Compromised Cloud Compute Credentials: Case Studies From the Wild

By Dror Alon

Published: 2022-12-08 · Archived: 2026-04-05 20:08:15 UTC

Executive Summary

Cloud breaches often stem from misconfigured storage services or exposed credentials. A growing trend of attacks specifically targets cloud compute services to steal associated credentials and illicitly gain access to cloud infrastructure. These attacks could cost targeted organizations both in terms of unexpected charges for extra cloud resources added by the threat actor, as well as time required to remediate the damage.

This blog highlights two examples of cloud compute credentials attacks in the wild. While the initial access phase is important, we will focus on the post-breach actions executed during the attack, and share the flow of these two attacks against the cloud infrastructure. The attack flows show how threat actors abuse stolen compute credentials to pursue a variety of attack vectors (such as cryptomining, data theft, etc.) and abuse cloud services in unexpected ways.

To detect the attacks described below, cloud logging and monitoring best practices as outlined by Amazon Web Services (AWS) and Google Cloud are essential, as they provide visibility into activity at the cloud infrastructure level. This emphasizes how important it is to follow Amazon Web Services and Google Cloud logging and monitoring best practices.

Palo Alto Networks helps organizations address security issues in the cloud with Cortex XDR for cloud, which detects cloud attacks such as cloud compute credentials theft and Prisma Cloud, which manages identity entitlement with least privilege entitlements.

Key Principle of Working in the Cloud

Before diving in, we should understand a very basic and important rule of working in the cloud. An entity, whether a human or a compute workload, needs legitimate and associated credentials to access a cloud environment at the infrastructure level. The credentials are used for authentication (to verify the entity's identity) and authorization (to verify what the entity is allowed to do).

As a best practice, when a compute workload executes API calls in the cloud (e.g., to query a storage service), the workload's associated credentials should be dedicated only to it. They should also be used only by that workload or human, and not by anyone else.

As we will see in both examples, an important security principle that can help reduce risk in cloud compute credentials attacks is least privilege access. In particular, this means that the privileges associated with those

credentials should be scoped down to the minimum set actually required by the code using them.This limits the actions an attacker can take when compute credentials are stolen.

Attack Case 1: Compromised AWS Lambda Credentials Led to Phishing Attack

An attacker was able to execute API calls on the behalf of the Lambda function by stealing the Lambda's credentials. This allowed the attacker to execute multiple API calls and enumerate different services in the cloud environment, as shown in Figure 1. While most of these API calls were not allowed due to lack of privileges, the attack resulted in a phishing attack launched from a AWS Simple Email Service (SES) that the threat actor created.

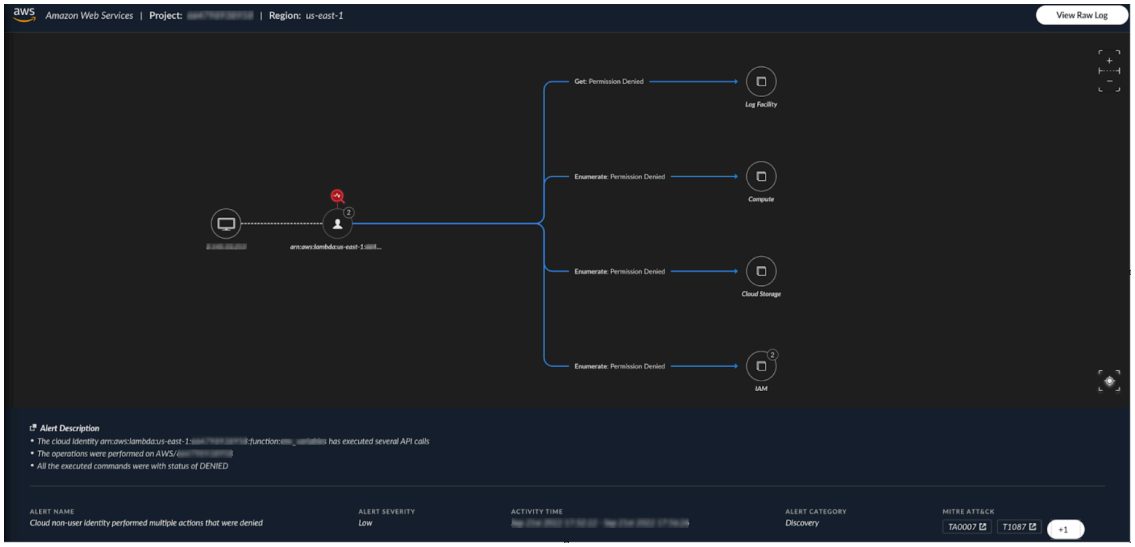

Figure 1. An attacker enumerating the cloud environment using a compromised Lambda function's credentials.

This phishing attack not only resulted in unexpected costs for the organization, it also exposed other organizations to extra risk as well.

While the outcome of this attack caused substantial impact, it could have been greater. In this case the victim didn't have an active SES, but if they had, the attacker could have abused it to launch an attack on the victim's organization or they could even have used a legitimate email account for the phishing attack.

Due to the variety of cloud services used by organizations, it can be difficult to predict where a cloud attack will end. Going from cloud to phishing was not necessarily an expected path.

Attack Flow

An attacker was able to exfiltrate the Lambda's environment variables and export them into their attacking machine (AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_SESSION_TOKEN).

As the credentials were exfiltrated, the attack was launched with the following steps:

> WHOAMI - 2022-05-20T20:35:49 UTC

The attack started with the GetCallerIdentity command. This command is equivalent to whoami, as it provides information about the entity the credentials are associated with. From the response, the attacker can gain additional information such as the account ID and the credentials type that was stolen. They cannot, however, determine anything about the privileges associated with the identity.

> Identify and Access Management (IAM) Enumeration - 2022-05-20T20:35:55 UTC

The next phase of the attack was an identity and access management (IAM) enumeration. IAM is considered to be a crown jewel for an attack. By gaining access to IAM, an attacker can elevate permissions and gain persistence on the victim's account.

The IAM enumeration included two API calls, which were denied due to lack of permissions:

- ListAttachedRolePolicies
- ListRolePolicies

It is fair to assume that the attacker noticed the lack of permission and therefore terminated the IAM enumeration after only two commands (possibly to avoid making unnecessary noise).

> General Enumeration 2022-05-20T20:39:59 UTC

After failing to enumerate IAM, the attacker started an enumeration on different services in different regions. This technique is much noisier as the attacker is trying to learn the architecture of the targeted account and, more importantly, gain access to sensitive information that could exist in the cloud account.

Some of the main services and API calls that were executed were:

Storage Enumeration

- ListBuckets
- GetBucketCors
- GetBucketInventoryConfiguration
- GetBucketPublicAccessBlock
- GetBucketMetricsConfiguration
- GetBucketPolicy
- GetBucketTagging

EC2 Enumeration

- GetConsoleScreenshot
- GetLaunchTemplateData
- DescribeInstanceTypes
- DescribeBundleTasks
- DescribeInstanceAttribute
- DescribeReplaceRootVolumeTasks

Network Enumeration

- DescribeCarrierGateways
- DescribeVpcEndpointConnectionNotifications
- DescribeTransitGatewayMulticastDomains
- DescribeClientVpnRoutes
- DescribeDhcpOptions
- GetTransitGatewayRouteTableAssociations

Logging Enumeration

- GetQueryResults
- GetBucketLogging
- GetLogRecord
- GetFlowLogsIntegrationTemplate
- DescribeLogGroups
- DescribeLogStreams
- DescribeFlowLogs
- DescribeSubscriptionFilters
- ListTagsLogGroup

Backup Enumeration

- GetPasswordData
- GetEbsEncryptionByDefault
- GetEbsDefaultKmsKeyId
- GetBucketReplication
- DescribeVolumes
- DescribeVolumesModifications
- DescribeSnapshotAttribute
- DescribeSnapshotTierStatus
- DescribeImages

SES Enumeration

- GetAccount
- ListIdentities

General Enumeration

- DescribeRegions
- DescribeAvailabilityZones
- DescribeAccountAttributes

> Backdoor 2022-05-20T20:43:22 UTC

While failing to enumerate IAM, the attacker tried (unsuccessfully) to create a new user by executing the CreateUser command.

> From the Cloud to a Phishing Attack 2022-05-20T20:44:40 UTC

As most of the API calls during the enumeration resulted in permission denied, the attacker was able to successfully enumerate SES. Therefore, the attacker launched a phishing attack by abusing the cloud email service, which included executing commands such as VerifyEmailIdentity and UpdateAccountSendingEnabled.

> Defense Evasion 2022-05-20T23:07:06 UTC

Finally, the attacker tried to hide some of his activities by deleting the SES identity by executing the DeleteIdentity command.

Additional Insights for Detection

A very important indicator of compromise (IoC) for this attack was the IP address 50.82.94[.]112.

API calls from the Lambda function are typically executed from its IP with the credentials (including AccessKeyId) that were generated for the Lambda. Therefore, every API call with that AccessKeyId is considered to be the Lambda function. However, during the attack, the threat actor was able to steal the Lambda's credentials, allowing the attacker to impersonate it.

Because of this, the IP is the key IoC as it is the way to detect that this is not the Lambda. The attacker was using the stolen credentials to impersonate and execute API calls on behalf of the Lambda function, but they were doing so from an IP address that wasn't attached to the Lambda, nor did it belong to the cloud environment.

Attack Case 2: A Compromised Google Cloud App Engine Service Account Deploying Cryptomining Instances

An attacker was able to steal the credentials for a Google Cloud App Engine service account (SA). There are many ways that attackers can accomplish this that are not necessarily related to any vulnerability in the cloud service provider. In many cases, for example, users store credentials in insecure locations or use easily guessed or brute-forced passwords.

In this case, the stolen SA was the default SA, which had a highly privileged role (Project Editor). This allowed the threat actor to launch an attack that ended with the creation of multiple high-core CPU virtual machines (VMs) for cryptomining purposes, as shown in Figure 2.

Figure 2. An attacker abusing a compromised App Engine SA to allocate multiple cloud instances for mining.

When the threat actor launched thousands of VMs in the victim's environment, it significantly increased their expected costs. Even if someone had noticed an attack like this in their environment after a short period of time, it would still have had a substantial cost impact.

Attack Flow

Google App Engine is a Google Cloud fully managed serverless platform, and the service account is the token. When the user creates an App Engine instance, the cloud provider creates a default SA and attaches it to the created App Engine.

This App Engine default SA has the editor role in the project. The editor role is highly privileged, which is a key factor the attacker took advantage of. This role allowed execution of high-privilege API calls, such as the following:

- Compute workload launch
- Firewall (FW) rule modification
- SA key creation

> Privilege Escalation 2022-06-16T12:21:17.624 UTC

The attack started with an attempt to escalate privileges. As mentioned above, by default the App Engine's SA has editor permissions on the project. With these permissions, the attacker tried to add the compute/admin role by adding the following object into the IAM policy:

```
{                                    "members":
["serviceAccount:XXXXXXX@appspot.gserviceaccount.com"],
                                     "role": "roles/compute.admin"}
```

As we can see, the appspot in the SA domain prefix indicates this SA belongs to an App Engine service.

> Allow Any/Any 2022-06-16T12:21:29.406 UTC

Next the attacker modified the FW rules on the project level. First, the attacker attempted to create a subnet (named default). Then, the attacker added the rule below into that subnet:

```
"request": {
                        "@type": "type.googleapis.com/compute.firewalls.insert",
                        "alloweds": [{
                                "IPProtocol": "tcp"
                        }, {
                                "IPProtocol": "udp"
                        }],
                        "direction": "EGRESS",
                        "name": "default-allow-out",
                        "network":
"https://compute.googleapis.com/compute/v1/projects/XXXXXXX/global/networks/default",
                        "priority": "0"}
```

This action furthers the attacker's goal of mining cryptocurrency. To enable unlimited cryptocurrency mining, the attacker removed any limitation on the networking level.

It is important to note the priority field. By setting this to zero, the attacker's rule is set to the highest priority, meaning it will take effect first in the order of the existing FW rules.

> An Army of Miners 2022-06-16T12:21:38.916 UTC

Once setup was complete, the main phase of the attack began, launching VMs in multiple regions.

While the attacker could have created high CPU machines, in this case the attacker instead created a standard VM (e.g., n1-standard-2) equipped with four high performance GPUs (e.g., nvidia-tesla-p100):

```
"methodName": "v1.compute.instances.insert",
                "request": {
                        "@type": "type.googleapis.com/compute.instances.insert",
                        "canIpForward": false,
                        "deletionProtection": true,
                        "disks": [{
                                "autoDelete": true,
                                "boot": true,
                                "initializeParams": {
                                        "sourceImage":
"https://compute.googleapis.com/compute/v1/projects/XXXXXXXX/zones/-/imageFamilyViews/ubuntu-2004
-lts"
                                },
                                "mode": "READ_WRITE",
                                "type": "PERSISTENT"
                        }],
                        "guestAccelerators": [{
                                "acceleratorCount": "4",
                                "acceleratorType":
"https://compute.googleapis.com/compute/v1/projects/XXXXXXXX/zones/us-central1-a/acceleratorTypes
/nvidia-tesla-p100"
                        }],
                        "machineType":
"https://compute.googleapis.com/compute/v1/projects/XXXXXXXX/zones/us-central1-a/machineTypes/n1-
standard-2",
                        "name": "node-api-82916",
                        "networkInterfaces": [{
                                "accessConfigs": [{
                                        "name": "external-nat",
                                        "type": "ONE_TO_ONE_NAT"
                                }],
                                "network":
"https://compute.googleapis.com/compute/v1/projects/XXXXXXXX/global/networks/default"
                        }],
                        "scheduling": {
                                "automaticRestart": true,
                                "onHostMaintenance": "TERMINATE"
                        }
                }
```


Overall, more than 1,600 VMs were created during this attack.

> Backdoor 2022-06-16T13:25:56.037 UTC

The attacker assumed the SA key used for the attack would be detected and revoked, and therefore created multiple SA keys (for later usage) by executing the google.iam.admin.v1.CreateServiceAccountKey API call.

Additional Insights for Detection

As in the first case we discussed, the IP is an important IoC. In this case, the attack was launched from multiple IPs (nine different IPs overall), some of which were active Tor exit nodes.

Again, we expect the App Engine's SA to be used from an IP within the cloud environment. It definitely should not be used from a Tor exit node.

Firewall rule modification is a common and popular technique used in attacks like these. Many organizations enforce network traffic rules that deny access to active mining pools, so attackers must modify firewall rules to accomplish their goals.

Lastly, by editing a network named default, the attacker tried to avoid detection. Unless this option is disabled, by default each new project is created with a default network. It seems the attacker was trying to take advantage of this, thus avoiding having to create their own network.

Conclusion: Compute Token Theft Is a Growing Threat

The theft of a compute workload's token is the common denominator for both cases we've discussed. While both examples above involve serverless services, this attack vector is relevant to all compute services.

It is important to emphasize that this type of attack could occur from different attack paths – including application vulnerabilities or zero-day exploits such as Log4Shell – not only from misconfigurations or poor cloud security posture management (CSPM).

To handle such attacks, cloud audit logs are essential, both for detection and for investigation and response (IR). Cloud audit monitoring is even more critical for serverless workloads where agents cannot be installed, thus making it harder to stop such attacks.

Logging and monitoring best practices provided by AWS and Google Cloud give clear guidance for how to prevent such cases. The AWS GuardDuty service could also help with detecting and alerting on similar attacks, such as EC2 instance credentials used from another AWS account. An additional prevention method is to configure an interface endpoint policy for Lambda to limit the usage of Lambda only within a VPC.

Palo Alto Networks customers receive protections from compute token theft with:

- Cortex XDR for cloud, which provides SOC teams with a full incident story across the entire digital domain by integrating activity from cloud hosts, cloud traffic, and audit logs together with endpoint and network data.
- Prisma Cloud, which assists organizations in managing identity entitlement, addresses the security challenges of managing IAM in cloud environments. Prisma Cloud IAM security capabilities automatically calculate effective permissions across cloud service providers, detect overly permissive access, and suggest corrections to reach least privilege entitlements.

Find out how you can protect your organization from cloud environment attacks with Unit 42 Cloud Incident Response services to investigate and respond to attacks and Cyber Risk Management Services to assess your security posture before an attack takes place.

Learn How to Detect Cloud Threats at Ignite '22
Attend the session "Unearth advanced threats using your network and cloud data," at Palo Alto Networks Ignite '22, the security conference of the future, to find out how to detect compromised cloud compute tokens. Our researchers will show you how to investigate and respond to cloud attacks with a live demonstration. Register now!
