What is the IMDSv1 attack?

EC2 instances reach a special internal URL at 169.254.169.254 to read their own configuration and temporary IAM credentials. The original Instance Metadata Service (IMDSv1) answered any HTTP request that arrived. If an attacker could trigger a request from inside the instance (most commonly through a Server-Side Request Forgery flaw in an application running on it), they could read the instance's IAM credentials from outside. AWS introduced IMDSv2 in 2019 to require a session-token handshake, which defeats simple SSRF chains. Many production AWS environments still have IMDSv1 enabled on at least some instances.

HOW IT WORKS

01 What went wrong with IMDSv1?

IMDSv1 answered any HTTP GET request that arrived at the metadata endpoint. There was no authentication, no proof that the request actually came from a workload on the instance. The intended design was 'only the instance can reach this address,' but any HTTP client running on the instance (legitimate or attacker-controlled) could read the credentials.

The attacker's job became: get any HTTP request fired from inside the instance to 169.254.169.254. The most reliable way to do that was to exploit a Server-Side Request Forgery flaw in an application running on the instance. SSRF lets an attacker tell the application to fetch a URL of the attacker's choosing. Point it at the metadata endpoint, read the response, and you have live temporary IAM credentials.

02 How did this lead to the Capital One breach?

In March 2019, an attacker exploited a misconfigured web application firewall in front of Capital One's AWS environment. The misconfiguration enabled an SSRF that reached the IMDSv1 endpoint and retrieved the temporary credentials assigned to an EC2 instance. Those credentials had `s3:GetObject` permissions across many buckets. The attacker enumerated buckets, downloaded approximately 100 million customer records (including 140,000 Social Security numbers and 80,000 bank account numbers), and posted samples on a public forum.

SOURCES

[1] AWS IMDSv2 Documentation

[2] AWS Blog, Add defense in depth against open firewalls (IMDSv2 announcement)

[3] CWE-918: Server-Side Request Forgery

The settlement reached $190 million plus a $80 million regulatory fine.

This exact chain (SSRF -> IMDS -> credentials -> data exfiltration) is one of the most common cloud-era attack patterns. It is the single most-cited reason AWS introduced IMDSv2.

03 What did IMDSv2 change?

IMDSv2 added a required session-token handshake. Before reading any metadata, the client must:

1. Send a PUT request to `/latest/api/token` with a header naming a desired session TTL. 2. Receive a session token in the response body. 3. Include that token as a header on every subsequent metadata request.

Why this defeats simple SSRF: most SSRF flaws can only fire GET requests. A PUT with a header is outside what they can do. Even when an SSRF supports PUT, the response token has to make it back through the SSRF response to the attacker, which often breaks in transit.

IMDSv2 also added a TTL on the session token (1 to 6 hours) and an IP hop-count check that drops requests proxied through more than one network hop. Either layer alone closes most exploit chains.

04 Why are some environments still vulnerable?

Three reasons we see in engagements:

- IMDSv1 backwards-compatibility. AWS lets instances accept both IMDSv1 and IMDSv2 by default. Operators must explicitly require IMDSv2 (set the metadata options to `HttpTokens=required`). Many do not.
- Older AMIs and base images. Long-running instances built from older AMIs may not have been updated to enforce IMDSv2.
- Auto Scaling Groups with old launch configurations. A modern AMI plus an old launch template equals new instances that accept IMDSv1.

The one-line fix at the account level: enable the EC2 'Required' default at the regional level via the

API, then audit existing instances and enforce the same setting.

05 How does SecureLayer7 test for this?

Every AWS engagement audits IMDSv2 enforcement:

- Enumerate every EC2 instance and read its `MetadataOptions`. Flag every instance that allows IMDSv1.
- For each in-scope web application running on EC2, test for SSRF that can reach the metadata endpoint. Confirm whether credentials are extractable.
- For each set of credentials extractable, document the realistic blast radius: which buckets, which databases, which services, which secrets reachable.
- For Kubernetes-on-EC2 environments, test that workloads cannot reach the node's metadata service from within pods (a separate finding class often missed by IMDSv2 audits).

Verify IMDSv2 enforcement across your AWS environment.
securelayer7.net/learn/cloud-security/imdsv1-attacks

Open online