From Co-location to Exfiltration: Practical Cache Side-Channel Attacks in the Modern Public Cloud

Zirui Neil Zhao[1], Adam Morrison[2], Christopher W. Fletcher[3], Josep Torrellas[4]
[1]NVIDIA/UT Austin (Work done at UIUC), [2]Tel Aviv University, [3]UC Berkeley, [4]UIUC

Abstract

Sharing resources among tenants is fundamental to public clouds, enhancing efficiency but also creating opportunities for microarchitectural side-channel attacks. However, cloud vendors remain skeptical about the practicality of these attacks, particularly regarding the ability to co-locate attacker and victm, and to overcome system noise. In this work, we develop a series of techniques for each step of the attack and, for the first time, demonstrate cross-tenant information leakage on the public Google Cloud Run, refuting the belief that such attacks are impractical. Our findings highlight the need to secure public clouds against side-channel attacks.

1 Introduction

Microarchitectural side-channel attacks are well-known threats capable of leaking sensitive user information like cryptographic keys [5, 9]. These attacks exploit a victim's secret-dependent usage of *shared* microarchitectural resources, such as caches, enabling an attacker to infer secret data. Such vulnerabilities pose significant risks to modern public clouds, which increasingly rely on extensive resource sharing among mutually distrusting tenants for efficiency, as seen in emerging computing paradigms like Function-as-a-Service (FaaS).

Despite the potency of microarchitectural side channels, cloud vendors remain skeptical about their practicality "in the wild". For example, Amazon's white paper on Elastic Compute Cloud (EC2) security design [1] explicitly dismisses last-level cache (LLC) Prime+Probe attacks [5]—a powerful class of cache-based side channels—as impractical due to noise in cloud systems. While other vendors, such as Google Cloud, do not disclose their stance on the threat, we empirically find no mitigations in their systems, suggesting they share a view similar to Amazon's.

Indeed, executing side-channel attacks in public clouds requires overcoming many practical challenges at each attack step. The first step in most attacks is to *co-locate* the attacker's processes with the victim's on the same physical host [7]. In modern cloud environments like FaaS, co-location is particularly challenging since container instance placement is fully managed by proprietary policies unknown to the public. As a result, an attacker who naively launches containers without knowledge of these policies is unlikely to achieve co-location. Moreover, the vast scale of modern data centers and the dynamism introduced by the frequent launching and termination of containers in FaaS environments further complicate co-location.

The second step after co-location is to prepare and monitor a chosen side channel to *exfiltrate* information from the victim. Unlike co-location, exfiltration depends heavily on the specific side channel used. The primary challenge in this step, as noted by Amazon, is the noise created by other tenants' activities, which interferes with both the preparation and monitoring of the side channel. Additionally, FaaS environments are highly dynamic, with user workloads typically lasting on a host only a few to tens of minutes. As a result, the attacker has a *limited time window* to complete exfiltration while co-located with the victim. Finally, there are channel-specific challenges, such as the lack of huge pages in some containerized environments and the widespread adoption of non-inclusive LLCs, both of which hinder LLC Prime+Probe attacks.

In this article, we refute the belief that side-channel attacks are impractical in public clouds. Specifically, we present the first demonstration of cross-tenant information leakage in Google Cloud Run [2], a production Function-as-a-Service (FaaS) platform, using LLC Prime+Probe as a case study. Notably, every step of the attack requires novel techniques to overcome the practical challenges of the cloud environment.

For the co-location step, we present the first comprehensive study of co-location techniques in modern public FaaS environments. To help reverse-engineer their proprietary policies for instance placement, we develop two novel *host fingerprinting* techniques. We show that, despite the use of sandboxing and virtualization technologies in the cloud, an attacker can still learn identifying host information by directly interacting with the host hardware—specifically, the timestamp counter.

Armed with host fingerprints, we perform a large-scale study on Google Cloud Run to analyze its instance placement policies. We uncover an exploitable instance placement behavior in Cloud Run and then develop an instance launching strategy that exploits this behavior. Our strategy can spread attacker instances across a large number of Cloud Run hosts within a data center—drastically increasing co-location probability and reducing the financial cost of the attack.

For the exfiltration step with LLC Prime+Probe, we characterize the noise resilience of the state-of-the-art Prime+Probe attack techniques, which are mostly developed for and evaluated in quiescent local environments. We show that these techniques *fail in the noisy cloud*, and examine the root causes of failures. Based on the insights from the characterization, we develop new techniques for every step of LLC Prime+Probe to solve the practical challenges posed by the cloud.

Note that while our attack targets Google Cloud Run, the techniques we develop are applicable to any modern server. Therefore, we believe that multi-tenant cloud products from other vendors, such as Amazon Web Services (AWS) and Azure, are also susceptible to our attack techniques.

In the remainder of this article, we outline our new attack techniques for each step, with a stronger emphasis on co-location, as it is often overlooked. Full details of these techniques are available in our ASPLOS 2024 papers [10, 11]. We have open-sourced our implementations at `https://github.com/zzrcxb/EAA0` for the co-location step and `https://github.com/zzrcxb/LLCFeasible` for exfiltration using LLC Prime+Probe.

Industry Responses. After we reported our findings, Google Cloud acknowledged the threat and internally issued a critical-level bug report to their product team. AWS also revised their

stance on LLC side-channel threats by updating the EC2 Security Design white paper in February 2024. Our work has also drawn significant attention from hardware vendors such as Intel, AMD, and Micron, as fully mitigating these attacks likely requires hardware changes.

2 Background

2.1 Function-as-a-Service and Google Cloud Run

Function-as-a-Service (FaaS) is an emerging cloud computing paradigm that disaggregates a large monolithic application into many small stand-alone web services (also known as functions). These services collaborate and communicate with each other over the network to perform the task of the original monolithic application. To simplify service deployment and management, each service is packed with its software dependencies into a lightweight, self-contained container image.

The FaaS platform orchestrator fully manages services at a container-level granularity. When a service is invoked by a user or another service, the orchestrator launches a new container instance of the requested service to process the incoming request. The selection of the physical host on which to place this new instance is based on proprietary policies unknown to the user. After serving the request, the instance enters an idle state, releasing its CPU, and awaiting further incoming requests. FaaS platforms dynamically adapt the number of container instances to the service's demand, a feature known as *autoscaling*.

In this article, we target Cloud Run [2], which is a FaaS platform from Google Cloud. Cloud Run offers two types of execution environments to sandbox untrusted user containers for software security. In its first-generation environment (Gen 1), Cloud Run uses gVisor [4] to sandbox Linux containers *without* hardware virtualization by intercepting and emulating Linux system calls. Consequently, the user application cannot access sensitive host information. For example, gVisor conceals host-identifying information like the host IP address.

In the second-generation environment (Gen 2), Cloud Run uses lightweight virtual machines (VMs) for sandboxing. Using hardware virtualization, the hypervisor can trap and emulate certain x86 instructions like `cpuid`, hiding sensitive host information. Compared to Gen 1, Gen 2 offers full Linux compatibility, but has a larger resource footprint and thus incurs a longer start-up latency. Therefore, at the time of this writing (March 2025), Cloud Run uses Gen 1 for services and Gen 2 for batch workloads by default. Since we target user-facing services, this article focuses on Gen 1. Our ASPLOS 2024 paper [10] discusses the transferability of our results to Gen 2.

2.2 Co-location and LLC Prime+Probe Attack in the Cloud

Side-channel attacks in the cloud start with the attacker co-locating their container with the target victim container on the same physical machine (Step 1). Then, the attacker sets up the side channel for exfiltration (Step 2). In this article, we demonstrate both steps on Google Cloud Run and use LLC Prime+Probe attack for Step 2.

Step 2 has three sub-steps. First, the attacker prepares LLC channels by constructing LLC eviction sets (Step 2.1). An *eviction set* for a specific LLC set is a collection of addresses that, once accessed, can evict any cache line mapped to that LLC set [5]. Using an eviction set for an LLC set s, the attacker can monitor victim memory accesses to s with Prime+Probe. The high-level idea is that the attacker fills set s with lines from the eviction set (i.e., *prime*), waits for the victim execution, and then reloads every line from the eviction set (i.e., *probe*). If the victim accessed set s, at least one of the attacker lines will be evicted from the cache, which will be observed by the attacker due to the high probe latency.

In practice, the victim program makes secret-dependent accesses to only a few LLC sets. We refer to these LLC sets as the *target LLC sets*, which are of the attacker's interest but unknown to the attacker. Hence, in Step 2.1, the attacker needs to build up to *tens of thousands* of eviction sets, each corresponding to a set in the distributed LLC [5]. Then, the attacker scans through each LLC set and identifies the target LLC sets (Step 2.2). Finally, the attacker monitors the target LLC sets with Prime+Probe and exfiltrates the secret (Step 2.3). The complete attack steps are summarized in Table 1.

Table 1: Steps of an LLC Prime+Probe attack in the cloud.

Step	Description
Step 1. Co-location	Co-locate the attacker container on the same physical host as the target victim container
Step 2.1. Prepare LLC side channels	Construct numerous LLC eviction sets, each corresponding to a set in the distributed LLC
Step 2.2. Identify target LLC sets	Scan LLC sets to identify those that the victim accesses in a secret-dependent manner
Step 2.3. Exfiltrate information	Monitor the target LLC sets and exfiltrate information

2.3 Eviction Set Construction Algorithms

Constructing LLC eviction sets is pivotal to Step 2.1. An eviction set for a cache set s with W ways consists of W *congruent addresses* that are mapped to s. Constructing an eviction set generally consists of two steps [5]: (1) Create a *candidate set* that contains a sufficiently large number of *candidate addresses*, of which at least W are mapped to s. (2) Prune the candidate set into an eviction set, using state-of-the-art algorithms such as group testing [8] or Prime+Scope [6]. In our work, we use a candidate set comprising almost 30,000 addresses for the processor used by Cloud Run (i.e., Intel Skylake-SP).

3 Threat Model

We consider an attacker who aims to exfiltrate sensitive information from a victim service running on a public FaaS platform such as Google Cloud Run [2] through microarchitectural side channels like LLC Prime+Probe [5]. We assume the attacker is an unprivileged user of the FaaS platform. Therefore, the attacker's interaction with the platform is limited to the standard interfaces that are available to all platform users. Using these interfaces, the attacker can launch many attacker containers through autoscaling and then run arbitrary programs *inside* the containers. We also assume that the attacker can trigger the victim's execution by sending requests to the victim service, either directly or through interactions with a public web application that the victim service is part of. Finally, we assume that the attacker has access to the victim's executable that is vulnerable to side channels. This assumption generally holds because developers often use programs and libraries whose source code or executables are publicly released.

4 Host Fingerprinting for Co-location

Public FaaS platforms, such as Google Cloud Run, do not reveal their container instance placement policies. It can be shown that it is difficult to achieve co-location without any insight into these policies [10]. Therefore, we introduce two physical host fingerprinting techniques for reverse engineering how instances are placed in Cloud Run's Gen 1 and Gen 2 environments, respectively. Our key insight is that, despite gVisor's sandboxing or hardware virtualization, the attacker can still learn host-identifying information by directly interacting with the underlying host hardware. Note that our two techniques are not specific to Cloud Run and thus can be adapted to other FaaS platforms. Since this article focuses on Gen 1, which is the default environment for services, we detail our fingerprinting technique and results for Gen 1, and refer interested readers to our ASPLOS 2024 paper [10] for fingerprinting in Gen 2.

4.1 Fingerprinting the Gen 1 Environment

In the Gen 1 environment, gVisor sandboxes user programs and hides the host information, thereby blocking host fingerprinting through IP addresses or statistics in the `/proc` filesystem. However, we find that we can bypass gVisor's software countermeasures to learn sensitive host information by directly interacting with the non-virtualized host hardware.

For example, the attacker can use the unprivileged `cpuid` instruction to extract information like the CPU model and cache hierarchy structure, which are essential for many cache-based side-channel attacks [5]. Similarly, the attacker can use the unprivileged `rdtsc` and `rdtscp` instructions to read the host's timestamp counter (TSC).

On x86, the TSC is reset to 0 when the host boots up and then increments at a known constant rate f, irrespective of CPU frequency scaling. Hence, the attacker can simultaneously read the current real-world time T_w and the value of the timestamp counter tsc, and then derive the host's boot time $T_{boot} = T_w - tsc/f$. Since hosts very likely have different boot times due to system maintenance, hardware failures, and power conservation measures, we use T_{boot} as the host fingerprint in the Gen 1 environment. Note that the derived T_{boot} on a given host can exhibit small variations across measurements due to noise. Consequently, we round T_{boot} to a certain precision p_{boot} (e.g., 1 s). With the rounded value, measurements of the same host consistently produce the same fingerprint.

4.2 Verifying Instance Co-Location

To evaluate the accuracy of our fingerprints, we leverage a covert channel [3] based on the hardware random number generator to obtain the co-location ground truth. Specifically, if a pair of containers can establish reliable communication over the covert channel, then they are co-located. However, such a naive approach is not scalable, as it only tests whether a *pair* of containers are co-located. Obtaining the co-location ground truth of N containers would require $O(N^2)$ pairwise tests, where N is usually in the hundreds to thousands.

To address this challenge, we develop a scalable fingerprint-assisted co-location verification methodology that requires $O(M)$ tests in the best scenario, where M is the number of hosts occupied by the N containers and $M \leq N$. Using our co-location verification methodology, we evaluated the fingerprint accuracy in 15 experiments across three different Cloud Run datacenters. The fingerprint accuracy is measured by the Fowlkes-Mallows Index (FMI), a common metric for clustering performance. FMI ranges from 0 to 1, with 1 representing the perfect accuracy. Averaged across the 15 measurements, the fingerprints have an FMI of 0.9999 in the Gen 1 environment.

4.3 Understanding Instance Placement Policy

Leveraging our host fingerprinting techniques, we perform a set of experiments to study the instance placement policy of Cloud Run. In our experiments, we vary the Google account that launches the containers, the number of container launches, the time interval between container launches, and more. We summarize our findings as follows.

Observation 1. Cloud Run prefers to run container instances owned by the same account on a specific subset of hosts. We refer to these preferred hosts as the *base hosts*. This behavior can be explained by affinity scheduling. Affinity scheduling aims to reduce communication overhead by co-locating instances that frequently interact with each other, which is a likely scenario for services originating from the same account. Each account is assigned tens of base hosts, though the exact number varies across data centers.

Observation 2. Cloud Run often uses different base hosts for different accounts. Therefore, if the attacker and victim accounts have different base hosts, which is common, the attacker containers are very unlikely to co-locate with the victim's.

Observation 3. When a service has a high demand within the past 30 minutes, Cloud Run appears to use a load balancer that places container instances of that service onto extra hosts that are not base hosts. We refer to these extra hosts as the *helper hosts*.

This observation is critical to developing an effective adversarial container launching strategy to maximize the chance of co-location. For example, the attacker can artificially increase their services' demand and trick Cloud Run into spreading attacker container instances across many physical hosts that are beyond their base hosts. This helps the attacker increase the chance of co-locating with the victim.

Finally, we observed that different Cloud Run data centers and execution environments (i.e., Gen 1 and Gen 2 environments) have a similar instance placement policy.

4.4 Efficient Co-location in Google Cloud Run

We evaluate two instance launching strategies for co-location with victims. Our metric is the *victim instance coverage*, which is the percentage of victim instances that are co-located with the attacker. Our experiments involve three Google accounts: Account 1 as the attacker account, and Account 2 and Account 3 as victim accounts. Similar to the fingerprint accuracy evaluation, the co-location evaluation is repeated across three data centers: *us-east1*, *us-central1*, and *us-west1*. For each combination of data center and victim account, we repeat the measurement three times on different days and at different times of day.

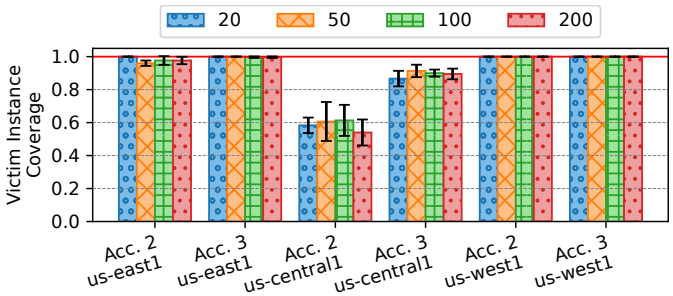
In each experiment, we set the number of victim instances to 20, 50, 100, and 200. Since Cloud Run allows a maximum of 100 instances per service by default, the 100-instance configuration is our default. We also use four different victim instance sizes: Pico, Small, Medium, and Large. They correspond to instance requests of 0.25, 1, 2, and 4 CPUs, respectively, and 256 MB, 512 MB, 1 GB, and 2 GB of memory, respectively. We choose Small as the default victim size since it is the default configuration for Cloud Run services. We also fix the attacker

instance size to SMALL.

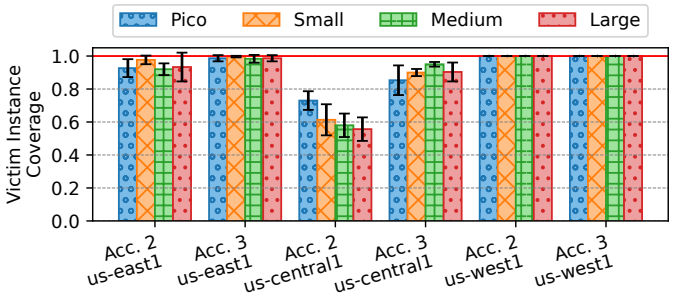
Strategy 1: Naive Instance Launching. Here, the attacker simply launches numerous instances without our insights into the Cloud Run's instance placement behavior. In our experiment, this naive strategy launches 4,800 instances from six services.

Despite the large number of attacker instances, we observe *zero* co-location with ACCOUNT 2 in *us-east1* and *us-central1*, or with ACCOUNT 3 in *us-east1* and *us-west1*. We see a high average victim instance coverage only with ACCOUNT 2 in *us-west1* (100.0%) and with ACCOUNT 3 in *us-central1* (81.0%), as the base hosts of the attacker and victim happen to be highly overlapped in the corresponding data centers. Changing the number of victim instances or their size does not yield significant variations. Overall, the data indicates that a naive launching strategy without any insight into Cloud Run's placement behavior is often ineffective.

Strategy 2: Optimized Instance Launching. This strategy exploits the load-balancing behavior of Cloud Run (Observation 3). The high-level idea is to prime the attacker service into a high-demand state by repeatedly launching many instances. This action enables the attacker to deploy instances onto numerous helper hosts. Similar to Strategy 1, the attacker launches 4,800 instances from six services.



(a) Varying the number of victim instances (20, 50, 100, and 200). The victim instance size is fixed to SMALL.



(b) Varying the victim instance size (PICO, SMALL, MEDIUM, and LARGE). The number of victim instances is fixed to 100.

Figure 1: Average victim instance coverage of our optimized launching strategy in different data centers. Error bars represent standard deviations. "Acc." is an abbreviation for "Account".

Figure 1a illustrates that our optimized instance launching strategy is highly effective. With the default configuration of 100 SMALL victim instances, we observe high victim instance coverage. From left to right, we see, in *us-east1*, victim instance coverages of 97.7% and 99.7% with ACCOUNT 2 and ACCOUNT 3, respectively. In *us-central1*, we see lower coverages of 61.3% with ACCOUNT 2 and 90.0% with ACCOUNT 3. The lowered coverages are likely due to the vast size of *us-central1*. Finally, in *us-west1*, we see 100.0% coverage with both ACCOUNT 2 and ACCOUNT 3. Figure 1b shows a similar behavior, as we vary the victim size while keeping the number of victim instances set to 100. On average, the estimated costs of the attack are only 24 USD, 23 USD, and 27 USD in *us-east1*, *us-central1*, and *us-west1*, respectively.

Overall, our evaluation shows that a launching strategy optimized for the instance placement policy can drastically outperform a naive approach. The reverse engineering of the instance placement policy is made possible only through our accurate physical host fingerprinting techniques.

5 High-Speed Noise-Resilient LLC Prime+Probe in the Cloud

The previously discussed techniques enable the attacker container to co-locate with the victim container. We now demonstrate, for the first time, cross-tenant information leakage using LLC Prime+Probe on the public Google Cloud Run. Notably, every step of LLC Prime+Probe (STEPS 2.1–2.3 in Table 1) requires novel techniques to overcome practical challenges in the cloud, such as noise from other tenants and short execution time limits. In this section, we outline the insights gained and the techniques developed to perform LLC Prime+Probe in the cloud. For full details, refer to our ASPLOS 2024 paper [11].

New Observation: Existing Prime+Probe Approaches Fail in the Cloud. We show that STEPS 2.1–2.3 of Prime+Probe in Table 1 are challenging to support in the Cloud Run environment. In particular, we find that state-of-the-art candidate pruning algorithms, such as group testing [8] and Prime+Scope [6], have a low success rate of constructing eviction sets in Cloud Run. Due to the noise present in the cloud, they take 10–24× longer than when operating in a quiescent local setting. Since the attacker needs to construct eviction sets for up to tens of thousands of LLC sets within a limited time window, this low performance makes existing pruning algorithms unsuitable for the public cloud.

We discover that Prime+Scope, which is also the state-of-the-art cache monitoring technique used in STEPS 2.2–2.3, is ineffective on Cloud Run. Specifically, Prime+Scope requires an average time of 6,024 cycles to reset the cache set after detecting each access to the set. During this time, the attacker will miss additional accesses to the set. Since we observe that the LLC is accessed very frequently in production Cloud Run environments, Prime+Scope's long reset latency will result in many victim accesses being undetected, leading to a poor-quality trace.

New Techniques: Effective Construction of Eviction Sets in the Cloud (STEP 2.1). To speed up eviction set construction in STEP 2.1, we introduce: (1) a generic optimization named *L2-driven candidate address filtering* that is applicable to all candidate pruning algorithms, and (2) a new *binary search-based* pruning algorithm. To explain them, we consider constructing an eviction set for set s where an address T_a maps.

L2-driven candidate address filtering is based on the observation that the physical address bits used to index into L2 sets are typically a subset of those used to index into LLC sets. Hence, if addresses A and B do not conflict in the L2, then they have different L2 and LLC set index bits and, therefore, they must *not* conflict in the LLC. Based on this observation, we first construct an L2 eviction set for T_a, a process that is

minimally impacted by the background noise because the L2 is private to the core. Then, for each address a in the candidate set, we use the L2 eviction set to test whether a conflicts with T_a in the L2 by checking if a can be evicted by the L2 eviction set; if not, a is removed from the candidate set. This technique can shrink the candidate set to 1/16 of its original size, and speed-up the subsequent pruning process.

We now briefly overview the design of the new binary search-based pruning algorithm. Given a list of candidate addresses, we test whether the first n addresses can evict a target address T_a. For a W-way cache, increasing n from zero will result in a negative test outcome until the first n addresses include W congruent addresses. We define the *tipping point*, denoted by τ, as the smallest n for which the first n addresses evict T_a. Therefore, τ is the index of the W-th congruent address in the list, assuming that the indexation begins from 1. For a given n, if the first n addresses evict T_a, it means that $n \geq \tau$; otherwise, $n < \tau$. Our main idea is *to use binary search* to efficiently determine τ and thus identify one congruent address. Then, we exclude the congruent address from any future search, and repeat the binary search process until W different congruent addresses are found. As detailed in our paper [11], this new binary search-based pruning algorithm outperforms existing algorithms on caches with a high associativity, which is common for the LLC and L2 of server processors.

By combining these two techniques, it now takes only 2.4 minutes on average to construct eviction sets for all the 57,344 LLC sets of an Intel Skylake-SP machine in Cloud Run, with a median success rate of 99.1%. In contrast, using the well-optimized state-of-the-art algorithms, this process is expected to take at least 14.6 hours, far exceeding the execution time limit of individual Cloud Run instances.

New Techniques: Noise-resilient Target Set Identification and Victim Monitoring (Steps 2.2–2.3). We also develop two novel techniques for Steps 2.2–2.3. First, we introduce *Parallel Probing*, which enables the monitoring of victim memory accesses with high time resolution. This technique optimizes Step 2.2 (identify target sets) and Step 2.3 (exfiltrate information) for the noisy cloud environment. Second, we leverage *Power Spectral Density* from signal processing to easily identify the victim's target cache set. This technique optimizes Step 2.2.

Parallel probing explores an overlooked trade-off between the prime latency and the probe latency. A short probe latency enables the attacker to monitor when accesses occur at a high time resolution. A short prime latency allows the attacker to quickly prepare the monitored cache set for detecting the next access. During priming, the state-of-the-art work Prime+Scope carefully prepares the cache replacement states so that probe latency is small (~94 cycles). However, the prime latency is high (~6,024 cycles), leading to missing many victim accesses on Cloud Run.

We discover that, thanks to the high memory-level parallelism supported by modern processors, we can simply probe *with overlapped accesses* all the W lines of a minimal eviction set, *without requiring careful preparation of the cache replacement states during priming*. As a result, the prime latency is only ~1,121 cycles, and the probe latency is ~118 cycles—only slightly higher than in Prime+Scope. When a cache set is frequently accessed by the victim at a short time interval of 2,000 cycles, parallel probing can detect 84.1% of the victim accesses in the noisy Cloud Run, while Prime+Scope has an average detection rate of only 15.4%.

To identify whether a cache set is a target cache set (Step 2.2), our insight is that the accesses of a victim program to the target cache set are often periodic, in a way that the attacker expects. Therefore, we propose to process the access traces *in the frequency domain*, where it is easy to spot periodic patterns. Specifically, we estimate the Power Spectral Density (PSD) of a memory access trace, which measures the "strength" of the signal at different frequencies. If the access trace is collected from the target set where the victim makes periodic accesses, we will observe peaks in the trace's PSD around the expected victim-access frequencies. Applying the PSD method on access traces collected from different LLC sets, the attacker can identify the target LLC set in 6.1 s, with an average success rate of 94.1%.

6 Cross-Tenant Information Leakage in the Production Google Cloud

Putting all the pieces together, our ASPLOS papers [10, 11] demonstrate end-to-end, *cross-tenant* nonce extractions from a vulnerable ECDSA implementation on Cloud Run. The attacker first co-locates their container with the victim container. Then, the attacker builds the eviction sets and finds the target set using the PSD method, while sending requests to trigger victim executions. Once the target set is identified, the attacker triggers the victim execution 10 more times to steal the different nonces used in each execution. Among 52 pairs of co-located containers, the attacker sees a signal in 47 of them. From the traces collected from these 47 victims, we extract a median of 81% of the nonce bits, with an average bit error rate of 3%.

7 Closing Thoughts

Our work is the first to demonstrate cross-tenant information leakage using LLC Prime+Probe in the public Google Cloud Run, refuting the common belief that such attacks are impractical in a noisy cloud environment. We systematically studied and improved each attack step—from attacker-victim co-location to information exfiltration via LLC Prime+Probe in a noisy environment—leading to numerous new attack techniques. Given the relatively low cost of our attack, securing the public cloud against side-channel attacks is imperative. For example, cloud vendors can develop new container instance placement policies that increase the difficulty of co-location. Another promising research direction is partitioning microarchitectural resources shared among tenants, as this not only provides strong isolation but also improves quality of service.

8 Acknowledgements

This research was funded in part by an Intel Resilient Architectures and Robust Electronics (RARE) gift; by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA; and by NSF grants 1942888, 1954521, and 1956007.

9 Authors' Short Bios

ZIRUI NEIL ZHAO is an Assistant Professor at the ECE department of the University of Texas at Austin. Prior to this, he was a postdoctoral researcher at NVIDIA and received a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign, where this work was done. His research interests lie in computer architecture, system security, and

cloud computing. Contact him at neil.zhao@utexas.edu.

ADAM MORRISON is an Associate Professor at the School of Computer Science, Tel Aviv University. His research is on the performance and security of multi-core computing (software and hardware), from algorithms through operating systems to microarchitecture. He received a Ph.D. in Computer Science from Tel Aviv University. Contact him at mad@cs.tau.ac.il.

CHRISTOPHER W. FLETCHER is an Associate Professor in Electrical Engineering and Computer Science at the University of California, Berkeley. His research interests include secure architecture, application-specific acceleration, computer architecture/systems and applied cryptography. He received a Ph.D. in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. Contact him at cwfletcher@berkeley.edu.

JOSEP TORRELLAS is the Saburo Muroga Professor of Computer Science at the University of Illinois at Urbana-Champaign, and director of the Semiconductor Research Corporation/DARPA JUMP 2.0 ACE Center for Evolvable Computing. His research interests include parallel computer architectures for performance, energy efficiency, programmability, and security. He received a Ph.D. in Electrical Engineering from Stanford University. Contact him at torrella@illinois.edu.

References

[1] AWS. The security design of the AWS Nitro system (initial publication). https://web.archive.org/web/20221229051409/https://docs.aws.amazon.com/pdfs/whitepapers/latest/security-design-of-aws-nitro-system/security-design-of-aws-nitro-system.pdf, 2022.

[2] Google Cloud. Cloud Run: Container to production in seconds. https://cloud.google.com/run/, 2023.

[3] Dmitry Evtyushkin and Dmitry V. Ponomarev. Covert channels through random number generator: Mechanisms, capacity estimation and mitigations. In *CCS*, 2016.

[4] gVisor Contributors. The container security platform. https://gvisor.dev/, 2023.

[5] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-level cache side-channel attacks are practical. In *IEEE S&P*, 2015.

[6] Antoon Purnal, Furkan Turan, and Ingrid Verbauwhede. Prime+Scope: Overcoming the observer effect for high-precision cache contention attacks. In *CCS*, 2021.

[7] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *CCS*, 2009.

[8] Pepe Vila, Boris Köpf, and José F. Morales. Theory and practice of finding eviction sets. In *IEEE S&P*, 2019.

[9] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *USENIX Security*, 2014.

[10] Zirui Neil Zhao, Adam Morrison, Christopher W. Fletcher, and Josep Torrellas. Everywhere all at once: Co-location attacks on public cloud FaaS. In *ASPLOS*, 2024.

[11] Zirui Neil Zhao, Adam Morrison, Christopher W. Fletcher, and Josep Torrellas. Last-level cache side-channel attacks are feasible in the modern public cloud. In *ASPLOS*, 2024.