CLOUD CYBERSECURITY RESEARCH

Serverless Tokens in the Cloud: Exploitation and Detections

13 min read

RELATED PRODUCTS

Cortex | Cortex Cloud | Unit 42 Incident Response

By: Zohar Zigdon

Published: June 13, 2025

Categories: Cloud Cybersecurity Research , Threat Research

Tags: AWS , Google Cloud , Microsoft Azure , Serverless

Share

Table of Contents

Attackers target serverless functions for these reasons:

- Serverless functions can often be vulnerable to remote code execution (RCE) or server-side request forgery (SSRF) attacks due to insecure development practices
- Serverless functions are often publicly exposed (either by design or due to misconfiguration) or process inputs from external sources
- Attackers can exploit serverless tokens to obtain unauthorized read/write access that could potentially jeopardize critical infrastructure and data

When using serverless functions, it is important to consider the risks involved when application developers deploy insecure code to cloud functions, and to understand the threats that target these functions.

How Serverless Authentication Works in Major Cloud Platforms

Using the serverless approach, applications are deployed by executing functions on demand. The primary advantage of this approach is that applications or specific components can be run on an as-needed basis, eliminating the need for a continuously running execution environment.

Each of the major cloud providers shares similar concepts for serverless functions, such as:

- Support for multiple code languages
- The absence of SSH access due to their fully managed architecture
- Using roles, service accounts or managed identities to securely manage resource access

AWS Lambda

The IAM service in AWS enables the creation and management of users, permissions, groups and roles. The service is responsible for managing identities and their level of access to AWS accounts and services, and control of various features within an AWS account.

Serverless functions use Lambda roles, which do not have default permissions; IAM policies must be used to manage permissions and secure access to other AWS services.

To make permission management simpler, AWS provides managed policies (like `AWSLambdaBasicExecutionRole`) that developers often attach to these roles to quickly enable basic functionality.

When an IAM role is associated with a Lambda function, the AWS Security Token Service (STS) automatically generates temporary security credentials for that role.

This is a secure way to get access to credentials at runtime without the risks associated with long-term, hard-coded

`AWS_SESSION_TOKEN`. These variables are accessible to the code of the function during runtime, allowing the function to interact securely with other AWS services.

Google Cloud Functions

Google Cloud Functions use service account tokens to authenticate and authorize access to other Google Cloud services. A service account is a specialized Google account that is tied to a project and represents a non-human identity, rather than to an individual user.

When deploying a Cloud Function, developers can attach the function to a custom service account or the default service account. When using a custom service account, developers can assign specific IAM roles to define the exact permissions the function requires to perform its tasks.

Default service accounts are user-managed accounts that Google Cloud automatically creates when users enable specific services. By default, Google Cloud Functions use different service accounts depending on the generation:

> First-generation Cloud Run functions use the App Engine default service account (`<project_id>@appspot.gserviceaccount.com`)
>
> Second-generation Cloud Run functions use the Default Compute service account (`<project_number>-compute@developer.gserviceaccount.com`)

These default service accounts are granted editor permissions when developers are onboarding to GCP without an organization, allowing them to create, modify and delete resources within the project. However, in projects under a GCP organization, default service accounts are created without any permissions. In such cases, they can only perform actions that are explicitly allowed by the roles assigned to them.

Although a GCP Function operates in a serverless environment, when it comes to accessing the credentials associated with the function, it behaves similarly to a traditional server, such as a virtual machine (VM) instance, by retrieving its service account tokens from the Instance Metadata Server (IMDS) at runtime. The IMDS at `hxxp://metadata.google[.]internal/` provides short-lived access tokens for the function's associated service account. These tokens enable the function to authenticate to GCP services.

Azure Functions

Azure managed identities provide a secure and seamless way for Azure resources to authenticate to and interact with other Azure services without the need for hard-coded credentials. In the same way as AWS and GCP's function services, these identities eliminate the risks associated with managing credentials in code.

System-assigned managed identities are tied to a single resource and automatically deleted when the resource is removed. On the other hand, user-assigned managed identities are independent resources that can be assigned to multiple services, offering more flexibility for shared authentication scenarios.

`IDENTITY_ENDPOINT`: An environment variable that contains the address of the local managed identity endpoint provided by Azure. This is a local URL from which an app can request tokens.

`IDENTITY_HEADER`: A required parameter when querying the local managed identity endpoint. This header is used to help mitigate SSRF attacks.

The function sends an HTTP GET request to the local managed identity endpoint with the `IDENTITY_HEADER` included as an HTTP header. (For details on the request structure, refer to Azure's documentation on acquiring tokens for App Services)

Microsoft Entra ID (formerly Azure Active Directory) verifies the function's identity and issues a temporary OAuth 2.0 token that is scoped specifically for the target resource.

The function includes the issued token in its request to the Azure resource (e.g., Azure Key Vault, Storage Account).

The Azure resource validates the token and checks the function's permissions using role-based access control, or resource-specific access policies.

If authorized, the resource grants access to perform the requested operation. This ensures secure and seamless authentication without the need to use hard-coded secrets in the code.

Token Exfiltration Attack Vectors

This section discusses the risks and threats involved in the use of serverless functions. When developing and configuring functions, application developers should be sure to secure those functions against attacks like SSRF and RCE. In the absence of such security, attackers could manipulate serverless functions that are vulnerable to SSRF, causing the functions to send unauthorized requests to internal services. This can lead to unintended access or exposure of sensitive information within the system, such as access tokens, internal database content or service configurations. It is important to emphasise that these risks primarily arise when functions are public, or process inputs from external users and other sources.

In SSRF attacks, an attacker tricks a server (like a serverless function) into making HTTP requests to internal or external resources that the attacker shouldn't have access to. Since the server itself makes the request, it can access internal services that are not exposed to the internet. A vulnerable serverless function takes a URL as input and fetches data from it.

In GCP, SSRF can be used to access the IMDS at `hxxp://metadata.google[.]internal/`, extracting short-lived service account tokens. An attacker can then leverage these tokens to impersonate the function and perform unauthorized actions within its IAM role's permissions. Remote code execution vulnerabilities allow attackers to execute arbitrary code within a function's environment.

For AWS Lambda, RCE attacks could expose temporary credentials stored in environment variables, such as `AWS_ACCESS_KEY_ID` and `AWS_SESSION_TOKEN`.

The following simulations demonstrate possible ways attackers could extract serverless tokens and use them for malicious activities in different cloud service providers (CSPs). These attacks could leverage unsecure function code that was deployed by application developers.

Simulation 1: Gaining Direct Access to IMDS from GCP Function

To conduct this simulation, we deployed two Google Cloud Run functions that access the function metadata service and extract the tokens of the two different attached service accounts from the following path:

```
hxxp[://]metadata.google[.]internal/computeMetadata/v1/instance/service-accounts/default/token
```

The first example demonstrates the extraction of a default service account. The second example demonstrates the extraction of a custom service account. These examples show how code can directly access the metadata service, just as a vulnerable SSRF code application could access it as well.

Example 1: Extracting a GCP Default Service Account

As shown in Figure 1, the service account attached to the function was the default serverless service account. Figure 2 demonstrates that the returned access token belongs to the same service account.

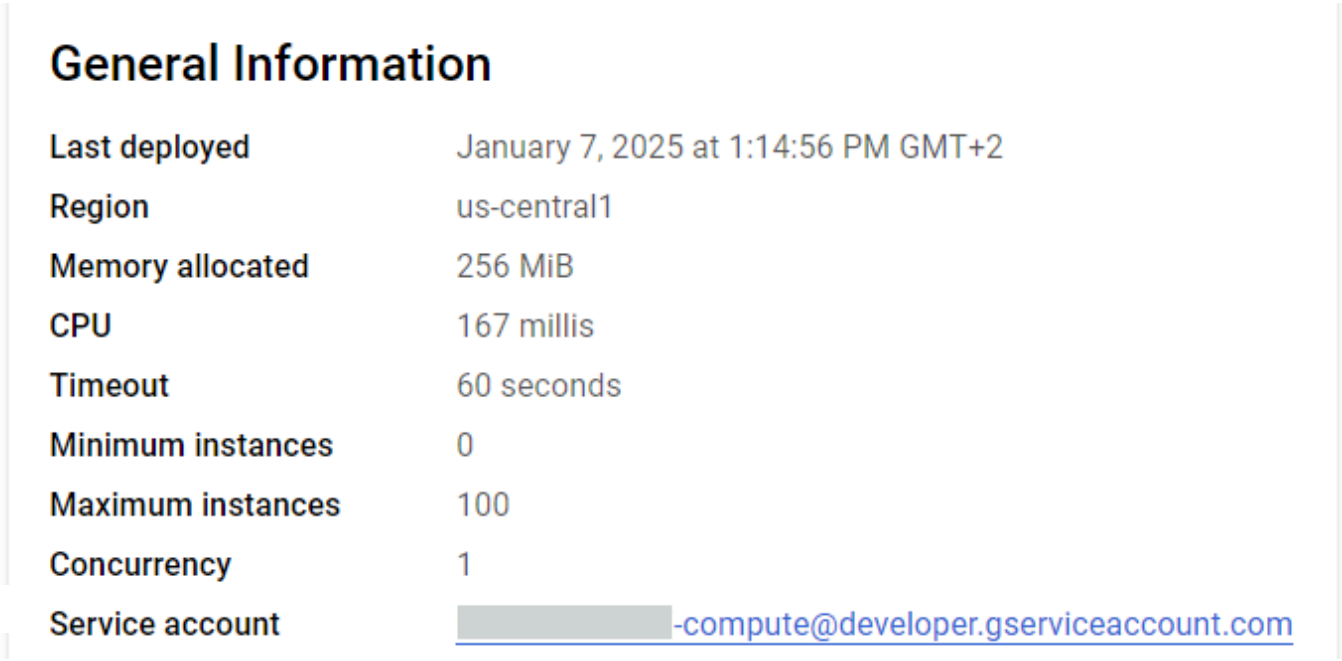


Figure 1. General information about the function, including an attached service account name (the default service account).

```
@cloudshell:~ (xdr-analytics)$ curl -m 70 -X POST https://us-central1-[redacted].cloudfunctions.net/zzigdon-imds-access-test-function -H "Authorization: bearer $(gcloud auth print-identity-token)" -H "Content-Type: application/json"
{"access_token":"ya29.c.[redacted]","service_account_email":"[redacted]-compute@developer.gserviceaccount.com"}
```

Figure 2. Code snippet of a command used to extract the service account access token, including the output.

Example 2: Extracting a GCP Custom Service Account

Figure 3 shows the custom service account that we attached to the function. Figure 4 shows the returned access token belonging to the custom service account. We then used the returned token to perform operations in the environment.

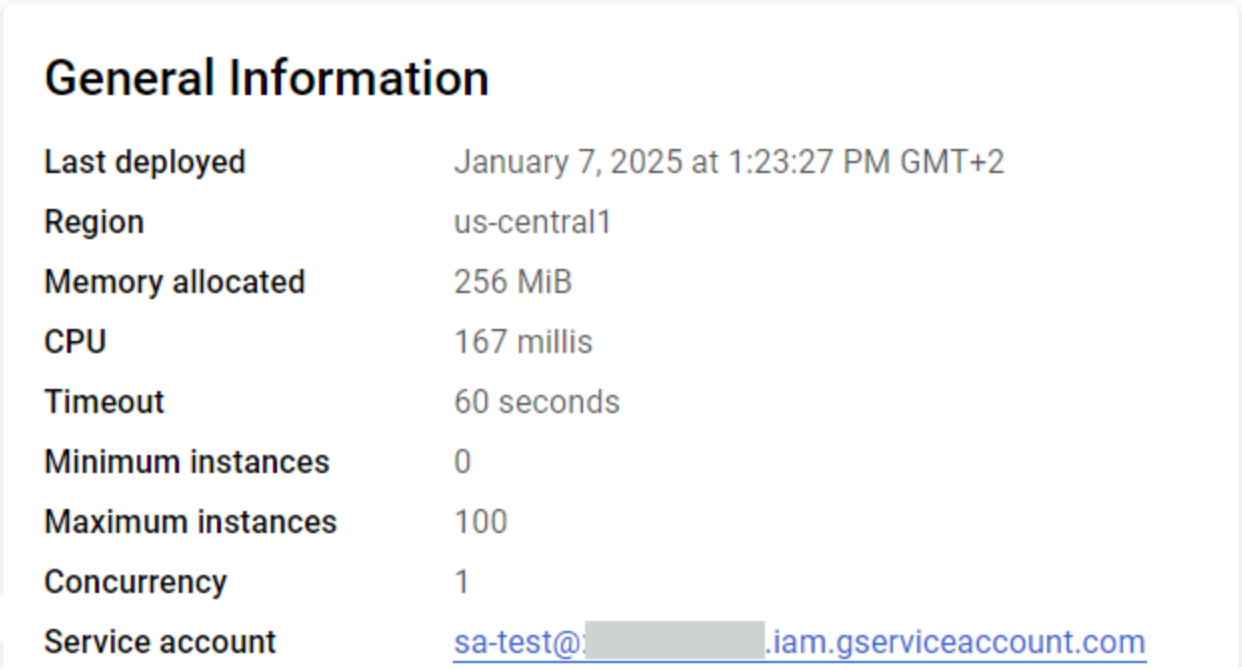


Figure 3. General information about the function, including the attached service account name (a custom service account).

```
@cloudshell:~ (xdr-analytics)$ curl -m 70 -X POST https://us-central1-[redacted].cloudfunctions.net/function-1 -H "Authorization: bearer $(gcloud auth print-identity-token)" -H "Content-Type: application/json"
{"access_token":"ya29.c.[redacted]
```

Figure 5 below shows an example of a command to list buckets.

```
curl -H "Authorization: Bearer SERVICE_ACCOUNT_ACCESS_TOKEN"
  https://storage.googleapis.com/storage/v1/b?project=<>"
```

Figure 5. Using the returned token to list buckets in a service account.

If attackers can list and read bucket contents in GCP, they can access sensitive files like credentials, backups or internal configurations. This allows them to steal data, compromise the system or move laterally within the environment.

Once attackers obtain an access token for a service account with Editor permissions in GCP (such as the default Compute Engine service account), they can modify, delete or create resources across most services. This includes accessing sensitive data, deploying malicious workloads, escalating privileges or disrupting services. Editor access effectively grants near-full control over the project.

Simulation 2: Using RCE to Retrieve Tokens Stored in AWS Lambda Function Environment Variables

In this simulation, we accessed the environment variables of the Lambda function and extracted from it the `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` and `AWS_SESSION_TOKEN`.

Figure 6 shows the output of the Lambda function code.

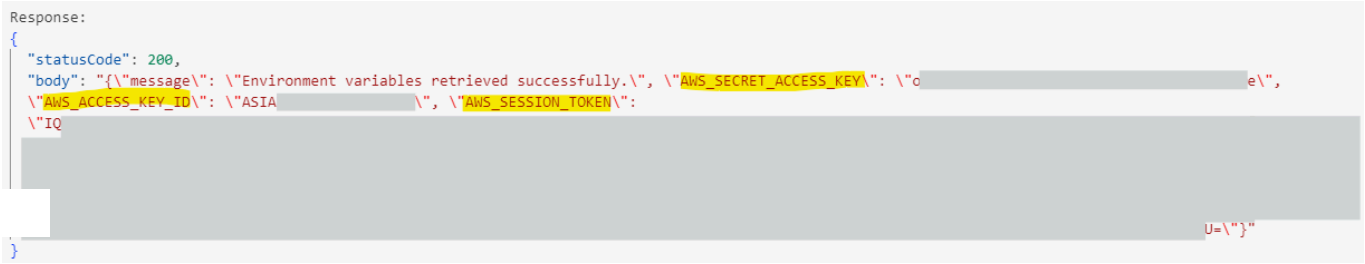


Figure 6. The values of the returned environment variables.

Figure 7 shows the setup of temporary AWS credentials obtained from the previous step (the Lambda environment variables).

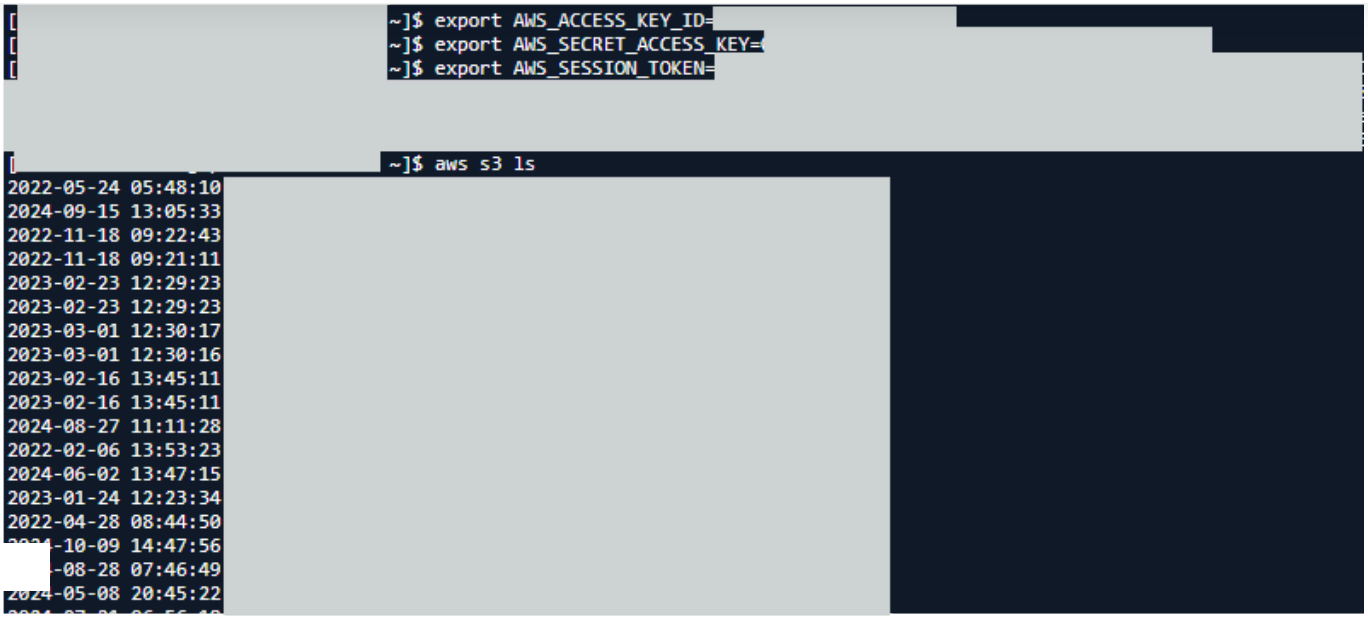


Figure 7. Code snippet of commands used to list S3 buckets.

Then we used the following command to list all S3 buckets the authenticated session could access.

```
1 aws s3 ls
```

Simulation 3: Using RCE to Retrieve Tokens From Local Identity Endpoint of an Azure Function

In this simulation, we accessed the environment variables of the Azure function and extracted the `IDENTITY_ENDPOINT` and the `IDENTITY_HEADER` by executing remote commands. Then we extracted the managed identity token from the local identity endpoint, providing these parameters shown below in Figure 8:

Resource

`api-version`

`X-IDENTITY-HEADER`

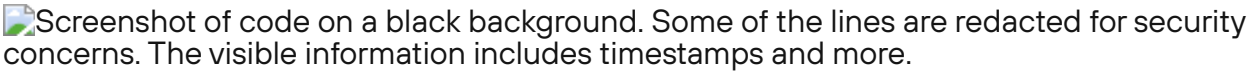

Screenshot of code on a black background. Some of the lines are redacted for security concerns. The visible information includes timestamps and more.

Figure 8. The script output, including the access token.

Detecting and Preventing Token Exfiltration

Detection consists of two stages:

- Validating that the identity is attached to a serverless function
- Identifying anomalous behavior of the serverless identities. Such behavior could include:
  - Source IP addresses that do not suit the context in which the function is executing, such as addresses from Autonomous System Numbers (ASNs) that are not associated with a cloud provider
  - Serverless identities making requests with suspicious user agents

Step 1: Identifying Serverless Identities

To identify service accounts attached to serverless functions in GCP, we analyze the `serviceAccountDelegationInfo` section in the logs. This information provides crucial insights into the delegation chain of service accounts. Specifically, when a service account is attached to a function, it delegates its authority to a default serverless service account:

- Google Cloud Run Service Agent (`service-<PROJECT_NUMBER>@serverless-robot-prod.iam.gserviceaccount[.]com`)
- `gcf-admin-robot.iam.gserviceaccount[.]com` (`service-PROJECT_NUMBER@gcf-admin-robo.iam.gserviceaccount[.]com`)

These service accounts execute tasks on behalf of the function.

For example, in the log entry in Figure 9, we see the custom service account that was attached to a function (`sa-test@<project-id>.iam.gserviceaccount[.]com`) delegating its authority to the Cloud Run Service Agent.

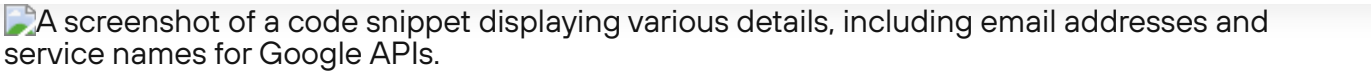


Figure 9. Log entry showing custom service account.

In Figure 9 above, the `principalEmail` field under `authenticationInfo` specifies the service account being used (`sa-test[@]xdr-analytics.iam.gserviceaccount[.]com`).

The `serviceAccountDelegationInfo` shows the first-party principal (in this case: `service-<PROJECT_NUMBER>@serverless-robot-prod.iam.gserviceaccount[.]com`), indicating that the service account is operating within a serverless environment like Cloud Functions or Cloud Run.

An effective approach to discover serverless identities is to profile service accounts that have previously delegated their authority to default serverless service accounts. This ensures that even if non-default service accounts are attached to functions, they are identified.

In AWS, Lambda functions rely on IAM roles for secure access to AWS services. These roles generate temporary credentials. We can identify the Lambda identity by its role name.

In Azure, managed identities attached to Azure Function Apps are used for authentication.

Step 2: Identifying Unusual Behavior for Serverless Identities

In a secure cloud environment, serverless functions are typically intended to perform automated, scoped tasks such as

Of note, this method of identifying remote use of serverless tokens according to the user agent cannot be performed in Azure, because user agent information does not appear in Azure logs.

Another approach for detecting remote use of a serverless identity token is to correlate the location of a token's use with ASN ranges of known cloud provider IP addresses. If a request originates from an external IP address not associated with the CSP, it triggers an alert, highlighting potential unauthorized token usage outside the cloud environment.

Prevention Strategies

Securing serverless tokens requires a combination of proactive measures, posture management and runtime monitoring security practices to minimize the risk of exploitation. First, implement the principle of least privilege by assigning roles with the minimum required permissions for serverless functions. This reduces the potential impact of token misuse.

Additionally, to protect serverless runtime environments in GCP and Azure, restrict access to IMDS by configuring network-level controls and applying request validation mechanisms. Ensure robust input validation and sanitization to prevent attackers from using exploitation techniques like SSRF to access sensitive metadata, tokens and other cloud resources like APIs and databases.

Conclusion

Serverless computing is the preferred choice for modern application development because it offers significant advantages in scalability, cost-efficiency and simplified infrastructure management.

The credentials that enable these functions to interact with cloud services are a critical security element and a prime target for attackers. Compromising these credentials can lead to severe consequences, including unauthorized access to cloud resources and data exfiltration.

Implementing proactive posture management and runtime monitoring protections is a crucial strategy in protecting cloud environments.

Organizations can better protect their serverless environments by:

- Understanding the mechanics of serverless credentials and best practices to provision and manage them in AWS, Azure and GCP
- Recognizing common attack vectors like token exfiltration via IMDS exploitation or environment variable access

Palo Alto Networks Protection and Mitigation

Palo Alto Networks customers are better protected from the threats discussed above through the following product:

North America Toll-Free: 866.486.4842 (866.4.UNIT42)

EMEA: +31.20.299.3130

APAC: +65.6983.8730

Japan: +81.50.1790.0200

Palo Alto Networks has shared these findings with our fellow Cyber Threat Alliance (CTA) members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. Learn more about the Cyber Threat Alliance.

Additional Resources

Working with Lambda environment variables – AWS Lambda Development Guide

Securing Lambda environment variables – AWS Lambda Development Guide

Managing permissions in AWS Lambda – AWS Lambda Development Guide

Function Identity – Google Cloud

What are managed identities for Azure resources? – Microsoft Learn

Securing Azure Functions – Microsoft Learn

What is AWS Secrets Manager? – AWS Secret Manager User Guide

Azure Key Vault basic concepts – Microsoft Learn

Secret Manager overview – Google Cloud Secret Manager Documentation

What is identity and access management (IAM)? – Microsoft Security

What are managed identities for Azure resources? – Microsoft Learn

Identity and Access Control – Introduction to AWS Security AWS Whitepaper

Service account credentials – Google Cloud IAM Documentation

Temporary security credentials in IAM – AWS Identity and Access Management User Guide

AWS Security Token Service API Reference – AWS Security Token Service Documentation

Compute Engine instances – Google Cloud Documentation

Google Cloud Projects – Google Cloud Documentation

User-managed service accounts – Google Cloud Documentation

Compare Cloud Run functions – Google Cloud Run Documentation

Default service accounts – Google Cloud Documentation

Authorize access to blobs using Microsoft Entra ID – Microsoft Learn

Storage account overview – Microsoft Learn

What is Azure role-based access control (Azure RBAC)? – Microsoft Learn

Resource-specific access policies – Microsoft Learn

Back to top

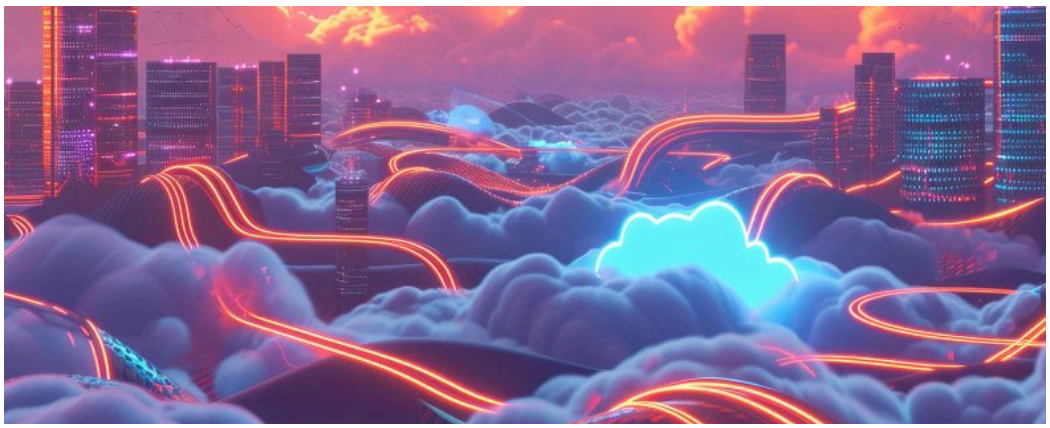
TAGS

AWS | Google Cloud | Microsoft Azure | Serverless

Threat Research Center

Next: JSFireTruck: Exploring Malicious JavaScript Using JSF*ck as an Obfuscation Technique

Related Cloud Cybersecurity Research Resources



THREAT RESEARCH

June 10, 2025

The Evolution of Linux Binaries in Targeted Cloud Operations

Linux Malware | Endpoint

THREAT RESEARCH

June 9, 2025

Roles Here? Roles There? of AWS IAM Roles Anywhe

AWS | Kubernetes

Newsletter

UNI
42
Sma
Log

Get updates from Unit 42

Peace of mind comes from staying ahead of threats. Subscribe today.

Your Email

Subscribe for email updates to all Unit 42 threat research.
By submitting this form, you agree to our Terms of Use and acknowledge our Privacy Statement.

Subscribe

Ri
Ar

Products and Services

Company

Popular Links

Privacy

Trust Center

Terms of Use

Documents

EN