Establishing guardrails and monitoring for presigned URLs

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Establishing guardrails and monitoring for presigned URLs

Table of Contents

Introduction **1**
Best practices 1
Intended audience 2
Objectives 2
Prerequisites 2
Overview of presigned URLs **4**
Motivations for using presigned requests 5
Comparison with temporary AWS STS credentials 6
Comparison with signature-only solutions 6
Identifying presigned requests **7**
Identifying requests that used a presigned URL 7
Identifying other types of presigned requests 8
Identifying request patterns 8
Best practices for using presigned requests **13**
Foundational best practices 13
Apply the principle of least privilege 13
Implement a data perimeter 14
Additional guardrails 14
Guardrail for s3:signatureAge 14
Resource control policies 18
Guardrail for s3:authType 19
Combining presigned guardrails and exceptions to other guardrails 21
Limitations to s3:signatureAge 22
Targeting buckets at scale 23
Logging interactions and mitigations 23
Mitigations 24
FAQ **26**
Can a presigned request be used multiple times? Is that a security risk? 26
Can someone other than the intended user use a presigned request? 26
Can an authorized user use a presigned request to exfiltrate data? 26
Can I deny access from a presigned URL if I suspect it's been shared in an unauthorized way? 27
Resources **29**
Amazon S3 documentation 29

Other references 29
Appendix A: How AWS services use presigned URLs 30
Amazon S3 console 30
Amazon S3 Object Lambda 31
AWS Lambda Cross-Region CopyObject 32
AWS Lambda GetFunction 32
Amazon ECR 33
Amazon Redshift Spectrum 33
Amazon SageMaker AI Studio 33
Appendix B: How controls for presigned URLs affect AWS services 35
Guardrail for s3:signatureAge 35
Guardrail for s3:authType when not using network restrictions 35
Document history 37

Establishing guardrails and monitoring for presigned URLs

Antonio Aga Rossi, Amazon Web Services

Security is a critical concern for all companies and a key pillar in the AWS Well-Architected Framework. As a security engineer, you will want to implement administrative guardrails that are aligned with organizational control requirements. In the AWS Well-Architected Framework, guardrails define the boundaries that limit activity.

This guide provides background information and best practices for using presigned URLs, which are used with Amazon Simple Storage Service (Amazon S3) objects. Presigned URLs allow users or applications that have access to valid credentials to generate requests that are signed in advance and are accepted until a defined expiration time. A common use case for presigned URLs is to extend access to objects or resources by sharing these requests. Shared presigned requests are generated by systems or users that have the rights to perform a specific request, and can then be sent to other systems or users to extend the ability to perform that same request.

In this guide, you will learn:

- The concepts of presigned URLs
- Use cases for presigned URLs
- Recommended and optional guardrails
- Monitoring options
- Examples of how AWS services use presigned URLs

Best practices

The following sections of this guide discuss best practices that a security engineer should consider. The guidelines include:

- Foundational best practices, which are practices that every organization should follow.
- Additional guardrails, which are practices that you should consider, but might decide to implement partially or with exceptions. These are intended to provide additional control and defense in depth, but should be balanced against overall complexity.

- Logging interactions, which might result from devices or services that are part of your or your customer's responsibility in the shared responsibility model. This section includes precautions to limit the information that is accessible through logs.

Intended audience

This guide is intended for architects and security engineers who are responsible for implementing security controls in the AWS Cloud.

Objectives

As a security engineer, you want to be aware of how solution builders are implementing security and the type of access your end users have. This guide covers one type of access, presigned URLs, which are often used with Amazon S3. Presigned URLs provide builders with options for efficiently bridging authentication mechanisms.

In Amazon S3, presigned URLs represent a unique category of requests. Security engineers can monitor and manage these requests to ensure that they are used only where appropriate and necessary. The objective of this guide is to help security engineers provide this type of high-level oversight.

After reading this guide, you should understand what a presigned URL is, when it is typically used, and the motivations for its use.

Prerequisites

If your company has not defined a security policy, control objectives, or standards, as described in the guide Implementing security controls on AWS, we recommend that you complete those governance tasks before proceeding with this guide.

Before you begin, you should also be familiar with recommended and optional best practices for control and monitoring. For more information, see:

- Service control policies (AWS Organizations documentation)
- Resource control policies (AWS Organizations documentation)
- Bucket policies for Amazon S3 (Amazon S3 documentation)
- Logging requests with server access logging (Amazon S3 documentation)

- Logging Amazon S3 API calls using AWS CloudTrail (Amazon S3 documentation)

Overview of presigned URLs

A presigned URL is a type of HTTP request that's recognized by the AWS Identity and Access Management (IAM) service. What differentiates this type of request from all other AWS requests is the X-Amz-Expires query parameter. As with other authenticated requests, presigned URL requests include a signature. For presigned URL requests, this signature is transmitted in `X-Amz-Signature`. The signature uses Signature Version 4 cryptographic operations to encode all other request parameters.

Notes:

- Signature Version 2 is currently in the process of being deprecated, but it is still supported in some AWS Regions. This guide applies to Signature Version 4 signing.
- The receiving service could process unsigned headers, but support for that option is limited and targeted, in line with best practices. Unless otherwise noted, assume that all headers must be signed for a request to be accepted.

The `X-Amz-Expires` parameter allows a signature to be processed as valid with a greater deviation from the encoded date time. Other aspects of signature validity are still evaluated. The signing credentials, if temporary, must not be expired at the time the signature is processed. The signing credentials must be attached to an IAM principal who has sufficient authorization at the time of processing.

Presigned URLs are a subset of presigned requests

A presigned URL is not the only method to sign a request for a future time. Amazon S3 also supports POST requests, which are commonly presigned as well. A presigned POST signature allows uploads that comply with a signed policy and has an expiration date that's embedded in that policy.

Signatures for requests can be future dated, although this is uncommon. As long as the underlying credentials are valid, the signature algorithm doesn't prohibit future dating. However, these requests can't be successfully processed until their valid timing window, which makes future dating impractical for most use cases.

What does a presigned request allow?

A presigned request can only allow actions that are allowed by the credentials that were used to sign the request. If the credentials implicitly or explicitly deny the action that's specified by the presigned request, the presigned request is denied when it's sent. This applies to the following:

- Session policies that are associated with the credentials
- Identity-based policies that are associated with the principal who is associated with the credentials
- Resource policies that affect the session or principal
- Service control policies that affect the session or principal
- Resource control policies that affect the session or principal

Motivations for using presigned requests

As a security engineer, you should be aware of what motivates solution builders to use presigned URLs. Understanding what is necessary and what is optional will help you communicate with solution builders. Motivations might include the following:

- To support a non-IAM authentication mechanism while benefiting from scalability in Amazon S3. A core motivation is to communicate directly with Amazon S3 to benefit from the built-in scalability provided by this service. Without this direct communication, a solution would need to support the load from retransmitting bytes that are sent in **PutObject** and **GetObject** calls. Depending on total load, this requirement adds scaling challenges a solution builder might want to avoid.

  Other means of communicating directly with Amazon S3, such as using temporary credentials in AWS Security Token Service (AWS STS) or Signature Version 4 signatures outside URLs, might not be appropriate for your use case. Amazon S3 identifies users through AWS credentials, whereas presigned requests presume identification through mechanisms other than AWS credentials. Bridging this difference while maintaining the direct communication for data is achievable through presigned requests.
- To benefit from a browser's native understanding of URLs. URLs are understood by browsers, whereas AWS STS credentials and Signature Version 4 signatures are not. This is beneficial when integrating with browser-based solutions. Alternative solutions require more code, will use more memory for large files, and might be treated differently by extensions such as malware and virus scanners.

Comparison with temporary AWS STS credentials

Temporary credentials are similar to presigned requests. They both expire, allow scoping of access, and are commonly used to bridge non-IAM credentials to usage that requires AWS credentials.

You can tightly scope a temporary AWS STS credential to a single S3 object and action, but this can result in scaling challenges because AWS STS APIs have limits. (For more information, see the article How can I resolve API throttling or "Rate exceeded" errors for IAM and AWS STS on the AWS re:Post website.) In addition, each generated credential requires an AWS STS API call, which adds latency and a new dependency that might affect resilience. A temporary AWS STS credential also has a minimum expiration time of 15 minutes, whereas a presigned request can support shorter durations. (60 seconds is practical given the right conditions.)

Comparison with signature-only solutions

The only inherently secret component of a presigned request is its Signature Version 4 signature. If a client knows the other details of a request and is provided with a valid signature that matches those details, it can send a valid request. Without a valid signature, it cannot.

Presigned URLs and signature-only solutions are cryptographically similar. However, signature-only solutions have practical advantages such as the ability to use an HTTP header instead of a query string parameter to transmit the signature (see the Logging interactions and mitigations section). Administrators should also consider that query strings are more commonly treated as metadata, whereas headers are less commonly streated as such.

On the other hand, AWS SDKs provide less support for generating and using signatures directly. Building a signature-only solution requires more custom code. From a practical perspective, using libraries instead of custom code for security is a general best practice, so the code for signature-only solutions requires extra scrutiny.

Signature-only solutions do not use the `X-Amz-Expires` query string and provide no explicit validity period. IAM manages the implicit validity periods of signatures that don't have an explicit expiration time. Those implicit periods aren't published. They don't typically change, but they are managed with security in mind, so you shouldn't take a dependency on the validity periods. There's a tradeoff between having explicit control over the expiration date and having IAM manage the expiration.

As an administrator, you might prefer a signature-only solution. However, in a practical sense, you'll need to support solutions as built.

Identifying presigned requests

Identifying requests that used a presigned URL

Amazon S3 provides two built-in mechanisms for monitoring usage at a request level: Amazon S3 server access logs and AWS CloudTrail data events. Both mechanisms can identify presigned URL usage.

To filter logs for presigned URL usage, you can use the authentication type. For server access logs, examine the Authentication Type field, which is typically named authtype when it's defined in an Amazon Athena table. For CloudTrail, examine AuthenticationMethod in the `additionalEventData` field. In both cases, the field value for requests that use presigned URLs is `QueryString`, whereas `AuthHeader` is the value for most other requests.

`QueryString` usage isn't always associated with presigned URLs. To restrict your search to only presigned URL usage, find requests that contain the query string parameter `X-Amz-Expires`. For server access logs, examine Request-URI and look for requests that have an `X-Amz-Expires` parameter in the query string. For CloudTrail, examine the `requestParameters` element for an `X-Amz-Expires` element.

```
{"Records": [{…, "requestParameters": {…, "X-Amz-Expires": "300"}}, …]}
```

The following Athena query applies this filter:

```
SELECT * FROM {athena-table} WHERE
  authtype = 'QueryString' AND
  request_uri LIKE '%X-Amz-Expires=%';
```

For AWS CloudTrail Lake, the following query applies this filter:

```
SELECT * FROM {data-store-event-id} WHERE
  additionalEventData['AuthenticationMethod'] = 'QueryString' AND
  requestParameters['X-Amz-Expires'] IS NOT NULL
```

Identifying other types of presigned requests

The POST request also has a unique authentication type, `HtmlForm`, in Amazon S3 server access logs and CloudTrail. This authentication type is less common, so you might not find these requests in your environment.

The following Athena query applies the filter for `HtmlForm`:

```
SELECT * FROM {athena-table} WHERE
  authtype = 'HtmlForm';
```

For CloudTrail Lake, the following query applies the filter:

```
SELECT * FROM {data-store-event-id} WHERE
  additionalEventData['AuthenticationMethod'] = 'HtmlForm'
```

Identifying request patterns

You can find presigned requests by using the techniques discussed in the previous section. However, to make that data useful, you'll want to find patterns. The simple `TOP 10` results for your query might provide an insight, but if that's not enough, use the grouping options in the following table.

Grouping option	Server access logs	CloudTrail Lake	Description
User agent	`GROUP BY useragent`	`GROUP BY userAgent`	This grouping option helps you find the source and purpose of requests. The user agent is user supplied and not reliable as an authentication or authorization mechanism. However, it can reveal a lot if you're looking for patterns, because

			most clients use a unique string that is at least partially human readable.
Requester	`GROUP BY requester`	`GROUP BY userIdentity['arn']`	This grouping option helps find IAM principals who signed requests. If your goal is to block these requests or to create an exception for existing requests, these queries provide enough information for that purpose. When you use roles in accordance with IAM best practices, the role has a clearly identified owner, and you can use that information to find out more.

Source IP address	`GROUP BY remoteip`	`GROUP BY sourceIPAddress`	This option groups by the last network translation hop before reaching Amazon S3. • If the traffic passes through a NAT gateway, this will be the NAT Gateway address. • If the traffic passes through an internet gateway, this will be the public IP address that sent the traffic to the internet gateway. • If the traffic originates from outside AWS, this will be the public internet address that's associated with the origin. • If it passes over a gateway virtual private cloud (VPC) endpoint, this will be the IP address of the instance in the VPC.

- If it passes through a public virtual interface (VIF), this will be the on-premises IP of the requester or any intermediary such as a proxy server or firewall that exposes only its IP address.

- If it passes through an interface VPC endpoint, this could be the IP address from an instance in the VPC. It could also be an IP address from another VPC or an on-premises network. As with public VIFs, this could be any intermediary's IP address.

This data is useful if your goal is to impose network controls. You might have to combine this option with data such as `endpoint` (for server access logs) or

			`vpcEndpointId` (for CloudTrail Lake) to clarify the source, because different networks might duplicate private IP addresses.
S3 bucket name	`GROUP BY bucket_name`	`GROUP BY requestParameters['bucketName']`	This grouping option helps find buckets that received requests. This helps you identify the need for exceptions.

Best practices for using presigned requests

The following sections of this guide discuss best practices that a security engineer should consider. The guidelines include:

- Foundational best practices, which are practices that every organization should follow.
- Additional guardrails, which are practices that you should consider, but might decide to implement partially or with exceptions. These are intended to provide additional control and defense in depth, but should be balanced against overall complexity.
- Logging interactions, which might result from devices or services that are part of your or your customer's responsibility in the shared responsibility model. This section includes precautions to limit the information that is accessible through logs.

Foundational best practices

General best practices that are effective controls for other AWS API requests apply to presigned requests as well. This section reviews two of the most relevant practices: least privilege and data perimeters. These practices create a depth of controls that other practices extend.

Apply the principle of least privilege

The first step in limiting the use of presigned requests is limiting access to Amazon S3 in general. A presigned URL cannot provide access to resources that weren't granted to the principal that generated the signature for the presigned URL. Nor can it provide access to a resource in a way that wasn't granted to that principal. As such, applying best practices to grant those principals least privilege is an effective guardrail.

The process of creating a presigned URL is an algorithmic operation that's based on a published standard (Signature Version 4) for signature generation. Therefore, it's not possible to place limits on the generation of presigned URLs. However, to be relevant, a presigned URL must be valid and provide access to resources, so the validity of a presigned URL is also an effective guardrail.

For more information about least privilege, see Grant least privilege access in the AWS Well-Architected Framework, Security pillar.

Implement a data perimeter

An extension of least privilege is to maintain a data perimeter that's consistent your organization's needs. Presigned URLs are compatible with data perimeters. As with other requests, a presigned URL request's validity is evaluated at request time. If the properties of the network, resource, role-session, and principal change, they are evaluated at the time and by using the method by which a request is received.

For example, let's say that a service that's running in an Amazon Elastic Kubernetes Service (Amazon EKS) container signs a request. The request is later sent from a user's personal computer system that's connected to the internet. In this case, the aws:SourceIp condition evaluates the visible public IP address of the request from the the user's personal system, not the IP address of the service in the Amazon EKS container.

Similarly, if the tags of the principal or resource change before the request is sent, the updated, not original, values will apply to the request through the aws:PrincipalTag/tag-key and aws:ResourceTag/tag-key conditions.

Additional guardrails

When presigned requests are used appropriately by solution builders and users, they provide a secure mechanism for giving users access to data. In addition, the ability to generate presigned requests doesn't provide principals with access that they didn't already have.

In that context, are additional controls necessary? The justification for additional controls isn't based on a need to deny access but to provide the ability to monitor, to approve usage and set boundaries, and to reduce risk from user errors. In this way you can help ensure that usage is appropriate and necessary.

The following guardrails assist you in this goal. Before you enable these controls, you might want to determine existing usage by identifying presigned requests. This identification helps you prepare for the guardrail's impact to existing usage or to plan exceptions where needed.

Guardrail for s3:signatureAge

One defining characteristic of presigned requests is that they describe an expiration time. The signature for the request contains a date. This date is transmitted as an `X-Amz-Date` query string parameter for presigned URLs, and as a Date or x-amz-date header for a presigned POST.

Amazon S3 provides a condition key, s3:signatureAge, which you can use to limit the maximum time between the signed date and the effective expiration of the request. This condition can never increase the validity period, but it can reduce it.

In the following policy, the `s3:signatureAge` condition key limits presigned requests to 15 minutes of validity. The following examples all use 15 minutes to limit validity to a similar timeframe as standard signing supports.

The second statement from the policy denies any Signature Version 2 access. This version of the signing protocol is being deprecated, but it is still supported in some AWS Regions. We recommend that you explicitly block it before it is fully deprecated.

You can apply the following policy as an AWS Organizations service control policy (SCP). Users can still use presigned requests and deploy solutions that depend on those requests, as long as the time between signature generation and usage is less than 15 minutes. Depending on the implementation, this limitation might have no impact, it might cause a solution to become unusable, or it might cause occasional failures that can be retried.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyPresignedOver15Minutes",
      "Effect": "Deny",
      "Action": "s3:*",
      "Resource": "*",
      "Condition": {
        "NumericGreaterThan": {
          "s3:signatureAge": "900000"
        }
      }
    },
    {
      "Sid": "DenySignatureVersion2",
      "Effect": "Deny",
      "Action": "s3:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
           "s3:signatureversion": "AWS"
        }
      }
```

```
    }
  ]
}
```

Exceptions

If a solution requires a longer time before expiration and is therefore affected by the preceding policy, we recommend that you provide a method to approve exceptions. To avoid enumerating exceptions in an SCP, use aws:PrincipalTag, as in the following policy, to manage exceptions in a scalable way. Other AWS examples, such as the AWS data perimeter policy examples, use this strategy.

If you implement an exception policy by using `aws:PrincipalTag`, you must control access to setting tags on principals. Tags of this type can come directly from principals and can be controlled by an SCP, as in this example of controlling which tag values can be set. A tag of this type can also come from session tags, which are set by an identity provider (IdP) or AWS STS. Controlling access to `aws:PrincipalTag` is a complex topic. However, an organization that has experience in using attribute-based access control (ABAC) will have the experience and controls to enable the appropriate use of `aws:PrincipalTag` for this use case.

In the following example, the `aws:PrincipalTag` condition creates an exception that allows any principal with the named tag (`long-presigned-allowed`) assigned, and set to `true`. With this exception the restriction on signature age is not applied.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyPresignedOver15Minutes",
      "Effect": "Deny",
      "Action": "s3:*",
      "Resource": "*",
      "Condition": {
        "NumericGreaterThan": {
          "s3:signatureAge": "900000"
        },
        "StringNotEquals": {
          "aws:PrincipalTag/long-presigned-allowed": "true"
        }
      }
    }
```

```
  ]
}
```

Bucket policies

You can apply bucket policies to all or selected buckets by using a policy as in the following example. Unlike an SCP, a bucket policy also targets service principal usage. Appendix A doesn't document any expected service principal usage of presigned requests, but if you wanted to implement a control that for the sake of proving that limit, the following policy would provide that control. Also, unlike an SCP, a bucket policy can apply to principals in your management account.

ABAC-based exceptions work in bucket policies in the same way as an SCP. A goal of a bucket policy might be to apply to principals outside the organization, so ABAC exceptions should be constrained to the principals for which ABAC controls apply.

In the following example, the aws:PrincipalTag condition in the first statement creates an exception for a principal with the named tag (`long-presigned-allowed`) assigned, and set to `true`. With this exception the restriction on signature age is not applied. The second statement applies this restriction to all principals outside the AWS organization that owns the bucket. The scope of this second statement should match ABAC controls to set the named tag for principals.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyPresignedOver15MinWithExceptions",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::{bucket-name}/*",
      "Condition": {
        "NumericGreaterThan": {
          "s3:signatureAge": "900000"
        },
        "StringNotEquals": {
          "aws:PrincipalTag/long-presigned-allowed": "true"
        }
      }
    },
    {
      "Sid": "DenyPresignedOver15MinutesOutsideOrg",
      "Effect": "Deny",
```

```
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::{bucket-name}/*",
      "Condition": {
        "NumericGreaterThan": {
          "s3:signatureAge": "900000"
        },
        "StringNotEquals": {
          "aws:PrincipalOrgID": "${aws:ResourceOrgID}"
        }
      }
    }
  ]
}
```

Resource control policies

You can apply a policy to buckets at scale by using resource control policies (RCPs). Like SCPs and unlike bucket policies, RCPs do not target service principal usage. RCPs affect non-service principals from any account, but they do not affect resources in the management account. For more information, see the AWS Organizations documentation.

As with bucket policies, if you use `aws:PrincipalTags` to create exceptions for principals, keep in mind the scope of ABAC controls on the tagging of principals.

The following RCP restricts presigned URL usage across all S3 buckets in an organization by limiting signature age to 15 minutes:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyPresignedOver15MinWithExceptions",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::*/*",
      "Condition": {
        "NumericGreaterThan": {
          "s3:signatureAge": "900000"
        },
        "StringNotEquals": {
```

```
          "aws:PrincipalTag/long-presigned-allowed": "true",
        }
      }
    },
    {
      "Sid": "DenyPresignedOver15MinutesOutsideOrg",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::*/*",
      "Condition": {
        "NumericGreaterThan": {
          "s3:signatureAge": "900000"
        },
        "StringNotEquals": {
          "aws:PrincipalOrgID": "${aws:ResourceOrgID}"
        }
      }
    }
  ]
}
```

Guardrail for s3:authType

Presigned URLs use query string authentication, and presigned POSTs always use POST authentication. Amazon S3 supports the denial of requests based on authentication type through the s3:authType condition key. `REST-QUERY-STRING` is the s3:authType value for query strings, and `POST` is the s3:authType value for POST.

You can apply the following policy as an SCP. The policy uses `s3:authType` to allow only header-based authentication. It also configures a method to provide exceptions to individual users or roles.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyNonHeaderAuth",
      "Effect": "Deny",
      "Action": "s3:*",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
```

```
          "s3:authType": "REST-HEADER",
          "aws:PrincipalTag/non-header-auth-allowed": "true"
        }
      }
    }
  ]
}
```

Denying requests based on authentication type affects any solution or feature that uses the denied authentication type. For example, denying `REST-QUERY-STRING` prevents users from performing uploads or downloads from the Amazon S3 console. If you want users to use the Amazon S3 console, don't use this guardrail, or make an exception for users. On the other hand, if you don't want users to use the Amazon S3 console, you can deny `REST-QUERY-STRING` for users.

Perhaps you already deny users direct access to Amazon S3 resources. In this case, a guardrail for authentication type is redundant. However, an `s3:authType` deny statement provides defense-in-depth utility because implementations to deny direct access usually span many control statements, some with exceptions.

Roles that are used for workloads won't typically need access to query string or `POST` authentication. Exceptions are roles that support services designed to use presigned requests. You can create specific exceptions for those roles.

You can also apply a bucket policy to all or selected buckets by using a policy such as the following:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyNonHeaderAuth",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::{bucket-name}/*",
      "Condition": {
        "StringNotEquals": {
          "s3:authType": "REST-HEADER",
          "aws:PrincipalTag/non-header-auth-allowed": "true"
        }
      }
    }
  ]
```

```
}
```

This bucket policy has the effect of denying the use of the **CopyObject** and **UploadPartCopy** APIs to make cross-Region copies. Amazon S3 Replication isn't affected because it doesn't rely on these APIs.

If you want to use a bucket policy such as the preceding policy and still support the cross-Region **CopyObject** or **UploadPartCopy** API, add a condition for `aws:ViaAWSService` similar to the following:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyNonHeaderAuth",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::{bucket-name}/*",
      "Condition": {
        "StringNotEquals": {
          "s3:authType": "REST-HEADER",
          "aws:PrincipalTag/non-header-auth-allowed": "true"
        },
        "Bool": {
          "aws:ViaAWSService": "false"
        },
      }
    }
  ]
}
```

Combining presigned guardrails and exceptions to other guardrails

If you don't plan to apply a guardrail generally to your users and roles, you might want to apply it to the exceptions of other common guardrails, so those exceptions don't support presigned requests.

If you have network restrictions but allow exceptions for external partners or special use cases, you should block query string or P0ST authentication when those exceptions are applied, unless they're specifically identified as being required.

Limitations to s3:signatureAge

Administrators will find it useful to understand the implications of `s3:signatureAge` more fully. Every signed request includes `X-Amz-Date`, which should indicate the current time. This value is filled by the client and request signer. AWS rejects requests that it considers to have invalid times. However, a signer could generate signatures in advance with a future time. Amazon S3 rejects requests that specifies a future time if it's sent too far in advance. However, if the request isn't sent until the time that's signed into the signature, the signature could be generated earlier and sent later.

`s3:signatureAge` limits the maximum age of `X-Amz-Date` in a signature only for presigned requests. Requests that are older than the specified age are denied, even if the expiration in `X-Amz-Expires` or a `POST` policy would have declared it valid. `s3:signatureAge` doesn't change the valid period for requests that don't include an explicit expiration. It also doesn't control the value of `X-Amz-Date` that a client uses for a signature.

If the system clock is wrong or if a client intentionally future-dates requests, the signed time might not be the time the signature was generated. This limits how much `s3:signatureAge` can control solutions. A solution that uses the current time when it generates signatures is limited in the expected way: The signatures remain valid for the number of milliseconds specified in `s3:signatureAge`. A solution that doesn't use the current time will have different limits. One restriction is that the credentials that were used to sign the signature must still be valid. As an administrator, you can control the maximum validity of temporary credentials issued. You can allow the credentials to be valid for up to 36 hours or restrict validity to as low as 15 minutes. The expiration of temporary credentials isn't dependent on the value of `X-Amz-Date`.

Permanent credentials don't have this restriction. Using only temporary credentials is a best practice, and you can explicitly revoke any permanent credential, which would also invalidate any signatures based upon that credential.

Although `s3:signatureAge` is measured in milliseconds, it isn't practical to set it to less than 60 seconds, even if you have a well-synchronized clock and low-latency usage. Settings that are lower than 60 seconds carry the risk of rejecting valid requests. If you expect delays between signature generation and request submission, or issues with clock synchronization, you should account for these in your management of `s3:signatureAge`.

Targeting buckets at scale

SCPs and RCPs can use `aws:PrincipalTag` to make exceptions for users. You can't use tags on a bucket to control access through `aws:ResourceTag`—only object tags are used for access control. It isn't generally scalable to add a tag to every object you want to apply this control to.

A solution that fits many use cases is to apply the policy and exception at the account level, either by changing the accounts that the SCP or RCP applies to or by using aws:ResourceAccount, aws:ResourceOrgPaths, or aws:ResourceOrgID. For example, an SCP or RCP might be applied to a set of production accounts.

Another solution is to use a custom AWS Config rule to implement a detective control or responsive control. The goal would be for every bucket to contain a bucket policy with the appropriate guardrail. In addition to testing the content of a bucket policy, the custom AWS Config rule can retrieve the tags from the bucket and exclude the bucket from the rule if the bucket is tagged with a specific value. If that rule fails its compliance check, it could either mark the bucket as non-compliant or invoke a remediation to add the guardrail to the bucket's policy.

> **Note**
>
> You can't restrict the tag content of requests to PutBucketTagging. To maintain control over how a bucket is tagged, you must limit access to `PutBucketTagging` and DeleteBucketTagging.

Logging interactions and mitigations

A presigned URL contains a signature and can be used, during the period before expiration, to perform the specific API operation it was signed for. It should be treated as a temporary access credential. The signature should remain private to only parties that need to know it. In most environments, this is the client that sends the request and the server that receives it. Sending the signature as part of a direct HTTPS session maintains its private nature, because only a participant of the HTTPS session has visibility into the URI that transmits the signature.

For presigned URLs, the signature is transmitted as the `X-Amz-Signature` query string parameter. Query string parameters are a component of a URI. The risk is that clients could log the URI and the signature with it. Clients have access to the entire HTTP request and could log any part of the request, data, and headers (including authentication headers). However, this is by convention less

common. URI logging is more common and is required in cases such as access logging. Clients should use redaction or masking to remove the signature before logging URIs.

In some environments, users allow intermediaries (proxies) to gain visibility into their HTTPS sessions. Enabling proxies requires a high level of privileged access to client systems, because they require configuration and trusted certificates. The installation of proxy configuration and trusted certificates, within the local context of the client intermediary environment, allows a very high level of privilege. For this reason, access to such intermediaries should be tightly controlled.

The purpose of an intermediary is usually to block unwanted egress and to track other egress. As such, it's common for such intermediaries to log requests. Although intermediaries could, like clients, log any content, headers, and data (all of which would be very sensitive), it's more common for them to log URIs, such as those that include the `X-Amz-Signature` query string parameter.

Mitigations

We recommend that URI logging redact the `X-Amz-Signature` query string parameter, redact the entire query string, or treat the information as highly confidential, as with direct access to the intermediary server.

Although these protections are highly recommended, the fact that presigned URLs expire mitigates the risks of log exposure, as long as the exposure is delayed long enough for the signatures to expire.

Amazon S3 also sees the signatures and must handle them appropriately. Amazon S3 server access logs include the request URI but redact the `X-Amz-Signature`, as recommended. The same is true when CloudTrail data events are logged for Amazon S3. You can configure Amazon CloudWatch Logs to mask data by using custom data identifiers.

The following regular expression matches the `X-Amz-Signature` as it appears in a URI:

```
X-Amz-Signature=[a-f0-9]{64}
```

This following regular expression adds grouping patterns to identify the text to replace more specifically:

```
(?:X-Amz-Signature=)([a-f0-9]{64})
```

If you have an access log entry such as the following:

```
X-Amz-Signature=733255ef022bec3f2a8701cd61d4b371f3f28c9f193a1f02279211d48d5193d7
```

The first regular expression translates the access log entry to:

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

The second regular expression, on systems that support non-capturing groups, translates the access log entry to:

```
X-Amz-Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

FAQ

Can a presigned request be used multiple times? Is that a security risk?

Yes, a signature in a presigned request could be used more than once. Whether this is a security risk is a contextual question. Other methods of accessing AWS services allow repetition as well. A user or workload with AWS credentials can send many requests to AWS services, and any of those requests could be duplicates.

If your use case requires once and only once execution, you should implement other mechanisms to enforce single use. Single use is not a feature of presigned requests. As a security engineer, you should review the use cases and implementations, but in many cases, multiple use will fit acceptable use.

Can someone other than the intended user use a presigned request?

A signature in a presigned request can be sent by anyone in possession of it. It will be accepted only if it passes other forms of validation, such as data perimeter controls. If the signature has expired, the signing credentials have expired, or the signing credentials don't have access to the requested resources, the request will be denied.

The same is true for other methods of authenticating with AWS services. Credentials that are shared inappropriately allow inappropriate access. The core best practice is to share credentials and signatures only with the intended audience. If you can't trust your intended audience to keep private data secure and not share it with others, this will undermine any form of authentication.

Can an authorized user use a presigned request to exfiltrate data?

Securing data requires robust action. Enabling access for intended purposes while maintaining a data perimeter requires a comprehensive approach. Least-privilege access, data perimeter controls, and using only temporary access credentials are general best practices that apply to securing data.

The appropriate use of these controls also limits the ability of users to perform actions through the presigned requests they generate.

This is because the access provided by a presigned requests is a subset of the access granted to the credentials that are used to sign the request. In this context, best practices that apply to accessing data generally do apply to presigned requests, but presigned requests create no new access to data.

- The maximum expiration is limited to the expiration of the signing credentials. If the signing credentials are revoked, signatures based upon the credentials are no longer valid.
- If the permissions for the IAM principal that is associated with the signing credentials do not include the execution of the action associated with the presigned request, invoking a presigned request results in an "access denied" response. The response depends on the current state of permissions at the time of invocation, which has no relationship to the time that the presigned request's signature was generated.
- Properties of the principal are evaluated based upon the principal that is associated with the signing credentials.
- Properties of a role session are evaluated based on the role session that is associated with the signing credentials.
- Properties of the network are evaluated based upon how the request was received, as with normal requests.

In this context, the examination of risks associated with presigned requests is constrained to areas where they are signed with credentials that are different from the user's credentials and provide access that was not part of a user's principal. This examination should be applied to the design of the service, the workload, or the solution that generates signatures on a user's behalf, instead of the presigned request capability itself.

Can I deny access from a presigned URL if I suspect it's been shared in an unauthorized way?

Yes. This requires invalidating the credentials the URL was signed with. There are multiple ways to accomplish this:

- Remove permissions from the IAM principal the credentials belong to. If that IAM principal no longer has access to the resource and operation that the URL is signed for, the URL can't execute that operation. This affects all matching use from that IAM principal.
- If the credentials used to sign the URL are temporary AWS STS credentials, you can revoke the session permissions for temporary credentials that were issued before a specific time for the IAM principal. Depending on the use case, there might be other valid sessions that become invalidated before their normal expiration time, but new sessions won't be affected. Revoking session permissions also invalidates any URLs that were signed by using credentials associated with those sessions, but new URLs associated with new sessions won't be affected.
- If the credentials used to sign the URL are permanent credentials, deactivate the access key. This affects all usage tied to those credentials.

Resources

Amazon S3 documentation

- Authenticating Requests (AWS Signature Version 4)
- Authenticating Requests: Using Query Parameters (AWS Signature Version 4)
- Authenticating Requests: Browser-Based Uploads Using POST (AWS Signature Version 4)
- Amazon S3 Signature Version 4 Authentication Specific Policy Keys
- Working with presigned URLs

Other references

- Building a Data Perimeter on AWS (AWS whitepaper)
- SEC03-BP02 Grant least privilege access (AWS Well Architected Framework, Security pillar)
- SEC03-BP05 Define permission guardrails for your organization (AWS Well Architected Framework, Security Pillar)

Appendix A: How AWS services use presigned URLs

This appendix provides information about AWS services and features that use presigned URLs. This information serves two purposes:

- To provide security engineers who implement controls with information about the possible impacts of those controls.
- To create awareness of situations where this risk might be relevant for URL logging interactions.

> **Important**
>
> This appendix doesn't provide a complete list of AWS services or their usage of presigned URLs. It also doesn't cover custom or third-party solutions.

Amazon S3 console

Principal: Console user **Default expiration:** 5 minutes

Disclaimer: This section documents the current behavior of the Amazon S3 console. AWS console behaviors are subject to change without notice.

The Amazon S3 console supports downloading and uploading objects. Downloads use a presigned URL that has an expiration time of 300 seconds (5 minutes). The URL is generated by a request to `https://<bucket-region>.console.aws.amazon.com/s3/batchOpsServlet-proxy`.

That request is initiated when the user clicks a download button, so the URL isn't generated in advance or sent to the client until the explicit request to download occurs.

Uploads are similar, except that the console sends two requests: `OPTIONS` as a pre-flight CORS check, and `PUT`. Both requests use the same signature.

The credentials used for signing are temporary credentials that are associated with the currently logged in user. Details about the method for obtaining those temporary credentials are out of scope for this guide.

Amazon S3 Object Lambda

Principal: Access point caller **Default expiration:** 61 seconds

> **Note**
>
> As of November 7th, 2025, S3 Object Lambda is available only to existing customers that are currently using the service as well as to select AWS Partner Network (APN) partners. For capabilities similar to S3 Object Lambda, learn more here - Amazon S3 Object Lambda availability change.

Amazon S3 Object Lambda uses AWS Lambda functions to automatically process and transform data as it is retrieved from Amazon S3. When S3 Object Lambda invokes a function, the function is provided a presigned URL (`inputS3Url`) that it can use to download the original object from the supporting access point.

These presigned URLs are signed for the supporting Amazon S3 access point, which is provided when you configure S3 Object Lambda. (This is not the same as the Object Lambda access point.) Instead of using a role that's bound to the Lambda function, the URL is signed by using the original caller's identity, and that user's permissions will apply when the URL is used. If there are signed headers in the URL, the Lambda function must include these headers in the call to Amazon S3.

The presigned URL that's returned has an expiration time of 61 seconds (one second more than the maximum duration for an S3 Object Lambda function). The generated URL can be used only with the supporting access point. The caller of the S3 Object Lambda access point needs to have access to this access point. You can limit that access to the context of S3 Object Lambda by using the condition `"aws:CalledVia": ["s3-object-lambda.amazonaws.com"]`. When that condition is attached to a supporting access point or bucket, a user can't access the supporting access point or bucket directly.

The value of this approach is that there's no need to grant the Lambda function access to your S3 bucket or access point. The role that's associated with the Lambda function will need permissions for **WriteGetObjectResponse**, but it doesn't need permissions for **GetObject**.

When S3 Object Lambda generates presigned URLs, it doesn't add network restrictions, so a URL can be used outside the Lambda function. However, any restrictions placed on the caller of S3

Object Lambda still apply. For example, if your Lambda function runs in a VPC and you restrict the caller to using a VPC endpoint, anyone in possession of the presigned URL would need the ability to send it through that VPC endpoint. This restriction also applies to **SourceIp** and **VpcSourceIp**.

> **Note**
>
> To use an S3 Object Lambda function in a VPC, the VPC must have a route to public S3 endpoints to call **WriteGetObjectResponse**. This does not indicate that requirements to use a VPC endpoint would not apply to the requests to retrieve data from the bucket.

AWS Lambda Cross-Region CopyObject

Principal: AWS internal

Default expiration: 3600 seconds

When you use the CopyObject or UploadPartCopy API to copy across AWS Regions, Amazon S3 uses presigned URLs internally. These APIs can be called directly from SDKs or from the AWS CLI commands `aws s3api copy-object` and `aws s3api upload-part`. These APIs aren't used for Amazon S3 Replication, but they are used by the AWS CLI `aws s3 cp` and `aws s3 sync` commands when the source and destination are S3 buckets. They are also supported by `TransferManager` implementations in various AWS SDKs.

AWS Lambda GetFunction

Principal: AWS internal **Default expiration:** 10 minutes

AWS Lambda stores the user version in a S3 bucket that the Lambda team owns, before generating the assets deployed to Lambda containers. When you want to access the code for your function, you call the GetFunction API. This API responds with `Code.Location`, which contains a presigned URL that's valid for 10 minutes (this expiration time is current behavior and not a published contract). If you don't want the code, you can use a combination of GetFunctionConfiguration, GetFunctionConcurrency, and ListTags to retrieve the other data that's returned by `GetFunction`.

The returned URL isn't signed with the credentials of the currently logged in user, but on behalf of the user by Lambda. For this reason, condition keys (such as `aws:SourceIP`) that are applied to the currently logged in user or the user's temporary session credentials don't apply to the

generated URL. This is true whether condition keys are applied to **GetFunction** only, or applied to all AWS API usage for the user or session.

The Lambda console also uses **GetFunction** and the presigned URL it returns. The console uses the temporary credentials associated with the currently logged in user to call **GetFunction**. Details about obtaining those temporary credentials are out of scope for this document.

Amazon ECR

Principal: AWS internal **Default expiration:** 1 hour

Amazon Elastic Container Registry (Amazon ECR) provides the GetDownloadUrlForLayer API, which returns a presigned URL that's valid for one hour and supports the download of a single layer from an Amazon ECR image. However, this operation is used by the Amazon ECR proxy and isn't generally used by users for pulling and pushing images.

Amazon Redshift Spectrum

Principal: Role passed to CREATE EXTERNAL SCHEMA through `IAM_ROLE` **Default expiration:** 1 hour

Amazon Redshift Spectrum uses presigned URLs internally and prohibits restrictions on the combination of the bucket and Amazon Redshift role that would limit presigned URLs. You can use a `s3:signatureAge` value of 16 minutes, but very low values are unreliable. The minimum value you can use depends on the timing and size of your query. Although a value that's lower than 16 minutes works for many scenarios, it requires testing. The role can and should be restricted to be used only by Redshift Spectrum, which does not disclose the URLs it generates, thus mitigating the typical justification for lower expiration values.

Amazon SageMaker AI Studio

Amazon SageMaker AI Studio supports two API actions: CreatePresignedDomainUrl and CreatePresignedNotebookInstanceUrl. However, these APIs aren't related to the Signature Version 4 presigned URL feature. These APIs create a URL that uses an `authToken` parameter, but they don't support any of the standard Signature Version 4 query parameters.

`authToken` is a different mechanism but has similarities to presigned URLs. It's sent as a query string parameter and supports an expiration time of 5 minutes.

SageMaker AI supports network restrictions. If you place a restriction on the `sagemaker:CreatePresignedDomainUrl` action, that action applies both to calling CreatePresignedDomainUrl and to the use of the generated URL. If a URL is generated from a valid network and then sent by a non-valid network, the API call to generate the URL succeeds, but the request that sends the URL fails. The same is true of CreatePresignedNotebookInstanceUrl and the `sagemaker:CreatePresignedNotebookInstanceUrl` action.

For more information, see the SageMaker AI documentation.

Appendix B: How controls for presigned URLs affect AWS services

This appendix describes interactions between the AWS services that use presigned URLs, as described in Appendix A, and the controls described earlier in this guide.

Guardrail for s3:signatureAge

The Amazon S3 console isn't disrupted by the maximum expiration of 5 minutes that's set by the `s3:signatureAge` condition key. The Amazon S3 console generates presigned URLs when you choose the **Download** button and applies its own 5-minute expiration time. A maximum duration that's shorter than 2 minutes might create random failures based on clock synchronization and latencies.

Amazon S3 Object Lambda uses an expiration time of 61 seconds, so setting conditions on an `s3:signatureAge` value of 61 seconds or more won't cause any disruption. Shorter durations might be less reliable and might cause intermittent failures.

Amazon S3 cross-Region **CopyObject** isn't disrupted by a maximum expiration of 5 minutes. However, shorter durations might create random failures based on clock synchronization and latencies.

In AWS Lambda, **GetFunction** provides a URL to objects outside the customer account, so customer policies don't affect the generated URLs.

Amazon Redshift Spectrum has been tested with an `s3:signatureAge` condition of 16 minutes. However, shorter durations might cause disruption.

Guardrail for s3:authType when not using network restrictions

The Amazon S3 console is usually affected by the `s3:authType` guardrail. The console routes to Amazon S3 based on the local network configuration. If the local network routes to Amazon S3 in a way that the network restriction allows, the Amazon S3 console would still work. However, if it's routed through a proxy or the public internet in a way that isn't allowed, usage would be blocked. However, blocking usage is probably the intent of this policy.

Amazon S3 Object Lambda is affected if the Lambda function isn't connected to an appropriate VPC. In this configuration, the VPC must have a NAT gateway, not to access the S3 bucket, but to call **WriteGetObjectResponse**.

Amazon S3 cross-Region **CopyObject** is disrupted if this guardrail is applied to a bucket policy without the recommended exception for when `aws:viaAWSService` is **true**.

Amazon Redshift Spectrum is affected by the `s3:authType` guardrail unless enhanced VPC routing is used. Currently, Redshift Spectrum supports enhanced VPC routing only with serverless clusters, not with provisioned clusters.

Document history

The following table describes significant changes to this guide.

Change	Description	Date
Updates and clarifications	In the Additional guardrails section, added information about RCPs and clarified exceptions.	August 8, 2025
Initial publication	—	July 23, 2024