kumo complements this line of work by prioritizing security analysis and explicitly modeling attackers, victims, and scheduler-induced isolation effects.

C. Co-location and Side-Channel Attacks in the Cloud

Co-location attacks have been widely studied in virtualized and multi-tenant cloud environments, where attackers attempt to place workloads on the same physical host as a victim to exploit side channels. Recent work has extended these attacks to serverless platforms, demonstrating that scheduler placement decisions and container reuse can enable information leakage even in short-lived execution environments [5], [16]. These studies typically rely on empirical experiments on production platforms or small-scale testbeds. Kumo provides a complementary approach by enabling controlled, reproducible exploration of co-location risk across different scheduling policies without requiring access to proprietary systems.

D. Denial-of-Service and Resource Exhaustion Attacks

DoS attacks through resource exhaustion have long been studied in cloud and distributed systems. Prior work has examined admission control, queuing policies, and over-provisioning as defenses against overload and performance collapse [26], [27]. In serverless settings, recent studies have highlighted how bursty workloads and unbounded scaling assumptions can lead to unexpected availability degradation [28], [29]. Unlike traditional network-level DDoS attacks, these behaviors arise from legitimate invocation patterns. Kumo models this class of attacks by explicitly capturing resource contention, queuing, and invocation drops, enabling systematic study of availability tradeoffs under adversarial load.

VII. Conclusion

In this paper, we presented Kumo, a security-focused simulator for serverless platforms that enables principled analysis of risks arising from scheduling and resource sharing decisions. Kumo is designed to support controlled, reproducible experiments with explicit modeling of workloads, schedulers,

attackers, and victims, bridging a gap between performance-oriented simulators and empirical security studies on production systems.

Through two complementary case studies, we demonstrated how Kumo can be used to analyze distinct classes of serverless security risks. Our co-location study showed that scheduler choice is a first-order security mechanism, with different policies inducing orders-of-magnitude differences in attacker–victim co-location probability and attack feasibility. In contrast, our DoS study revealed that availability degradation is largely governed by system-level factors such as resource contention, queuing behavior, and cluster capacity, and is comparatively insensitive to scheduler design once contention dominates. Together, these results highlight the importance of distinguishing scheduler-driven isolation risks from broader resource exhaustion vulnerabilities.

Kumo provides a flexible foundation for exploring these tradeoffs without requiring access to proprietary platforms or costly real-world experimentation. By enabling systematic, security-aware evaluation of serverless design choices, Kumo complements existing empirical work and opens new avenues for research on secure and resilient serverless computing.

VIII. Acknowledgment

The work in this paper is partially supported by the National Science Foundation (NSF) grants CNS-2155002 and 2311888.

References

[1] Y. Li, Y. Lin, Y. Wang, K. Ye, and C. Xu, "Serverless computing: state-of-the-art, challenges and opportunities," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1522–1539, 2022.

[2] H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless computing: a survey of opportunities, challenges, and applications," *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–32, 2022.

[3] E. Marin, D. Perino, and R. Di Pietro, "Serverless computing: a security perspective," *Journal of Cloud Computing*, vol. 11, no. 1, p. 69, 2022.

[4] M. Ghorbian, M. Ghobaei-Arani, and L. Esmaeili, "A survey on the scheduling mechanisms in serverless computing: a taxonomy, challenges, and trends," *Cluster Computing*, vol. 27, no. 5, pp. 5571–5610, 2024.

[5] C. Fang, H. Wang, N. Nazari, B. Omidi, A. Sasan, K. N. Khasawneh, S. Rafatirad, and H. Homayoun, "Repttack: Exploiting cloud schedulers to guide co-location attacks," in *Proceedings of the Network and Distributed Systems Security (NDSS) Symposium*, 2022.

[6] W. Shao, N. Nazari, B. Omidi, S. Rafatirad, H. Homayoun, K. N. Khasawneh, and C. Fang, "Bit of a close talker: A practical guide to serverless cloud co-location attacks," *arXiv preprint arXiv:2512.10361*, 2025.

[7] Z. N. Zhao, A. Morrison, C. W. Fletcher, and J. Torrellas, "Everywhere all at once: Co-location attacks on public cloud faas," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2024, pp. 133–149.

[8] N. Nazari, H. M. Makrani, C. Fang, B. Omidi, S. Rafatirad, H. Sayadi, K. N. Khasawneh, and H. Homayoun, "Adversarial attacks against machine learning-based resource provisioning systems," *IEEE Micro*, vol. 43, no. 5, pp. 35–44, 2023.

[9] C. Fang, N. Miao, H. Wang, J. Zhou, T. Sheaves, J. M. Emmert, A. Sasan, and H. Homayoun, "Gotcha! i know what you are doing on the fpga cloud: Fingerprinting co-located cloud fpga accelerators via measuring communication links," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2024–2037.

[10] C. Fang, N. Nazari, B. Omidi, H. Wang, A. Puri, M. Arora, S. Rafatirad, H. Homayoun, and K. Khasawneh, "Assessing and mitigating heterogeneity-driven security threats in the cloud," *ACM Transactions on Internet Technology*, vol. 25, no. 4, pp. 1–31, 2025.

[11] N. Mahmoudi and H. Khazaei, "Simfaas: A performance simulator for serverless computing platforms," *arXiv preprint arXiv:2102.08904*, 2021.

[12] P. Raith, T. Rausch, A. Furutanpey, and S. Dustdar, "faas-sim: A trace-driven simulation framework for serverless edge computing platforms," *Software: Practice and Experience*, vol. 53, no. 12, pp. 2327–2361, 2023.

[13] A. Mampage and R. Buyya, "Cloudsimsc: A toolkit for modeling and simulation of serverless computing environments," in *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. IEEE, 2023, pp. 550–557.

[14] V. Ishakian, R. Sweha, J. Londono, and A. Bestavros, "Colocation as a service: Strategic and operational services for cloud colocation," in *2010 Ninth IEEE International Symposium on Network Computing and Applications*. IEEE, 2010, pp. 76–83.

[15] Y. Azar, S. Kamara, I. Menache, M. Raykova, and B. Shepard, "Co-location-resistant clouds," in *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, 2014, pp. 9–20.

[16] C. Fang, N. Nazari, B. Omidi, H. Wang, A. Puri, M. Arora, S. Rafatirad, H. Homayoun, and K. N. Khasawneh, "Heteroscore: Evaluating and mitigating cloud security threats brought by heterogeneity," in *The Network and Distributed System Security Symposium (NDSS)*, 2023.

[17] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: A fast and stealthy cache attack," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 279–299.

[18] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," *Communications of the ACM*, vol. 63, no. 7, pp. 93–101, 2020.

[19] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom *et al.*, "Meltdown: Reading kernel memory from user space," *Communications of the ACM*, vol. 63, no. 6, pp. 46–56, 2020.

[20] Y. Yarom and K. Falkner, "{FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack," in *23rd USENIX security symposium (USENIX security 14)*, 2014, pp. 719–732.

[21] M. Nabi, M. Toeroe, and F. Khendek, "Availability in the cloud: State of the art," *Journal of Network and Computer Applications*, vol. 60, pp. 54–67, 2016.

[22] A. M. Sharifi, S. K. Amirgholipour, M. Alirezanejad, B. S. Aski, and M. Ghiami, "Availability challenge of cloud system under ddos attack," *Indian Journal of Science and Technology*, vol. 5, no. 6, pp. 2933–7, 2012.

[23] OpenWhisk, "OpenWhisk Documentation," https://openwhisk.apache.org/documentation.html, [Online; accessed Jun. 2024].

[24] Amazon, "AWS Lambda," https://aws.amazon.com/lambda/, [Online; accessed Jun. 2024].

[25] M. Azure, "Microsoft Azure," https://azure.microsoft.com/en-us/, 2022, [Online; accessed Apr. 2022].

[26] I. Cho, A. Saeed, J. Fried, S. J. Park, M. Alizadeh, and A. Belay, "Overload control for {μs-scale}{RPCs} with breakwater," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 299–314.

[27] J. Park, J. Park, Y. Jung, H. Lim, H. Yeo, and D. Han, "Topfull: An adaptive top-down overload control for slo-oriented microservices," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, pp. 876–890.

[28] A. Suresh, G. Somashekar, A. Varadarajan, V. R. Kakarla, H. Upadhyay, and A. Gandhi, "Ensure: Efficient scheduling and autonomous resource management in serverless environments," in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 2020, pp. 1–10.

[29] H. Qiu, S. Jha, S. S. Banerjee, A. Patke, C. Wang, F. Hubertus, Z. T. Kalbarczyk, and R. K. Iyer, "Is function-as-a-service a good fit for latency-critical services?" in *Proceedings of the Seventh International Workshop on Serverless Computing (WoSC7) 2021*, 2021, pp. 1–8.

APPENDIX

A. Artifact Description

1) Overview: This artifact contains the complete implementation of *Kumo*, a discrete-event simulator for analyzing security risks in serverless platforms, along with all configuration files and plotting scripts required to reproduce the results presented in the paper.

The artifact enables reproduction of:

- Section IV-A: Case Study A (Attacker–Victim Co-location)
- Section IV-B: Case Study B (Denial-of-Service Experiments)
- Figures 2–5
- Table I (runtime measurements)

The archived artifact is available at:

- DOI: https://doi.org/10.5281/zenodo.18635971
- Development repository: https://github.com/YoungKameSennin/kumo

2) Artifact Contents: The artifact includes:

- `src/` — C++17 implementation of the Kumo simulator
- `configs/` — Experiment configurations for Case Studies A, B1, B2, and B3
- `plot/` — Python scripts for generating all paper figures
- `Makefile` — Build system for the simulator
- `artifacts_reproduce.sh` — One-command reproduction script
- `requirements.txt` — Python plotting dependencies

The simulator is implemented in C++17 and uses Python for data analysis and plotting.

3) System Requirements: The artifact runs on commodity hardware.

Hardware requirements

- Standard laptop or desktop
- Single CPU core sufficient
- No GPU required

Software requirements

- Linux or macOS
- `g++` with C++17 support (GCC $\geq$ 7 recommended)
- Python 3.9+
- Python packages: `numpy`, `pandas`, `matplotlib`

Install Python dependencies:

```
pip install -r requirements.txt
```

4) Build Instructions: Compile the simulator:

```
make
```

This produces the binary `kumo_experiment`.

5) Reproducing Experimental Results: To reproduce all results:

```
make reproduce
```

The reproduction script performs the following steps:

1) Builds the simulator if necessary.
2) Executes all experiment configurations in `configs/`.
3) Generates result CSV files in `results/`.
4) Generates figures in `figs/`.
5) Performs sanity checks to confirm expected outputs exist.

6) Expected Outputs: After successful reproduction:

- CSV result files:
  - `case_study_A_result.csv`
  - `case_study_B1_result.csv`
  - `case_study_B2_result.csv`
  - `case_study_B3_result.csv`
- Figures corresponding to:
  - Figure 2 (Co-location analysis)
  - Figure 3 (Attack intensity sweep)
  - Figure 4 (Queue length sweep)
  - Figure 5 (Cluster size sweep)

7) Estimated Runtime: On a commodity desktop:

- Build time: $<$ 10 seconds
- Case Study A: $\sim$ 5 seconds per run
- Case Study B experiments: $\sim$ 2–3 seconds per run
- Full reproduction: typically 30–60 minutes

These runtimes are consistent with Table I in the paper.

8) Mapping to Paper Results:

Paper Result	Artifact Component
Figure 2(a–d)	`configs/case_study_A.cfg`
Figure 3	`configs/case_study_B1.cfg`
Figure 4	`configs/case_study_B2.cfg`
Figure 5	`configs/case_study_B3.cfg`
Table I	Execution logs and runtime measurements

9) Verification Guidelines: Readers may verify the following behaviors:

- The DoubleDip scheduler exhibits near-zero co-location probability.
- Increasing attacker intensity increases victim drop rate.
- Larger queue limits reduce drops but increase tail latency.
- Increasing cluster size reduces drop rate and latency.

Exact numerical equality is not required; reproduction focuses on behavioral consistency and trends.

10) Reproduction Steps: The artifact includes fixed configuration files and deterministic random seeds to ensure reproducible execution.

To reproduce the results reported in the paper:

1) Build the simulator:
   ```
   make
   ```
2) Install Python dependencies:
   ```
   pip install -r requirements.txt
   ```
3) Execute all experiments and generate plots:
   ```
   make reproduce
   ```
4) After completion, CSV outputs will appear in `results/` and generated figures will appear in `figs/`.
5) Compare the generated figures with Figures 2–5 in the paper to verify behavioral consistency.

11) License: The artifact is released under the MIT License.

12) Summary: This artifact provides a complete, self-contained implementation of Kumo and enables reproduction of all experimental results using commodity hardware. The artifact has been independently verified by the CCGRID Artifact Evaluation Committee and awarded the **Results Reproduced (ROR-R)** badge.