S3 changes to bucket authorization are coming in June 2023

Amazon S3 is updating its bucket policy evaluation to make it consistent with how similar policies are evaluated across AWS. If you have a statement in one of your bucket policies whose principal refers to the account that owns the bucket, this statement will now apply to all IAM Principals in that account instead of applying to only the account's root identity. This update will affect buckets in at least one of your AWS accounts. This update will be rolled out in all AWS Regions in June, 2023.

We are requesting that you review your bucket policies and access patterns, and update your bucket policies before June, 2023, to ensure that they correctly reflect your expected access patterns in S3.

Examples for common permission scenarios that you can use today to update your bucket policies are included at the end of this message.

A list of your S3 buckets in the US-EAST-1 Region which will be affected by the update to bucket authorization, is available in the 'Affected resources' tab.

Summary of update to bucket authorization:

The authorization update applies to S3 bucket permission statements that follow this pattern:

```
{
  "Effect": "Deny",
  "Principal": {
    "AWS": $BUCKET_OWNER_ACCOUNT_ID
  },
  "Action":
  "Resource":
  "Condition":
}
```

Currently, for the previously listed buckets, S3 applies this statement only to requests made by the bucket-owning account's root identity, and not to IAM Principals within the bucket-owning account. The authorization behavior update will make S3 consistent with policy evaluation in other AWS services. With this update, the previous statement applies to all IAM Principals in the bucket-owning account.

Recommended policy updates:

The following recommendations will work both with the current and updated authorization behavior in S3, and therefore you can implement them today. We strongly recommend testing them on a test bucket before putting them on a bucket that serves a production workload.

Usually, the "Principal": {"AWS": $BUCKET_OWNER_ACCOUNT_ID} is not necessary at all. Since the intention of these bucket policy statements is to assert some behavior across all callers, it can usually be replaced by "Principal": "*".

// Recommended alternative 1: This statement will apply to all callers

```
{
  "Effect": "Deny",
  "Principal": "*",
  "Action":
  "Resource":
  "Condition":
}
```

In the less common case where your intention is for the statement to apply specifically to IAM Principals in the bucket-owning account, the following permission statement will work:

// Recommended alternative 2 (less common): This statement will apply to all callers from the bucket-owning account

```
{
  "Effect": "Deny",
  "Principal": "*",
  "Action":
  "Resource":
  "Condition": {
    "StringEquals": {
      "aws:PrincipalAccount": $BUCKET_OWNER_ACCOUNT_ID
    },
    ...
  }
}
```

Important: Avoiding S3 bucket lockout

A Deny statement in an S3 bucket policy that is unintentionally overly broad can result in Access Denied errors for all users in the account, including actions such as s3:PutBucketPolicy that might be needed to remediate the accidentally-broad statement. Because the

s3:PutBucketPolicy action can be taken only by IAM Principals within the same account as the bucket, and never by external accounts, it is not necessary to deny access to this action outside the account. However, it is possible to deny this action within the same account, and when that is applied overly broadly, bucket lockout can occur.

You can remediate bucket-lockout situations by signing in with your account's root identity and updating the bucket policy.

To avoid this situation in the first place, we recommend as a best practice that you avoid using Deny statements that cover all S3 actions (s3:*) on the bucket resource (arn:aws:s3:::example-bucket). While these statements will work as desired when the Conditions elements are correctly configured, a mistake in specifying those Conditions will result in a full lockout. In particular, a statement such as the following one will prevent all further changes to the S3 bucket's policy except by the account's root identity.

// DO NOT USE - WILL RESULT IN BUCKET LOCKOUT

```
{
  "Effect": "Deny",
  "Principal": "*",
  "Action": "s3:*",
  "Resource": "arn:aws:s3:::example-bucket"
}
```

Furthermore, S3 recommends against the use of the NotPrincipal element in IAM statements. Most common permission scenarios can be implemented more straightforwardly with conditions, as detailed in the previous examples. For more information on the 'NotPrincipal' element, please refer the IAM documentation page [1].

If you have questions or concerns, please contact AWS Support [2].

The following are common scenarios and examples:

1) Scenario: Block access to an S3 bucket's data from outside a list of desired AWS accounts:

A common scenario is to add a bucket policy statement that asserts that account(s) outside a given list cannot access the data in an S3 bucket.

We recommend a bucket policy statement like the following one:

```
{
  "Sid": "BlockAccessToDataOutsideAllowedAccountList",
  "Effect": "Deny",
```

```
  "Principal": "*",
  "Action": "s3:*",
  "Resource": "arn:aws:s3:::example-bucket/*",
  "Condition": {
    "StringNotEquals": {
      "aws:PrincipalAccount": [ "111111111111", "222222222222", ... ]
    }
  }
},
```

This statement grants no access, you would need Allow statements (not shown) to grant access for example to account 222222222222 - but will block data access from accounts not on this list, regardless of what other policy statements or ACLs are present.

It is not correct to use 'aws:SourceAccount' or 'aws:SourceArn' to achieve this goal. These IAM values are used for a different use case, such as, one in which an AWS service is accessing your bucket. For limiting access to specific IAM Principals, we recommend using the 'aws:PrincipalArn' condition. Please refer the documentation on all IAM global conditions, including the above [3].

2) Scenario: Block access to an S3 bucket's data from outside known network locations (IP or VPC):

A common scenario is to add a bucket policy statement that prevents interaction (read/write) of the bucket's data from outside a list of known network locations. These locations might be public IPv4 address ranges, if using the general public endpoint of the S3 service, or particular Virtual Private Clouds (VPCs), when reaching S3 by way of a VPC Endpoint.

A bucket policy statement like the following one will prevent any access to objects in the bucket, except from the allowed network paths:

```
{
  "Sid": "BlockAccessToDataOutsideAllowedNetworkLocations",
  "Effect": "Deny",
  "Principal": "*",
  "Action": "s3:*",
  "Resource": "arn:aws:s3:::example-bucket/*",
  "Condition": {
    "NotIPAddressIfExists": {
      "aws:SourceIp": [ "1.2.3.0/24", ... ]
    },
    "StringNotEqualsIfExists": {
      "aws:SourceVpc": [ "vpc-111", "vpc-222", ... ]
```

```
    }
  }
},
```

If you are also trying to block List operations from outside the specified network paths, you can add this statement.

```
{
  "Sid": "BlockListOperationsOutsideAllowedNetworkLocations",
  "Effect": "Deny",
  "Principal": "*",
  "Action": "s3:ListBucket",
  "Resource": "arn:aws:s3:::example-bucket",
  "Condition": {
    "NotIPAddressIfExists": {
      "aws:SourceIp": [ "1.2.3.0/24", ... ]
    },
    "StringNotEqualsIfExists": {
      "aws:SourceVpc": [ "vpc-111", "vpc-222", ... ]
    }
  }
},
```

- Note the specific s3:ListBucket action; had the action instead been a generic s3:*, this policy would also have denied all actions on the bucket from outside those network paths, which can be undesired. For example, the ability to make further changes to bucket policy. See the previous discussion on avoiding bucket lockout.

3) Scenario: Blocking unencrypted uploads:

S3 offers a "default encryption" option for buckets that allow a customer to designate a desired Server-Side Encryption (SSE) scheme, either SSE-S3 or SSE-KMS. Because individual PutObject requests in S3 can specify schemes different from the bucket default, a common scenario for customers who want to be certain that no unencrypted data can be uploaded is to write an S3 bucket policy that blocks s3:PutObject from succeeding, unless the desired SSE scheme will be used for the object.

The following example bucket policy statement asserts that, in order to succeed, any PutObject request to this bucket must be SSE-KMS encrypted with a KMS key from account 111122223333.

```
{
  "Sid": "BlockNonKMSUploads",
```

```
  "Effect": "Deny",
  "Principal": "*",
  "Action": "s3:PutObject",
  "Resource": "arn:aws:s3:::example-bucket/*",
  "Condition": {
    "ArnNotLikeIfExists": {
      "s3:x-amz-server-side-encryption-aws-kms-key-id":
"arn:aws:kms:us-east-1:111122223333:key/*"
  }
}
}
```

Other common patterns: The condition s3:x-amz-server-side-encryption can be used for a more generic permission statement to require SSE-S3 (AES256) or SSE-KMS (aws:kms) respectively. A few other example policies for encryption are available in our Knowledge Center article [4].

Although the aws:PrincipalAccount IAM condition can be used to limit the effect of these policy statements to particular AWS accounts, that is usually unnecessary for the common case in which you are simply trying to assert that all data that gets written is encrypted in the desired way.

[1] https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements_notprincipal.html
[2] https://aws.amazon.com/support
[3] https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_condition-keys.html
[4] https://aws.amazon.com/premiumsupport/knowledge-center/s3-bucket-store-kms-encrypted-objects/

Sincerely,
Amazon Web Services