Assume the Worst: Enumerating AWS Roles through ‘AssumeRole’ - Rhino Security Labs

By Spencer Gietzen

Published: 2018-08-29 · Archived: 2026-04-05 23:43:27 UTC

Disclaimer: As always, use Pacu and similar AWS pentesting tools responsibly.
Only test against your own AWS accounts, or those you are authorized for.

Introduction: AWS IAM Roles

Amazon Web Services (AWS) IAM roles are sets of permissions that serve as a common way to delegate access to users or services. Roles can be bestowed to internal and external IAM users, AWS services, applications, and even external user accounts outside of AWS. Roles leveraged by users enable additional permissions for certain tasks. Applications using roles gain the access needed to transact with AWS services without the need to permanently embed an AWS key. Anytime additional access needs to be temporarily granted, IAM roles are a common go-to solution.

Assuming IAM roles is the process of obtaining the set of permissions designated by the role, along with the corresponding temporary credentials. When an entity assumes a role, the Security Token Service (STS) issues a set of role credentials which serve as a security token to access the environment. Assuming a role can occur via the AWS Management Console, or programmatically via PowerShell, the AWS CLI, or the SDK’s for various programming languages. For more granular control, additional policies can also be applied to the request to further limit the role permissions. Requests made after the role is assumed are performed with this new identity, although CloudTrail logs can be used to discern the underlying entity.

AWS Account ID Exposure

AWS account IDs uniquely identify every AWS account and are more sensitive than you might think. While divulging the ID does not directly expose an account to compromise, an attacker can leverage this information in other attacks. A reasonable effort should be made to keep AWS account IDs private, but in practice, they are often exposed to the public unintentionally.

Account IDs are commonly leaked via:

- Screenshots
- GitHub and other code repositories
- Many AWS error messages (even access denied)
- Public EBS snapshots (EC2 -> Snapshots -> Public Snapshots)
- Public AMIs (EC2 -> AMIs -> Public images)
- RDS public snapshots (RDS -> Snapshots -> All Public Snapshots)
- People looking for troubleshooting help online and posting their ID

If an attacker has your ID, various attack scenarios become feasible, including resource enumeration (identifying existing roles, users, etc.), Lambda function invocation, and IAM role assumption.

This post — and the accompanying script we have released – address using an AWS account ID to identify existing roles. As an extension of this concept, attackers can go a step further and assume misconfigured IAM roles to gain unauthorized access.

IAM Role Enumeration

AWS verbose error messages disclose whether a role exists or not, allowing us to enumerate role names. During the testing process, we identified information leakage in the AWS responses, but we didn't know if an account would be locked out after multiple failed attempts to assume a role. We ultimately discovered that a user could produce an arbitrary number of requests, making this enumeration technique highly scalable. It should also be noted that if AWS was blocking an account after a certain period, this would make the brute forcing process more difficult, but not impossible. By running this attack through a fragmented set of nodes and accounts, an attacker can bypass any IP or account-based blocking that may be in effect.

Simply enumerating role names can reveal:

- which AWS services a company is using internally
- internal software/stacks
- names of IAM users (to target for social engineering)
- 3rd party integrations being used (which could potentially be targeted separately)
    - A few common integrations we have observed include CloudSploit, Okta, and Datadog.

Once roles are enumerated, one can try to assume any open roles and pilfer the role credentials.

If you try to assume a role that you don't have permissions to, AWS will output an error similar to:

```
An error occurred (AccessDenied) when calling the AssumeRole operation: User: arn:aws:iam::012345678
```

This error message indicates that the role exists, but its assume role policy document does not allow you to assume it. By running the same command, but targeting a role that does not exist, AWS will return:

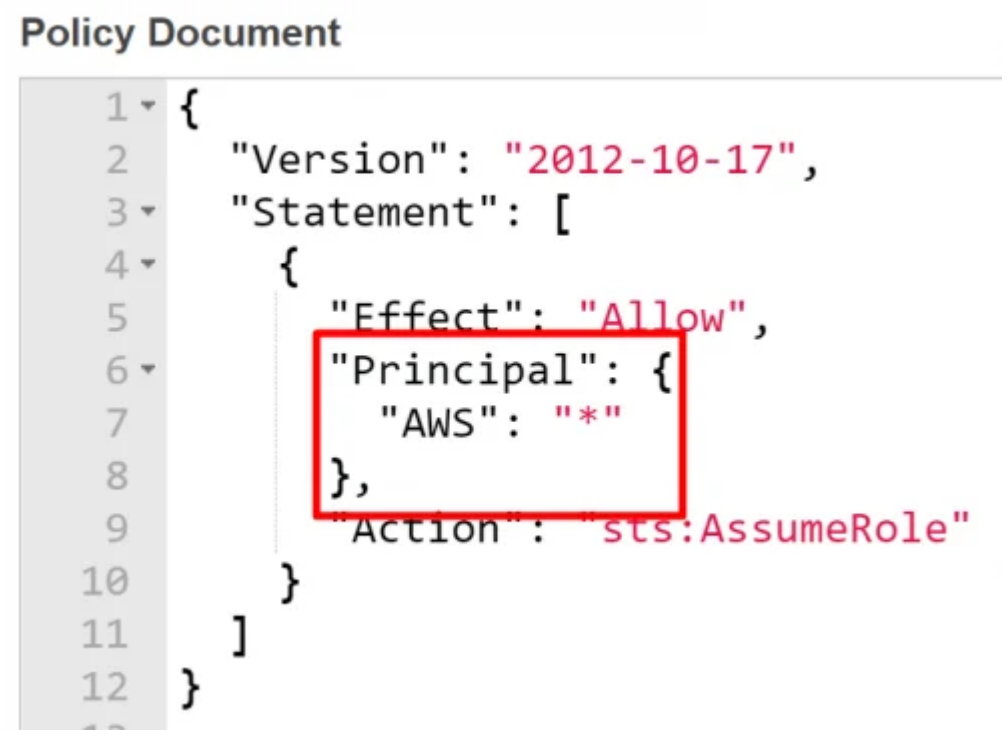
```
An error occurred (AccessDenied) when calling the AssumeRole operation: Not authorized to perform st
```

Surprisingly, this process works cross-account. Given any valid AWS account ID and a well-tailored wordlist, you can enumerate the account's existing roles without restrictions.

Hijacking Misconfigured Roles

Like other AWS IAM policies, the AssumeRole permissions are very flexible and, if misconfigured, could lead to unintended consequences. For example, if the access role *"AWS": "*"* is associated and any user from any account may be able to assume the role (given that they have the correct AWS Account ID and Role Name).

While this is the non-default state and a relatively rare occurrence, we've identified (and properly reported) upwards of 50 instances of accounts with global permissions for AssumeRole.



A poorly configured AssumeRole policy that allows any authenticated AWS user to assume it and retrieve temporary credentials.

AWS 'AssumeRole' Enumeration Script

We have created a standalone script to allow you to enumerate roles against a target account. By default, it comes with a 1100+ word wordlist with some common/generic role names. When a role is discovered, the script will alert you. If one is discovered and it is misconfigured to allow role-assumption from a wide group, the script is capable of automatically assuming the discovered role and outputting the issued credentials.

The script has just been released on GitHub. Pacu — the AWS exploitation framework — now also contains this functionality with a dedicated module.

All that is required is the account ID of your target account – they are all over the internet.

```
Usage: assume_role_enum.py [-h] [-p PROFILE] [-w WORD_LIST] -i ACCOUNT_ID
```

Note: The keys used must have the "sts:AssumeRole" permission on any resource (*) to be able to assume a misconfigured role. Lacking this permission, roles can still be identified, but not assumed.

Optional arguments:

```
  -h, --help            show this help message and exit
  -p PROFILE, --profile PROFILE
```

```
                        The AWS CLI profile to use for making API calls. This
                        is usually stored under ~/.aws/credentials. You will
                        be prompted by default.
  -w WORD_LIST, --word-list WORD_LIST
                        File path to a different word list to use. There is a
                        default word list with 1100+ words. The word list
                        should contain words, one on each line, to use to try
                        and guess IAM role names. Role names ARE case-
                        sensitive.
  -i ACCOUNT_ID, --account-id ACCOUNT_ID
                        The AWS account ID of the target account (12 numeric
                        characters).
```

The following screenshot demonstrates the tool in use in a mock environment. It begins enumerating roles and finds the restricted roles named “5” and “ADS”, as well as a misconfigured role named “APIGateway”. The script tries to assume the role for a minimum of one hour, then assumes it and displays the JSON-formatted role credentials. A summary of the results are then displayed.

```
PS C:\Users\      \Desktop\AssumeRoleEnum> py .\assume_role_enum.py --account-id 34      58 --profile default

Warning: This script does not check if the keys you supplied have the correct permissions. Make sure they are allo
wed to use sts:AssumeRole on any resource (*)! You can still enumerate roles that exist without the sts:AssumeRole
 permission, but you cannot assume (or identify) a misconfigured role.

Targeting account ID: 34      58

Starting role enumeration...

  Found restricted role: arn:aws:iam::34      58:role/5

  Found restricted role: arn:aws:iam::34      58:role/ADS

  ** Found vulnerable role: arn:aws:iam::34      58:role/APIGateway **
    Hit max session time limit, reverting to minimum of 1 hour...

  Successfully assumed role: arn:aws:iam::34      58:role/APIGateway

{
  "Credentials": {
    "AccessKeyId": "ASIAU7DSLMANDEQMCZL5",
    "SecretAccessKey": "dteGVIhL+tj+iwo/vgMGXjuM+/xtllBTufxEzFgR",
    "SessionToken": "FQoGZXIvYXdzEK7//////////wEaDK03iqs6GoTJgzqlCCL4AVTsLgFCTYbk6ZP8t+HOTqBTAfZx6WWQLRGGLwpvt2ohM
81Wq+P96OXQgnKLN/9vZQ8WqZRP0VnFXHxwsPE3DbpXE0+tlsKOhn6JR+OkWixkcQIRjqMor0VaWi0YChFJgXhrGXVNTuBolF2+ZKcMVn8BN2tunb+
56STqv+90Up4Z7C/jglEs83/WPav9E9P2KtsCkrh2W41GKsHyeLuULzXmaGjH+6q5JUvntSouYKAyTUP2G+bUXdoH2xtAlDSPge9JjJCc9dLzRoOFK
8J8mAIO3Q1fF2dL69ogBpg9pPutecSFobJEfFA1rPQOxYcXtAfZFHefvb7hKNXgltwF",
    "Expiration": "2018-08-28 21:28:05+00:00"
  },
  "AssumedRoleUser": {
    "AssumedRoleId": "AROAIQ          ZGY:oPznu4ufLRNguKHNgdEV",
    "Arn": "arn:aws:sts::34      58:assumed-role/APIGateway/oPznu4ufLRNguKHNgdEV"
  }
}
Found 2 restricted role(s):

    arn:aws:iam::34      58:role/5
    arn:aws:iam::34      58:role/ADS

.\assume_role_enum.py completed after 14 guess(es).

PS C:\Users\spenc\Desktop\AssumeRoleEnum>
```

Demonstrating using the script to successfully enumerate and assume a role. The issued role credentials are then displayed.

In addition to our default list of known users and roles to look for, check out this list that Daniel Grzelak (@dagrz) created, which includes many known 3rd party integrations.

Prevention

Unless AWS reduces the verbosity of the error messages returned after a failed role assumption, attackers will be able to enumerate role names using this method. Thankfully, mitigating an attack of this nature simply involves avoiding a few simple configuration errors.

Three actionable steps to take to protect your environment:

1. Require strict AssumeRole policies that specifically list only the AWS service or IAM user/account that needs the additional access
2. Don't use common, guessable names for your AWS resources
3. Avoid exposing your AWS account ID to the public

This bruteforcing technique and script will generate a large amount of "iam:AssumeRole" CloudTrail logs in the account you are using for enumeration. Any account you target will not see anything in their CloudTrail logs until you successfully assume a misconfigured role, so that means enumeration is completely log-free on the target account.

Conclusion

Assuming IAM roles in AWS is a common option for delegating permissions, but when improperly configured, this functionality can expose an AWS account to potential compromise.

The standalone enumeration script that automates this process is now available on GitHub.

If you are interested in leveraging other common AWS misconfigurations, try out Pacu – the open source AWS exploitation framework — which incorporates this script as a dedicated module, called "enum_assume_role".

If you need a safe, inexpensive AWS infrastructure to test out these new tools, consider CloudGoat — the vulnerable-by-design AWS environment.

Special thanks to Henry Hang for some great insight during our research into this enumeration method.
