AWS penetration testing.

AWS penetration testing means attacking your AWS setup the way a real attacker would, to find what is genuinely breakable before someone else does. It covers the part AWS leaves to you: who can do what (IAM users, roles, and policies), the resources you run (EC2, S3, Lambda, and so on), and how they all connect. AWS lets you test your own resources for most services without asking first.

HOW IT WORKS

01 What does an AWS pentest actually cover?

A useful engagement covers four layers:

- Identity. IAM users, roles, policies, service control policies, federation, and MFA. This is the most common path to a full takeover.
- Resource settings. S3 bucket policies, security groups, RDS exposure, Lambda permissions, ECR access. The CIS AWS Foundations Benchmark is a good starting checklist.
- Service-specific attacks. Pulling secrets from Lambda environment variables, abusing Step Functions, EventBridge rules, or SSM Session Manager, and CloudFormation drift.
- The application layer. The web apps, APIs, and services running on AWS. Normal application testing, plus AWS-specific pivots, like an SSRF that reaches the metadata endpoint or secrets sitting in a Lambda's environment variables.

02 What are the most common findings on an AWS pentest?

From the engagements we ran in the last year, the most common ones:

- Over-broad IAM policies. A role with `*:*`, or `iam:PassRole` against `*`, or a wildcard managed policy. Each one is a way to escalate privileges.
- Public S3 buckets. A leftover `s3:GetObject` for `Principal: *`, or a policy that grants more than the team realized. Often full of backups, logs, or build files with secrets inside.
- IMDSv1 still on. EC2 instances that still accept the old metadata service. Pair that with an SSRF in any app on the instance, and the attacker walks off with temporary IAM credentials.

SOURCES

[1] AWS Penetration Testing Policy

[2] AWS Shared Responsibility Model

[3] CIS AWS Foundations Benchmark

[4] MITRE ATT&CK for Cloud (IaaS)

- Long-lived access keys. Hardcoded in CI configs, committed to repos, shared between people, never rotated.
- Secrets in Lambda environment variables. Easy to read for anyone who can run the function or view its settings.
- Wide-open security groups. A database with port 5432 open to `0.0.0.0/0` because someone opened it once to debug and never closed it.

03 How does SecureLayer7 run an AWS pentest?

Four phases.

Phase 1, map the account. Every IAM user and role, every resource, every cross-account trust, every public endpoint. Open-source AWS auditing tools (ScoutSuite, Prowler) plus our own checks do the broad sweep. The pentester then reviews and prioritizes by hand.

Phase 2, chase identity paths. For each role, what can it do? What can the holder reach that the team did not expect? Which mix of permissions chains into an escalation? The classic move is a weak identity that can pass a stronger role somewhere downstream.

Phase 3, attack the resources. Test the known weak spots: public buckets, exposed Lambda calls, SSM document misuse, public RDS, IMDSv1 on EC2.

Phase 4, pivot through the apps. When your applications are in scope, test them for the AWS-specific impact: SSRF to the metadata endpoint, command injection into the AWS CLI, secrets pulled from a Lambda's environment.

The report gives a blast-radius note for each finding (what an attacker with this access could actually do) and the exact IAM or config change to close it.

Test your AWS environment end to end.
securelayer7.net/learn/cloud-security/aws-pentest

Open online