2025

Cyber Security

Assignment Manual

AWS SECURITY MISCONFIGURATION AUDIT

VIDHAAN VISWAS

https://www.vidhaanviswas.com

Security Misconfiguration Detection in AWS Environment using Python & boto3

Objectives

By the end of this lab, the learner will be able to:

- Configure an AWS environment for security testing.
- Create intentionally vulnerable AWS resources (S3, RDS, Security Groups).
- Write a Python script using boto3 to detect misconfigurations.
- Understand cloud security risks and audit techniques.
- Implement automated testing using pytest and mocking.
- Generate a vulnerability report.

Theoretical Learning

Cloud environments frequently suffer from misconfigurations such as:

✓ **Publicly exposed storage (S3 buckets)**
Leads to data leaks, unauthorized access.

✓ **Publicly accessible RDS databases**
Risk of brute-force attacks, credential leakage, data exfiltration.

✓ **Overly-permissive security group rules**
Ports like SSH (22) or MongoDB (27017) exposed to 0.0.0.0/0 significantly increase attack surface.

✓ **Missing versioning, logging, or backups**
Creates operational risk, lack of traceability, and inability to recover.

Using **Python + boto3**, we can automate the detection of these issues.

Prerequisites

Technical Knowledge:

- Basic Python programming
- Understanding of AWS (S3, RDS, EC2)
- Familiarity with CLI tools

Tools Required:

- Python 3.9+
- boto3
- pytest
- AWS CLI
- IAM user with Security Audit permissions

AWS Environment Setup (Creating Misconfigurations)

⚠ ***These steps intentionally create insecure resources for testing only.***

✓ Create an S3 Bucket (Misconfigured)

Go to: AWS Console → S3 → Create Bucket

❖ **Configure:**

- Disable → Block Public Access
- Logging → *Disabled*
- Versioning → *Disabled*
- After creation → upload a file → Make Public

❖ **Misconfigurations achieved:**

- Publicly accessible
- No logging
- No versioning

✓ Create an RDS Instance (Misconfigured)

Go to: AWS Console → RDS → Create database

❖ **Configure:**

- Public Access → YES
- Backups → *Disabled*
- Delete Protection → *Disabled*
- Engine → Any (MySQL, PostgreSQL, etc.)

❖ **Misconfigurations achieved:**

- Public DB
- No delete protection
- No automatic backups

✓ **Create a Security Group (Insecure Rules)**
Go to: AWS Console → EC2 → Security Groups → Create SG

❖ **Add inbound rules:**
- SSH → Port 22 → 0.0.0.0/0
- Custom TCP → Port 27017 → 0.0.0.0/0

❖ **Misconfigurations achieved:**
- Open SSH port
- Public MongoDB port

Python Environment Setup

✓ **Create virtual environment:**
- python -m venv aws-audit-env

✓ **Activate virtual environment:**
- aws-audit-env\Scripts\activate

Writing the Audit Script (aws_security_audit.py)

This script:
- Detects public S3 buckets
- Detects disabled logging & versioning
- Detects public RDS instances
- Detects RDS without backups/delete protection
- Scans Security Groups for open SSH/MongoDB ports

Execution Guide

Follow steps for execution:

Step 1. Install requirements
- pip install -r requirements.txt

Step 2. Configure aws credentials
- aws configure

```
(aws-audit-env) PS C:\Users\[redacted]\aws-security-audit> aws configure
File association not found for extension .py
AWS Access Key ID [****************W33J]:
AWS Secret Access Key [****************z98N]:
Default region name [Default region name [None]: eu-north-1   # or your region]:
Default output format [Default output format [None]: json]:
(aws-audit-env) PS C:\Users\[redacted]\aws-security-audit>
```

Figure 1 - AWS Configuration setup

Step 3. Run audit

- python aws_security_audit.py

The audit script is divided into three main functions:

1. check_s3_buckets()
   - Lists buckets
   - Checks ACL for public access
   - Checks logging/versioning status

2. check_rds_instances()
   - Retrieves DB instances
   - Checks public accessibility
   - Checks backup retention & delete protection

3. check_security_groups()
   - Lists security groups
   - Checks inbound rules for 22 and 27017

4. main()
   - Calls all audit functions sequentially
   - Displays results in a formatted output

✓ **Sample console output:**

- Publicly accessible S3 buckets: aws-test-bucket
- Publicly accessible RDS instances: test-rds
- SG allowing SSH: sg-xxxx (insecure-sg)

```
(aws-audit-env) PS C:\Users\[redacted]\aws-security-audit> python aws_security_audit.py
Starting AWS security misconfiguration scan...

=== S3 Bucket Checks ===

Bucket: aws-test-bucket[redacted]
  [MISCONFIG] Bucket is publicly accessible
  [MISCONFIG] Logging is DISABLED
  [MISCONFIG] Versioning is DISABLED

=== RDS Instance Checks ===

DB Instance: test-rds[redacted]
  [MISCONFIG] DB is publicly accessible
  [MISCONFIG] Delete protection is DISABLED
  [MISCONFIG] Backups are DISABLED (RetentionPeriod = 0)

=== Security Group Checks ===

Security Group: insecure-sg[redacted] (sg-0284e10343c655497)
  [MISCONFIG] SSH (22) is open to the world (0.0.0.0/0)
  [MISCONFIG] MongoDB (27017) is open to the world (0.0.0.0/0)

Security Group: default (sg-0b7ab570ba0f09481)
  [OK] No obvious SSH/MongoDB public access issues found.

Scan complete.
(aws-audit-env) PS C:\Users\[redacted]\aws-security-audit>
```

Figure 2 - Script Output

Implementing Tests using Pytest

Tests use monkeypatch to mock AWS services and validate logic without needing real AWS calls.

- ✓ **Run tests:**
  - Pytest
- ✓ **Expected output:**
  - 4 passed in 0.66s #Time may change

```
(aws-audit-env) PS C:\Users\[redacted]\aws-security-audit> pytest -q
....
4 passed in 0.66s
(aws-audit-env) PS C:\Users\[redacted]\aws-security-audit>
```

Figure 3 - pytest output

Vulnerability Summary Table

ID	Service	Vulnerability	Severity	Impact / Risk	Recommended Mitigation
V1	S3	Public Bucket Access	High	Unauthorized data exposure	Block public access, audit bucket policy
V2	S3	Logging Disabled	Medium	No auditability / lack of forensic evidence	Enable S3 access logging
V3	S3	Versioning Disabled	Medium	No rollback from accidental overwrites/deletions	Enable versioning
V4	RDS	Publicly Accessible Database	Critical	High attack surface for DB exploitation	Move DB to private subnet, disable public access
V5	RDS	Delete Protection Disabled	High	Database can be removed accidentally	Turn on deletion protection
V6	RDS	Backups Disabled	High	No disaster recovery path	Enable automated backups
V7	Security Group	SSH Port 22 Open to 0.0.0.0/0	Critical	Brute-force attacks, unauthorized server access	Restrict SSH to trusted IP ranges
V8	Security Group	MongoDB Port 27017 Open to the World	Critical	Database takeover, data exfiltration risk	Restrict DB access to internal VPC only

---

Conclusion

This lab demonstrated:

- Setting up vulnerable cloud environments
- Automating audits using Python & boto3
- Implementing automated testing using pytest
- Understanding cloud security principles
- Documenting findings in a structured manner

Completed a full cloud security audit workflow similar to what security engineers use in real organizations.

~ END OF REPORT ~