Poster: Measuring the Risk of Exposed Configurations in Serverless Environment

Dayeon Kang
Manning College of Information & Computer Sciences
University of Massachusetts Amherst
Amherst, MA 01003
Email: dayeonkang@umass.edu

Pubali Datta
Manning College of Information & Computer Sciences
University of Massachusetts Amherst
Amherst, MA 01003
Email: pdatta@umass.edu

Eric Pauley
Department of Computer Science
Virginia Tech
Blacksburg, VA 24060
Email: pauley@cs.vt.edu

Abstract

Motivation. Serverless computing, built on the Function-as-a-Service (FaaS) paradigm, allows developers to deploy individual functions to the cloud while the provider automatically handles scaling, provisioning, and lifecycle management. Because serverless functions are ephemeral and stateless – retaining no persistent storage, fixed IP address, or dedicated container – they depend heavily on external services such as API gateways, cloud storage, and third-party APIs. In practice, the URLs and credentials needed to reach those services are routinely hard-coded directly into function source code.

This architectural pattern creates a security risks that are structurally distinct from those in serverful cloud computing. Because serverless function URLs are user-defined and human-readable, a deleted function's endpoint can be immediately re-registered by an attacker to silently intercept traffic from clients holding stale references. The risk compounds because the credentials and resource identifiers hard-coded in function source code (e.g., access keys, secret keys, access tokens, and endpoint permissions) persist long after a function is deleted or abandoned. An attacker who obtains this code inherits a map of the application's endpoint connectivity.and can exploit live resources directly, or re-register expired ones to impersonate legitimate services. In this work, we measure and analyze the risks arising from these latent resources and credentials in serverless environments.

Related Works. Prior work on latent cloud configurations has focused primarily on serverful environments. Pauley et al. [1] studied IP address reuse in EC2 using the DScope Internet telescope [2], showing that reallocated addresses can expose sensitive data such as financial transactions and PII. In serverless environments, however, the absence of static IP addresses makes such techniques inapplicable. More recent work addresses serverless-specific concerns: Liu et al. [3] measured abused functions using passive DNS datasets, and Raffa et al. [4] proposed a static analysis framework for tracing sensitive data flows in serverless code. Both focus on actively deployed functions. Our work takes a different view, statically measuring the risks introduced by latent configurations, which are credentials and resources that remain exposed after the functions referencing them have been deleted or abandoned.

Threat Model. We consider an attacker who has obtained the source code of a deployed serverless function – in the most common case, through public GitHub repositories, where developers routinely publish deployment-ready code. The threat is more severe when code is accessed through private repositories or directly from a cloud platform, as private code is less likely to have been anonymized and typically contains a richer set of hard-coded secrets.

Research Questions. We investigate three questions: (1) How many dangling serverless functions, which are the potential target of takeover, appear in the Certificate Transparency log? (2) How prevalent are latent resources and credentials in the serverless code that attackers exploit? (3) How expansive is the attack surface when mapping linked cloud or third-party services reachable through serverless function code?

Analysis Pipeline. To address our research questions, we design a static analysis pipeline (Figure 1) that measures latent configurations in serverless environments and constructs a call graph centered on the serverless function. The pipeline operates over two datasets: (1) a Certificate Transparency (CT) log snapshot, and (2) a serverless function code dataset comprising public GitHub repositories and official application templates from AWS and Microsoft Azure [5], [6].

To build the code dataset, we collect public GitHub repository candidates by filtering on serverless-specific topics such as `serverless`, `aws-lambda`, and `azure-functions`. We then verify each candidate's relevance by checking for platform-specific configuration files, language patterns, and containerized function interfaces across nine serverless platforms. For instance, a Python file with the `handler(event, context):` signature is classified as AWS Lambda code, while the presence of `host.json` and `function.json` indicates Azure Functions. Using these platform-specific signals, we collect 16,530 verified serverless repositories along with their metadata: collection timestamp, owner, GitHub URL, star and fork counts, description, programming languages, and tagged topics.

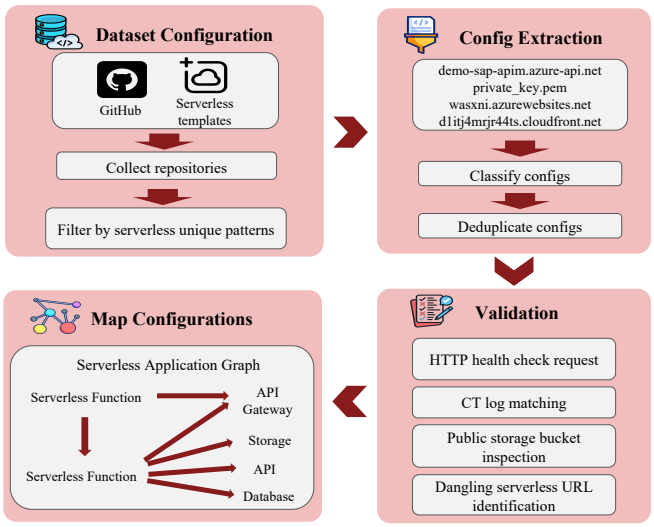


Figure 1. Overview of analysis pipeline

The pipeline proceeds in four stages. First, we cross-reference identified serverless function URLs against the CT log to determine whether they are dangling (unreachable) or eligible for re-registration. This step is particularly significant for Azure Functions, where URLs incorporate user-defined free text and can be immediately re-registered after deletion. From the October 2025 CT log snapshot, we identify 664,631 domains under `azurewebsites.net`, of which 62,032 (9.3%) are unreachable and dangling. Second, we extract and classify credentials and resource references from the code dataset using 118 regular expression patterns covering API keys, access tokens, and cloud resource URLs. Third, we validate the reachability of extracted resources by issuing non-intrusive health check requests and matching results against the CT log; credentials are assessed passively through pattern matching to respect ethical boundaries. Fourth, for publicly accessible cloud storage resources, we supplement our analysis by retrieving bucket contents via Grayhatwarfare [7] to examine latent configurations beyond the immediate serverless context. Finally, we aggregate all extracted resources and credentials into a per-application call graph, mapping the connectivity between each serverless function and its referenced external services. This graph structure makes lateral attack paths explicit: it reveals which user-created resources are dangling and eligible for impersonation or takeover, characterizes cross-provider dependencies through a bipartite graph of functions and services, and shows how an attacker who compromises a single function can follow hard-coded links to reach downstream databases, storage buckets, and APIs.

Preliminary Analysis Result. Table 1 summarizes our findings across 16,530 serverless repositories. We extracted 4,084 cloud credentials, 249 third-party credentials, 8,704 general credentials, and 1,457 key files — all without placeholders. As many extracted cloud product URLs point to platform portals, billing pages, or documentation rather than user-created infrastructure, we further cluster URLs by domain name and embedded resource ID patterns to isolate user-created resources. This yields 16,596 such resources across different cloud providers: 7,175 AWS S3 buckets, 3,935 AWS API Gateway endpoints, 420 AWS ECR registries, 140 AWS Lambda URLs, 903 GCP Cloud Functions, 559 GCP App Engine services, 775 Azure Web Apps, 281 Azure Blob Storage containers, among others.

These exposed credentials and user created resources have direct security implications. For resources that remain live, an attacker with the extracted references can inject malicious payloads into backend systems, exploiting credentials. For resources that have since been deleted – such as S3 buckets or Azure Function endpoints, both of which allow re-registration under the original identifier – an attacker can silently impersonate the original service and intercept traffic from clients that still hold the stale reference.

Contributions. We conduct a large-scale empirical measurement study to identify risks caused by latent configurations in serverless environments with static analysis pipeline. We curate 16,530 public serverless function repositories spanning nine different serverless platforms. We discovered 17,031 credentials and 16,596 user-created cloud resources from 16,530 public serverless applications.

TABLE 1. PRELIMINARY RESULT OF CLASSES WITH TOTALS AND PLACEHOLDER COUNTS

Class	Total	Placeholder
Cloud credential	4,581	497 (10.85%)
Cloud resource	72,232	18,814 (26.05%)
Cloud product URL	131,259	4,921 (3.75%)
Third-party credential	287	38 (13.24%)
Third-party URL	211,938	1,887 (0.89%)
Credential	10,706	1,966 (18.36%)
Key file	1,457	–

References

[1] E. Pauley, R. Sheatsley, B. Hoak, Q. Burke, Y. Beugin, and P. McDaniel, "Measuring and mitigating the risk of ip reuse on public clouds," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 558–575.

[2] E. Pauley, P. Barford, and P. McDaniel, "{DScope}: A {Cloud-Native} internet telescope," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 5989–6006.

[3] Y. Liu, M. Liu, Y. Zhang, B. Liu, J. Zhang, G. Hong, H. Duan, and M. Yang, "Dive into the cloud: Unveiling the (ab) usage of serverless cloud function in the wild," in *Proceedings of the 2025 ACM Internet Measurement Conference*, 2025, pp. 63–77.

[4] G. Raffa, J. Blasco, D. O'Keeffe, and S. K. Dash, "{CloudFlow}: Identifying security-sensitive data flows in serverless applications," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 1073–1090.

[5] "Aws serverless application repository," https://serverlessrepo.aws.amazon.com/applications, [Accessed 18-03-2026].

[6] "Sample code from microsoft developer tools," https://learn.microsoft.com/en-us/samples/browse/?expanded=azure, [Accessed 18-03-2026].

[7] "Public buckets by grayhatwarfare," https://buckets.grayhatwarfare.com/, [Accessed 18-03-2026].

Measuring the Risk of Exposed Configurations in Serverless Environment

Dayeon Kang, Pubali Datta, Eric Pauley*
University of Massachusetts Amherst, Virginia Tech*

Motivation

1. Serverless Architecture

• Serverless function is usually used with other cloud products and third-party APIs

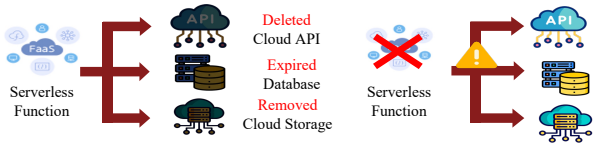
2. Hard-coded configurations

• Publicly available serverless function often contain hard-coded URLs, API endpoints, and credentials for external cloud services

Examples:
handler.py
API_URL = https://func-processor-abcdefgxzy.azurewebsites.net/api/orchestrators/
STORAGE = s3://montys-data-bucket/somebamfile.bam
TOKEN = ghp_alwajkvehlekasgndnjklabns
private_key.pem

3. Latent Configurations

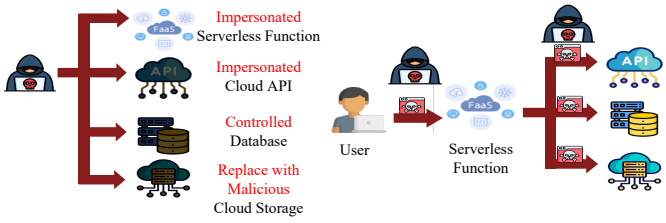
• If the resources or serverless functions are deleted or unreachable, they become latent configurations in the serverless environment



* Their outdated connection details remain hard-coded within the function's code

4. Threat Model

• Attackers can impersonate abandoned cloud resources and intercept requests from victim serverless applications



- Attackers may exploit the sensitive credentials
- Attackers may register or occupy the abandoned resource endpoint to perform resource re-registration and takeover
- Attackers may inject malicious messages in the middle of the communication

Challenge

Traditional Interapplynet Telescope methods are difficult to to serverless environments

(1) Limited Visibility

Traditional Method (e.g., Internet Telescope)

Cannot Capture Packet → Cannot observe every ingress traffic

HTTP Traffic Only Reached → Cannot observe every ingress traffic

(2) Dynamic Nature of Serverless Computing

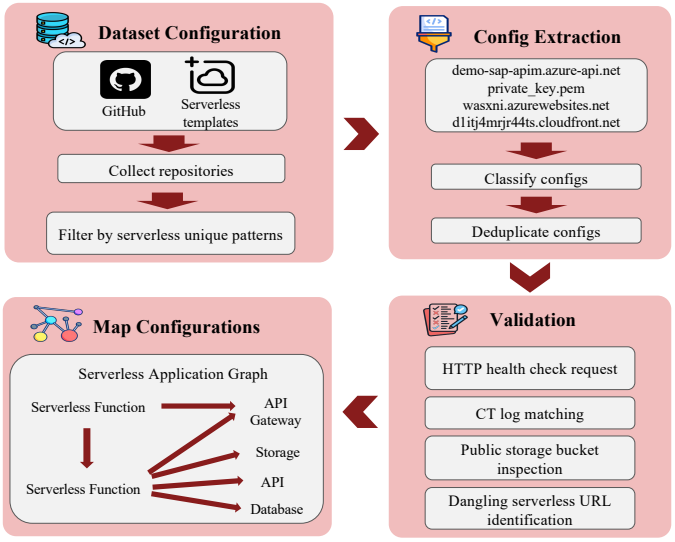
Serverless Function

⚠ Dynamic IP Address → Difficulty in measuring statically

⚠ Ephemeral Instances → Difficulty in measuring statically

We measure serverless environment risks through publicly accessible serverless function code instead of Internet Telescope methods

Analysis Pipeline



Preliminary Analysis Results

Large-scale Extraction from Public Repositories

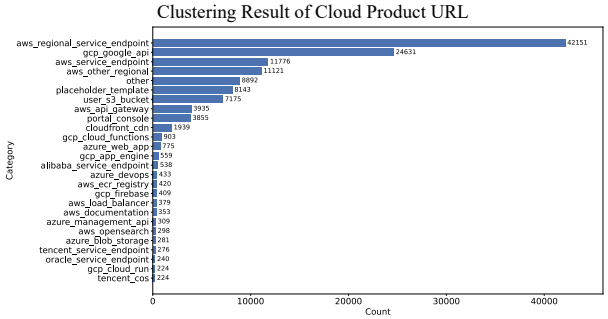
Step 1: Search by Topic
40,296 repositories
Collected metadata URL, description, created/updated/published date, language, etc.

Step 2: Filter by serverless patterns
16,530 serverless repositories
Serverless unique configuration files, invocation code, Docker container serverless base image

Step 3: Extract configurations, and classify them
* Used 118 fine-grained regex pattern matching

Class	Total	Placeholder
Cloud credential	4,581	497 (10.85%)
Cloud resource	72,232	18,814 (26.05%)
Cloud product URL	131,259	4,921 (3.75%)
Third-party credential	287	38 (13.24%)
Third-party URL	211,938	1,887 (0.89%)
Credential	10,706	1,966 (18.36%)
Key file	1,457	-

Clustering Result of Cloud Product URL



Conclusion and Future Work

Conclusion

- Publicly available serverless applications contain sensitive credentials and resources.
- Dangled and latent resources linked with serverless functions are vulnerable.
- We configured 16,530 public serverless repositories across 9 cloud platforms.
- We discovered 17,031 credentials and 16,596 user-created cloud resources.

Future Work

- Validate the classified resources and credentials.
- Construct the call graph with extracted resources.
- Analyze the risk with pipeline outcome.
- Prove the attack scenario.