OCI IAM Deny Policies Best Practices

Best Practices for Using OCI IAM Deny

Oracle Cloud Infrastructure (OCI) Identity and Access Management (IAM) Deny policies enable administrators to explicitly block unwanted actions, enhancing security and streamlining access control. These best practices guide enterprises to deploy OCI IAM Deny effectively, streamline operations, and meet regulatory requirements in complex tenancies.

1. Enable IAM Deny and Preserve the Default Policy

What is it?: IAM Deny must be explicitly enabled by a default administrator in the OCI Console before anyone can write a Deny statement. When enabled, OCI automatically seeds a **default Deny policy** in the root compartment (C0) that initially blocks all users from writing Deny statements except the default administrators group and the user enabling the feature.

Default Seeded Policy:

```
deny any-user to manage policies in tenancy where all {target.policy.type = 'deny',
request.principal.id != '<caller's ocid>'}
```

This policy ensures that only authorized tenancy administrators can modify or create Deny policies after enablement.

Do **not delete** this default system policy. Instead, update it carefully to **exclude only the groups that truly require Deny policy authorship** (for example, central security or compliance teams). This preserves tenancy-wide protection while granting limited authoring capabilities.

Why It Helps?: Prevents accidental lockouts or uncontrolled Deny policy creation by restricting authorship to trusted administrators. Maintaining the default policy as the foundation of Deny control ensures a secure baseline for your tenancy.

2. Define Deny Policies at Root for Tenancy Wide Control

What is it?: If your goal is to enforce tenancy-wide guardrails, write clear, specific deny policies at the **root compartment (C0)**. Doing this allows uniform enforcement across all compartments and simplifies auditing. Avoid overly broad policies that could disrupt tenancy operations.

Example:

A multinational enterprise seeks to secure tenancy-wide access by restricting unauthorized actions.

```
Deny group NonAdmins to use all-resources in tenancy where request.service = 'functions'
Deny group NonAdmins to manage volume-family in tenancy
Allow group Administrators to manage all-resources in tenancy
```

These policies block `NonAdmins` from using `Functions` and managing storage tenancy-wide, while ensuring Administrators retain access, securing the tenancy without disruptions.

Why It Helps?: This secures the tenancy with consistent, centralized access controls while minimizing risk of accidental outages caused by overly broad or ambiguous policies.

3. Restrict Sub-Admins Securely with Deny Policies

What is it?: To safely delegate tasks in large tenancies, use targeted deny policies to prevent users who can manage policy in a child compartments from accessing or managing sensitive resources, even if they have broader allow privileges elsewhere.

Example:

A public sector organization seeks to delegate compute tasks while safeguarding network configurations.

```
Allow group ComputeAdmins to manage instance-family in compartment Operations
Deny group ComputeAdmins to manage network-family in compartment Operations
```

These policies permit `ComputeAdmins` to manage compute resources but prohibit network modifications in the `Operations` compartment, ensuring secure delegation.

Why It Helps?: Enables fine-grained delegation— users who can manage policy in a child compartment can perform allowed operations (e.g., compute), while sensitive areas like networking remain protected.

4. Specify Granular Conditions for Precise Restrictions

What is It?: Use where clauses in deny policies to apply specific restrictions (e.g., by service, tag, or region). Avoid tenancy-wide allow conditions unless required.

Example:

A financial institution needs to comply with GDPR data residency requirements by restricting object storage access in a specific region.

```
Deny group Analysts to manage object-family in tenancy where request.region='sa-saopaulo-1'
```

This policy prevents the `Analysts` group from managing object storage in the `São Paulo` region, ensuring compliance with regional restrictions.

Why It Helps: Enforces regional compliance (like GDPR) and avoids blocking legitimate access in other areas—ensuring secure yet flexible access.

5. Simplify Deny Policies for Clarity

What is it?: Write clear, straightforward deny policies with minimal conditions to ensure auditable and understandable access controls, avoiding complex WHERE clauses like multiple != operators that can confuse users. Specific policies reduce ambiguity and misconfiguration risks.

Example:

A healthcare provider seeks to secure patient data by restricting unauthorized storage access and ended up writing a policy with multiple WHERE clauses, especially != operators, it created confusion about what is blocked. For example:

Deny group TemporaryUsers to manage volume-family in compartment HealthData where all in { request.source.ip != '192.168.1.0/24' request.user.name != 'Admin1' request.region = 'us-ashburn-1', request.service = 'volume'}

Instead, use clear deny statements:

Deny group TemporaryUsers to manage volume-family in compartment HealthData
Deny group TemporaryUsers to use volume-family in compartment HealthData
Allow group StorageAdmins to manage volume-family in compartment HealthData

These policies clearly block TemporaryUsers from managing or using storage in the HealthData compartment while permitting StorageAdmins, ensuring secure access control.

Why It Helps: Reduces confusion and makes the policy easier to audit, troubleshoot, and maintain.

6. Protect Against Lockouts with Controlled Deny Authorship

What is It?: If users who can manage policy in a child compartment do not require permission to author deny policies in OCI IAM, we recommend implementing a policy to prevent them from doing so, ensuring tenancy security and avoiding potential misconfigurations.

Example:

```
Deny group SubAdmins to manage policies in tenancy where target.policy.type='DENY'
```

This policy restricts users who can manage policy in a child compartment from creating or modifying deny policies while allowing them to manage allow policies, reducing the risk of unintended lockouts.

Why It Helps?: Reduces the chance of users who can manage policy in a child compartment unintentionally disabling access for themselves or others—ensuring that deny policy creation stays in the hands of central users who can manage policies if not needed by who can manage policy in a child compartment.

7. Avoid 'Allow All' Pitfall

What it is?: Be cautious when combining **broad allow policies** (e.g., Allow all-resources) with **narrow deny policies** targeting specific services (e.g., Deny access to streaming). This pattern may seem to work today, but it's fragile: as new OCI services are added, your broad allow policy will **automatically grant access** to those services—even if your original intent was to exclude them. Deny statements only block what they explicitly name and **do not automatically cover future services**.

Example:

```
Allow any-user to manage all-resources in tenancy
Deny any-user to inspect network-family in tenancy
```

This setup blocks networking today but **grants access to every current and future service**—except networking. If OCI launches a new service (e.g., "quantum-family"), users would instantly get access to it, potentially violating your security intent.

A more secure pattern:

```
Allow group DevTeam to use instance-family, object-family in compartment Sandbox
Deny group DevTeam to manage instance-family in compartment Sandbox
Deny group DevTeam to inspect object-family in compartment DataStorage
```

This setup limits access to only the services you've vetted, and prevents privilege creep as new services are added.

Why it helps?: This avoids the risk of **unintended access to new services** introduced after your policy was written. OCI's IAM model doesn't treat "deny X" as "allow all future unknowns except X." To stay aligned with least privilege and security best practices:

- Avoid "Allow all-resources" unless absolutely necessary
- Explicitly list the services you intend to allow

- Review and update allow policies regularly as OCI adds new services

This approach gives you full control over what your users can access today—and what they'll access tomorrow.

8. Understand Permission Hierarchy in Deny Policies

What is it?: Before writing OCI IAM policies, it's critical to understand the OCI permission hierarchy: `inspect, read, use, and manage.` Allow policies are `additive`, granting the specified permission and all lower levels (e.g., Allow manage permits manage, use, read, and inspect). Deny policies are `subtractive`, blocking the specified permission and all higher levels (e.g., Deny manage blocks only manage, allowing inspect, read, and use; Deny inspect blocks all permissions). Misunderstanding this hierarchy can lead to overly restrictive policies that mimic tenancy-wide lockouts (e.g., denying inspect when only manage was intended) or unintended access. For example, a broad allow policy paired with an insufficient deny policy may permit access to unintended resources. To avoid these pitfalls, carefully select permission levels and test policies using the OCI IAM Policy Simulator.

Example:

A research university needs to grant its research team limited access to Object Storage buckets in a data analysis compartment for processing datasets, while preventing modifications or management actions.

```
Allow group ResearchTeam to read bucket-family in compartment DataAnalysis
Deny group ResearchTeam to use bucket-family in compartment DataAnalysis
```

These policies allow the `ResearchTeam` to read data from buckets in the `DataAnalysis` compartment (e.g., viewing dataset contents) but prohibit using or managing buckets (e.g., uploading files or deleting buckets), while still permitting inspection (e.g., listing buckets). By specifying read in the allow policy and use in the deny policy, the university ensures precise control, avoiding overly restrictive denials (e.g., Deny inspect would block all access) or unintended permissions. Regularly review policies and audit logs to maintain alignment with access requirements. For more details, see the OCI IAM documentation.

Why It Helps: Facilitates precise access control by leveraging the permission hierarchy, preventing unintended restrictions or permissions.

9. Steer Clear of Common Deny Policy Pitfalls

What it is?: Avoid deny patterns that accidentally block legitimate users or create outages. Broad targets like any-user, root compartments, or tag-based deny conditions must be used with caution.

Examples of what *not* to do: The following customer pitfalls illustrate common errors and their consequences:

- **Tenancy-Wide Access Restriction**: A policy like `Deny any-user to inspect all-resources in tenancy` blocks all users, including teams, sub-admins, and applications, from accessing any resources, causing a tenancy-wide outage. This often stems from an intent to secure the tenancy but overlooks the broad impact of any-user.
- **Compartment-Wide Access Restriction**: A policy such as `Deny any-user to inspect all-resources in compartment Operations` halts all access within the compartment, including for the policy's author, disrupting workflows. The goal is typically to block unauthorized users, but the broad any-user scope affects legitimate users.
- **Tag-Based Lockouts**: A policy like `Deny group DevTeam to inspect compute-family in tenancy where target.resource.tag.Env='Prod'` locks out the group from all prod-tagged compute resources across all compartments, stalling critical tasks. This arises from misunderstanding the tenancy-wide reach of tag-based conditions.
- **Policy Management Loss**: A policy such as `Deny group RegionalAdmins to inspect policies in compartment Operations` prevents the authoring group, including the author, from managing policies, requiring

default admin intervention. This occurs when who can manage policy in a child compartment aim to restrict peers but inadvertently lock themselves out.

We advise customers to exercise caution when writing policies with tenancy-wide impact, such as those using `any-user` or targeting `all-resources` at the `root` compartment. While default administrator groups can intervene to update policies and unlock the tenancy, carefully designing policies avoids disruptions. To prevent these pitfalls, use group-specific policies (e.g., `Deny group Guests to inspect all-resources in tenancy`), scope conditions to compartments (e.g., `Deny group DevTeam to inspect compute-family in compartment Prod`), restrict deny policy authorship to default admins, and test policies with the OCI IAM Policy Simulator.

Why it helps: Avoids tenancy-wide outages, lockouts, and loss of policy control. Granular scope keeps policies safe and predictable.

10. Use Caution when Combining IAM Deny Statements conflicting With IDCS Admin Roles

What is it?: In the current OCI release, **IAM Deny policies do not override IDCS administrative roles** such as *Identity Domain Administrator, Security Administrator,* or *User Manager*. This limitation can unintentionally weaken your security guardrails. For instance, even if an IAM Deny policy is in place to block user creation in a specific compartment, a user with an IDCS admin role can still create users within the identity domain—bypassing the policy and violating its intended purpose. Until a future release enables IAM Deny policies to take precedence over IDCS roles, **use caution when applying these controls together** to avoid unintended access and ensure alignment with your tenancy's security requirements. We expect to address this limitation in a future release.

Example: Even if you write

```
Deny group admins to manage users in tenancy
```

An IDCS admin can still create users.

Why It Helps: Prevents security assumptions that aren't true in current OCI behavior. Plan for these limitations and consider alternative guardrails until deeper IAM-IDCS integration is available.

Get Started with OCI IAM Deny

Adopt these best practices to deploy OCI IAM Deny policies with confidence, avoiding pitfalls that lead to outages or unintended access. For detailed guidance, refer to the OCI IAM documentation. Contact your OCI account team for support.