ermetic

10 Steps to Eliminating Access Risks in AWS

Achieve least privilege and remove unused access to protect your cloud data

Least-Privilege is a Critical Step in Your Cloud Security Roadmap

In a recent survey of 300 CISOs across the US, more than 70% of respondents identified "least-privilege" as their most significant challenge

Securing identities and data in the cloud is a challenge for many organizations. Yet recent events have proven that **the risks associated with the compromise of identities and credentials cannot be taken lightly.** The problem becomes increasingly acute as organizations expand their cloud footprint without establishing the capability to effectively assign and manage permissions. As a result, **users and applications tend to accumulate permissions that far exceed technical and business requirements, which creates a permissions gap.**

The potential impact of lax authorization can be devastating. **Attackers can use compromised credentials with excessive permissions for misdemeanors like stealing sensitive data or deleting parts of the infrastructure.**

The Capital One breach is the perfect example of the risks of excessive access permissions, where a staggering 106 million credit card applications were exposed. The problem stemmed in part from a vulnerable open-source Web Application Firewall (WAF) that Capital One used as part of its operations hosted in the cloud with Amazon Web Services (AWS), its Cloud Security Provider (CSP). **This vulnerability allowed the attacker to obtain credentials to access any resource in the account to which that WAF had access.**

In AWS, exactly what access is associated with a set of credentials depends on the permissions assigned to the entity. In Capital One's case, **the vulnerable WAF was assigned excessive permissions; it could list all the files in any bucket of data and read the contents of each of those files.** These excessive permissions allowed the attacker to obtain access to a sensitive S3 bucket.

The Way to Least-Privilege: Theory and Practice

To mitigate risks associated with identity abuse in the cloud, organizations are trying to enforce the principle of least privilege. **Ideally, every user or application should be limited to the exact permissions required.**

In theory, the first phase to achieving least privilege is **understanding which permissions a given user or application has.** The next step is to **map all the permissions actually being used.** Comparing the two reveals **the permission gap to understand which permissions to keep and which to remove.** Finally, the permissions deemed excessive are removed or monitored and alerted on. By continuously re-examining the environment and removing unused permissions, your environment transitions to least privilege over time.

In practice, however, **the effort required to determine the precise permissions necessary for each application in a complex environment like AWS is prohibitively expensive and does not scale.** Even simple tasks like understanding the permissions granted to a single human user can be extremely complex.

Let us take AWS cloud as an example, since it is the most popular platform and it offers one of the most granular and complex Identity and Access Management (IAM) systems available. **AWS IAM is a powerful tool that allows you to securely configure access to AWS cloud resources.** With more than 2500 permissions (and counting), IAM gives users fine-grained control over which actions can be performed on a given resource in AWS.

Not surprisingly, **this degree of control introduces an unacceptable level of complexity for developers and DevOps teams.** In this paper, we explain the 10 steps to determining the access open to a single user or application. We will show why one must look beyond IAM policies to achieve least-privilege.

Assessing User Permissions

A simple task like understanding the permissions granted to a single human user can be extremely complex in a public cloud environment like AWS. We will present the process for determining such permissions, evaluate some available tools and discuss their limitations.

Step 1: Examine attached policies

The first step is to examine policies attached directly to the user. There are two types of policies:

- **Managed policies –** standalone policies that also come in two varieties:

  (a) AWS managed policies that are created and managed by the CSP, and (b) customer managed policies that you create and manage in your AWS account. Customer managed policies usually provide more precise policy control than AWS managed policies.

- **Inline policies –** policies that you create, manage, and embed directly into the user identity.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "s3:ListBucket",
    "Resource": "arn:aws:s3:::example_bucket"
  }
}
```

An AWS policy allowing the principal to list an S3 bucket

Step 2: Analyze IAM groups

The next step is to examine the IAM groups each user belongs to. These also have attached policies that indirectly grant a user access to additional resources. Just as with the user itself, groups may be attached to both managed and inline policies.

Step 3: Map IAM roles

Now, map all the IAM roles the user may assume. A role is yet another type of IAM identity that you can create in your account and attach policies that grant specific permissions. It is similar to an IAM user, but instead of being uniquely associated with a person, a role can be assumed by anyone who requires its permissions. Roles are often used to grant access permissions to applications.

Once you understand that the user may assume a role, you map all relevant roles and their respective policies (think of this as similar to Step 2, just replace the "groups" with "roles").

Step 4: Survey resource-based policies

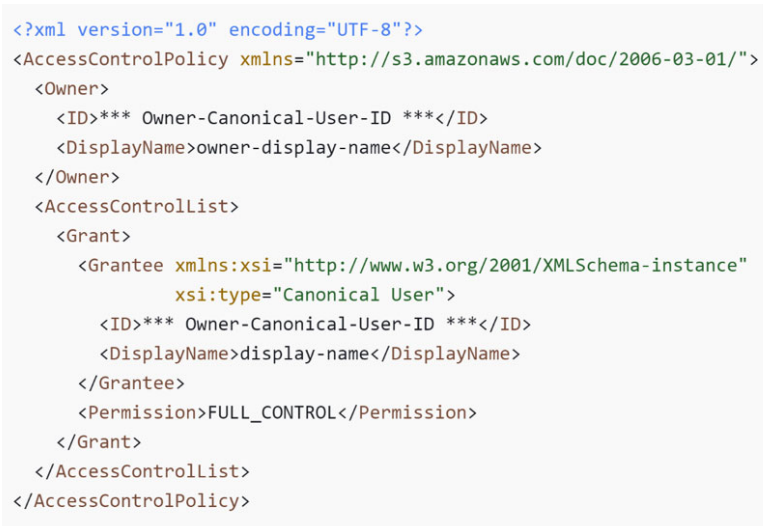
You are now ready to shift the focus from the user to the resources available in your AWS organization. The reason for the shift is resource-based policies (policy documents that you attach to a specific resource, such as an AWS S3 bucket). These policies may grant a user permission to perform actions on that bucket directly, independent of all the policies (direct and indirect) that you have analyzed so far.

To take stock of all resource-attached policies, you should conduct a comprehensive review of the resources in AWS, starting with the ones that contain important or sensitive data. Unfortunately, this task is tedious and time-consuming.

Step 5: Analyze access control lists

Continue analyzing resources by moving to Access Control Lists (ACLs). These are similar to resource-based policies and allow control over which identities in other accounts may access the resource. Since ACLs cannot be used to control access for identities within the same account, you can skip all resources held in the same account as the user.

But do not forget resources in other accounts within your AWS organization. Note that this is the only policy type that does not use the JSON policy format, and not all resources support ACLs.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>*** Owner-Canonical-User-ID ***</ID>
    <DisplayName>owner-display-name</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:type="Canonical User">
        <ID>*** Owner-Canonical-User-ID ***</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

A default ACL that grants an S3 bucket owner full control over the resource

Step 6: Review permission boundaries

Return to the user and review the permissions boundary. This is an advanced feature in which you set the maximum permissions a user, group or role may have. In other words, when you set a permission boundary for a user, he or she is able to perform only the actions that are allowed by both the attached policies and the permissions boundaries.

Be extremely careful with the details because permissions boundaries do not affect every policy the same way. For example, resource-based policies are not limited by the permissions boundary and an explicit deny in any of these policies overrides the allow.

Step 7: Check service control policies

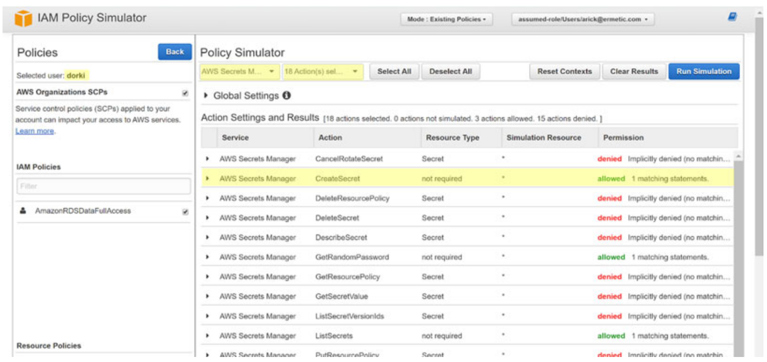
Finally, take a quick look at your service control policies. These are conceptually similar to permission boundaries defined on all the identities (i.e. users, groups, and roles) within your account. An SCP is defined at the AWS organization level, and you may choose the specific accounts to which the policy applies.

Last, but not least

All these policies may include both ALLOW and DENY statements, so be careful when combining the various statements and assessing the exact access an IAM user has to a specific resource.

Existing Tools and Limitations

You may be wondering if there any way to automate this process. A few years ago, AWS released a tool called Policy Simulator. Policy Simulator allows you to select any AWS entity (i.e. an IAM user, group, or role) and service type (e.g. a Relational Database Service or an S3 bucket) and automatically assess the user permissions for a specific service.



Selected user: dorki

AWS Organizations SCPs

Service control policies (SCPs) applied to your account can impact your access to AWS services. Learn more.

IAM Policies

AmazonRDSDataFullAccess

Resource Policies

Action Settings and Results [18 actions selected. 0 actions not simulated. 3 actions allowed. 15 actions denied.]

Service	Action	Resource Type	Simulation Resource	Permission
AWS Secrets Manager	CancelRotateSecret	Secret	*	denied Implicitly denied (no matchin...
AWS Secrets Manager	CreateSecret	not required	*	allowed 1 matching statements.
AWS Secrets Manager	DeleteResourcePolicy	Secret	*	denied Implicitly denied (no matchin...
AWS Secrets Manager	DeleteSecret	Secret	*	denied Implicitly denied (no matchin...
AWS Secrets Manager	DescribeSecret	Secret	*	denied Implicitly denied (no matchin...
AWS Secrets Manager	GetRandomPassword	not required	*	allowed 1 matching statements.
AWS Secrets Manager	GetResourcePolicy	Secret	*	denied Implicitly denied (no matchin...
AWS Secrets Manager	GetSecretValue	Secret	*	denied Implicitly denied (no matchin...
AWS Secrets Manager	ListSecretVersionIds	Secret	*	denied Implicitly denied (no matchin...
AWS Secrets Manager	ListSecrets	not required	*	allowed 1 matching statements.
AWS Secrets Manager	PutResourcePolicy	Secret	*	denied Implicitly denied (no matchin...

Identifying allowed and denied permissions in the IAM Policy Simulator

Although Policy Simulator is a great tool, it is far from mature. For example, Policy Simulator does not review all the roles a user may assume and their policies (Step 3). It also does not consider ACLs (Step 5) or permission boundaries (Step 6).

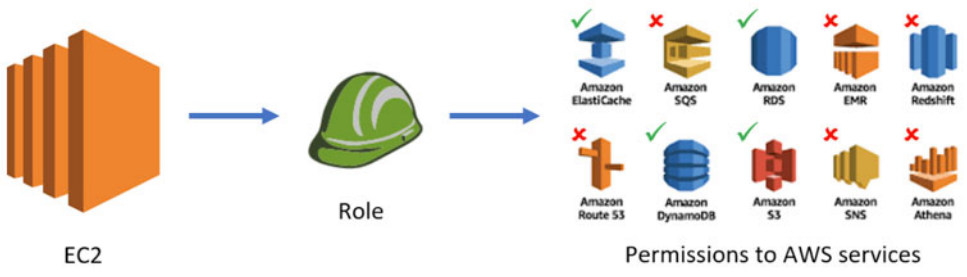
The Policy Simulator becomes even less useful if you decide to reverse the query and assess which users may directly or indirectly access a specific sensitive resource (e.g. an S3 bucket). Eventually, you must revert to doing things manually or writing proprietary scripts.

Working Toward Least-Privilege Access Policies

Now that you have mastered the process of evaluating existing permissions, we will shift gears to achieving least privilege. Continuous access profiling and role rightsizing sounds like a straightforward approach and natural next step; however, simple role-based access is not enough. Now let us analyze several common scenarios, continue to evaluate available tools, and review their limitations.

Step 8: Align a Single Application with a Single Role

Let us start with a simple example. An application assumes a role with excessive privileges, and the role has permission to access Amazon ElastiCache, RDS, DynamoDB, and S3 services:



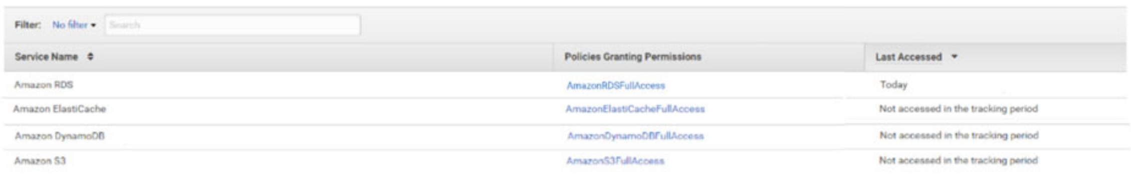
Specific application assumes role

How do you know which permissions are actually being used?

Introducing Access Advisor

AWS Access Advisor is a useful tool that allows you to investigate the list of services accessed by a given role and verify how it is being used. In this example, when we use Access Advisor to inspect the services accessed by Role, we see that Amazon RDS was the only service accessed during the past 90 days.

In general, you can review service last accessed information for your AWS organization in the IAM console in your organization's master account or programmatically using IAM access advisor APIs with the AWS Command Line Interface (AWS CLI) or a programmatic client.



Filter: No filter ▾ Search

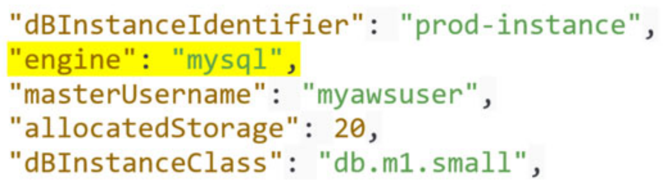
Service Name	Policies Granting Permissions	Last Accessed
Amazon RDS	AmazonRDSFullAccess	Today
Amazon ElastiCache	AmazonElastiCacheFullAccess	Not accessed in the tracking period
Amazon DynamoDB	AmazonDynamoDBFullAccess	Not accessed in the tracking period
Amazon S3	AmazonS3FullAccess	Not accessed in the tracking period

Only RDS service is accessed by the specific role

The simplest way to achieve least-privilege in our current scenario would be to right-size the permissions i.e. remove the unused services from this specific role.

Dig into CloudTrail logs

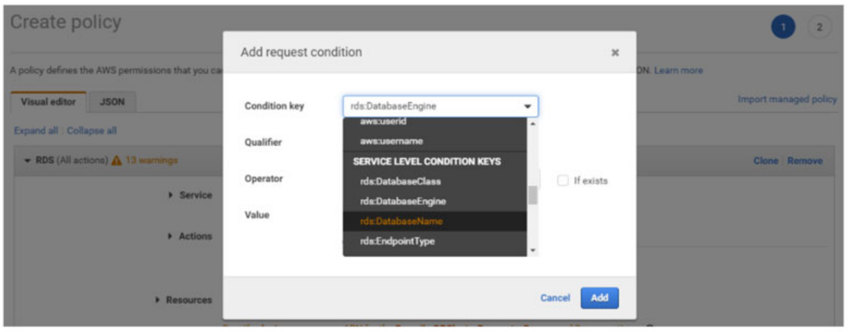
This is a good start, but clearly not enough. When you inspect the actual activities performed by the role and leverage the AWS CloudTrail service, you see the only RDS resource accessed is a MySQL DB:



```
"dBInstanceIdentifier": "prod-instance",
"engine": "mysql",
"masterUsername": "myawsuser",
"allocatedStorage": 20,
"dBInstanceClass": "db.m1.small",
```

Snapshot form CloudTrail log entry

The next step is to limit the permissions to allow access to only MySQL resources (as opposed to all resources that belong to the RDS service type). The other option is to further limit permissions by restricting the access policy to a specific database instance:


Create policy

A policy defines the AWS permissions that you ca… …ON. Learn more

Visual editor | JSON | Import managed policy

Expand all | Collapse all

RDS (All actions) ⚠ 13 warnings | Clone | Remove

Service, Actions, Resources

Add request condition

Condition key: rds:DatabaseEngine

Qualifier

Operator

Value

aws:userid
aws:username
SERVICE LEVEL CONDITION KEYS
rds:DatabaseClass
rds:DatabaseEngine
rds:DatabaseName
rds:EndpointType

If exists

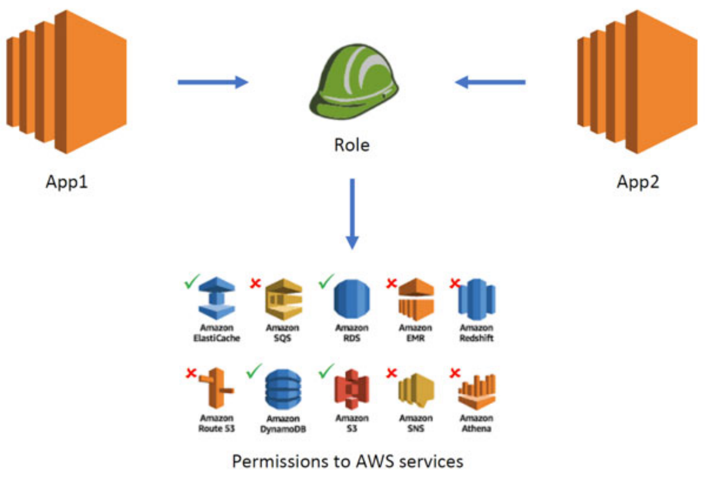
Cancel | Add

Restricting permission access based on database engine name

Of course, relying solely on the Access Advisor does not connect the dots between access permissions and individual resources for you to make this kind of policy decision, but it is still the best tool to start with.

Step 9: Manage two applications with one role

Let us consider a situation where two different applications have assumed the same role. This specific role has access permissions to Amazon ElastiCache, RDS, DynamoDB, and S3 services.

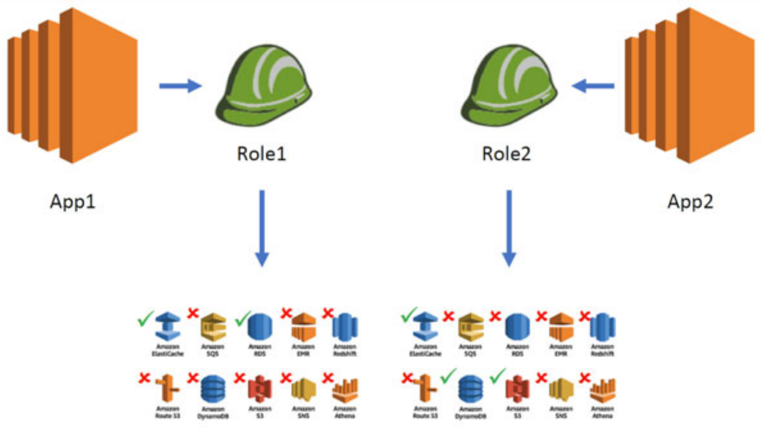


Two different applications assuming the same role

From here, determine whether the applications require access to all the resources and what type of access makes sense. You can adjust the permissions as required. In this case, however, there is an additional issue to be addressed.

Role splitting

In this case, each application uses a different set of services. While App1 uses RDS and ElastiCache services, App2 uses ElastiCache, DynamoDB and S3. Therefore, to achieve least-privilege, the correct action would not be simple role rightsizing, but role splitting followed by rightsizing.



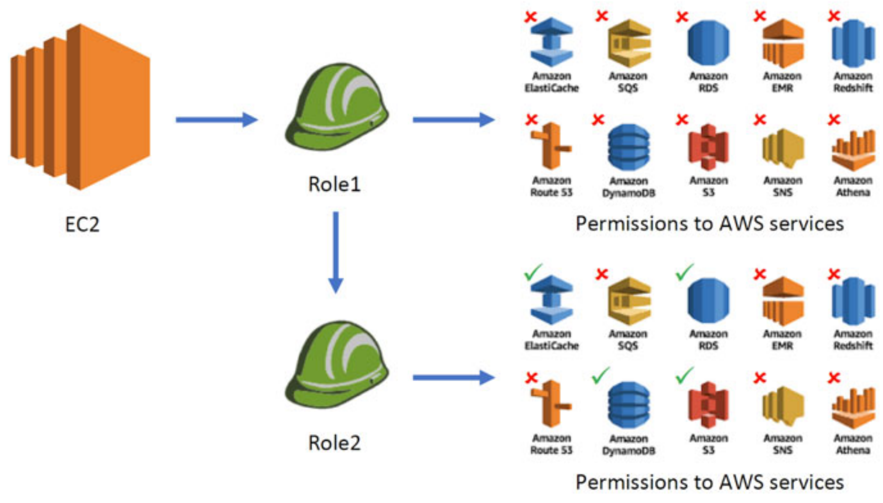
Splitting one broad role into two, fine-grained roles

To correctly analyze the situation and determine the best policy, dig deep into the CloudTrail logs and the compute management infrastructure like ECS, EKS, or native Kubernetes. This will enable you to correlate access actions to specific services with the compute resource running the relevant applications (App1 and App2).

A similar situation often occurs with federated users (i.e. users defined on an external, trusted Identity Provider (IdP) system like Okta, G-suite, or Active Directory). When these users connect to the AWS environment, they automatically assume a role based on the IdP definitions, and it is common for multiple users to assume the same role. As in the previous example, to correctly analyze and resolve the situation, you correlate CloudTrail and IdP logs. This is not easy, both in terms of data collection and analytics.

Step 10: Untangle role chains

Let us consider the scenario where an application assumes a role (Role1) that does not have any sensitive permissions. Role1 does have permission to assume a different, more privileged role (Role2), which has permission to access a variety of services like Amazon ElastiCache, RDS, DynamoDB, and S3.



Role1 has permissions to assume a privileged Role2

This kind of role chain can potentially consist of more than two roles. To really understand and right-size access permissions, you must be able to map the role chains within and between AWS accounts.

Conclusion

Intuitively, achieving least-privilege does not appear to be too complex. One should simply monitor the permissions each identity (i.e. user or application) actually uses, compare it to the permissions it has and use profiling data to right-size roles and permissions. By continuously re-examining the environment and removing unused permissions, your environment transitions to least privilege over time.

Unfortunately, this is easier said than done. Continuous access profiling and role rightsizing sounds like a straightforward approach, but simple role-based access is not enough to achieve least privilege. Even a simple task like understanding the permissions granted to a single human user can be extremely complex.

In this document we explained **how to evaluate permissions** users or applications have, and how to use this information in the road to least-privilege. We have analyzed several common scenarios, introducing **the concepts of role rightsizing, role splitting and role chaining.** We have examined several important tools like Policy Simulator, Access Advisor, and CloudTrail, we learned how to leverage these tools to achieve least-privilege, and we discussed their limitations.

This is just the beginning of the road to least privilege; the scenarios presented do not cover the full picture. So far, we have touched only on native IAM access controls. There are several additional issues one should consider when mapping access permissions to resources, for example indirect access or resource-level access controls. There are also additional resources like Key Management Systems or Parameter Stores, and certainly, there is more to automation than we have discussed so far.

Eliminate excess privileges simply, at scale.

Ermetic leverages continuous visibility into identities, entitlements and data to power access policy definition, automation, and enforcement at scale. Through advanced usage analytics, Ermetic aligns permissions with actual business needs to reduce the attack surface, and flag suspicious activity.

CONTINUOUSLY ENFORCE LEAST-PRIVILEGE ACCESS TO CROWN JEWEL DATA

Visibility
Discover all human and machine identities, data and compute resources, roles and policies.

Analytics
Analyze all access policies and activity to model and identify risks while ensuring business continuity.

Policy Enforcement
Eliminate excessive access and privileges based on actual access patterns and data sensitivity.

Anomaly Detection
Flag suspicious behavior such as sensitive data access, privilege escalation and deletion.

Automation
Generate access policies that optimize security and enable productivity.

Compliance
Monitor and report compliance violations against leading industry standards.

www.ermetic.com