AWS Whitepaper

Building A Data Perimeter on AWS

Building A Data Perimeter on AWS: AWS Whitepaper

Table of Contents

Building a Data Perimeter on AWS

Publication date: **June 13, 2023** (*Document history*)

Many organizations want to implement perimeter controls to help protect against unintended access and configuration errors through always-on permissions guardrails. This paper outlines the best practices and available services for creating a perimeter around your identities, resources, and networks in AWS.

Are you Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the AWS Architecture Center.

Introduction

In traditional, on-premises data center environments, a trusted network and strong authentication are the foundation of security. They establish a high-level perimeter to help prevent untrusted entities from coming in and data from going out. This perimeter provides a clear boundary of trust and ownership. When customers think about creating an AWS perimeter as part of their responsibility for security “in the cloud” in the AWS Shared Responsibility Model, they want to achieve the same outcomes. They want to draw a circle around their AWS resources, like Amazon Simple Storage Service (Amazon S3) buckets and Amazon Simple Queue Service (Amazon SQS) queues, that clearly separates “my AWS” from other customers.

The circle that defines an AWS perimeter is typically represented as an AWS organization managed by AWS Organizations. AWS Organizations is an account management service that lets you consolidate multiple AWS accounts into an organization that you create and centrally manage.

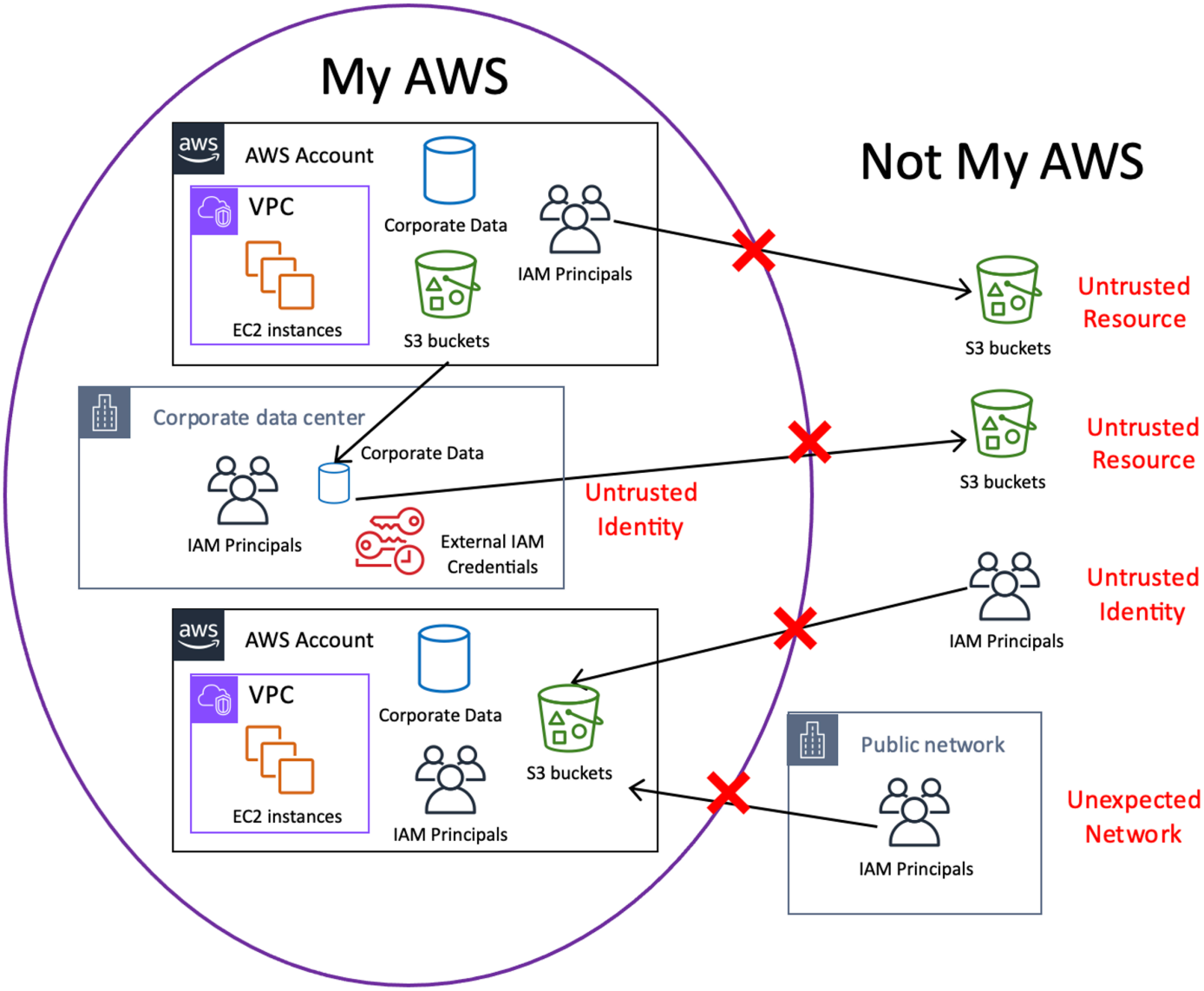
Each AWS account you own is a logical container for AWS identities, resources, and networks. The AWS organization groups of all of those items into a single entity. Along with on-premises

networks and systems that access AWS resources, it is what most customers think of as the perimeter of “my AWS”.

The perimeter defines the things you intend or expect to happen. It refers to the access patterns among your identities, resources, and networks that should be allowed. Using those three elements, we make the following assertion to define our perimeter’s goal: access can only be allowed if the identity is trusted, the resource is trusted, and the network is expected.

If any of these conditions are false, then the access inside the perimeter is unintended and should be denied. The perimeter is composed of controls implemented on your identities, resources, and networks to ensure the necessary conditions are true.

This paper discusses the perimeter objectives and how the applied controls prevent unintended access patterns, particularly to data. It is designed to help customers understand how to create a complete AWS data perimeter as part of their responsibility in the AWS Shared Responsibility Model.



A high-level depiction of defining a perimeter around your AWS resources to prevent interaction with untrusted AWS Identity and Access Management (IAM) principals, untrusted resources, and unexpected networks

Perimeter overview

This section provides an overview of a data perimeter's objectives and the AWS services used to implement it.

Topics

Perimeter objectives

The goal of an AWS perimeter is to ensure access is allowed only if an authorization involves:

- **Only trusted identities** - The AWS Identity and Access Management (IAM) principals in your AWS organization or AWS acting on your behalf.
- **Only trusted resources** - The resources in your AWS organization or resources AWS operates on your behalf.
- **Only expected networks** - Your virtual private cloud (VPC) and on-premises networks or the networks AWS uses on your behalf.

These are the necessary (but not sufficient) authorization conditions for access inside an AWS perimeter to be allowed.

```
Access in the Perimeter#(Trusted Identity)#(Trusted Resource)#(Expected
 Network)
```

Ensuring the truth of these three conditions ultimately defines the objectives of the perimeter. Each authorization condition has two objectives.

Authorization condition	Perimeter objective
Only trusted identities	Ensure that my resources can be accessed by only **trusted identities**.
	Ensure that only **trusted identities** are allowed from my networks.

Authorization condition	Perimeter objective
Only trusted resources	Ensure that my identities can access only **trusted resources**.
	Ensure that only **trusted resources** can be accessed from my networks.
Only expected networks	Ensure that my identities can access resources only from **expected networks**.
	Ensure my resources can only be accessed from **expected networks**.

AWS services

You can establish a data perimeter by using permissions guardrails that restrict access outside of an organization boundary. This is achieved using three primary AWS capabilities, AWS Organizations service control policies (SCPs), resource control policies (RCPs), and VPC endpoint policies. The following is a summary of these different types of policies and some considerations.

Service control policies

SCPs are a type of organization policy that you can use to manage permissions in your organization and control the maximum available permissions for your principals. SCPs offer central control over the maximum available permissions for all principals in your organization. SCPs configured as deny lists can limit the scope of access to resources or source networks in your organization or within specific accounts. It's important to note that SCPs do not apply to service-linked roles (SLR) or AWS service principals.

Although you can define similar controls with IAM identity-based policies, the solutions in this document will favor using SCPs. They are a more scalable way to implement permissions guardrails for all your IAM principals than creating and managing individual policies for every IAM principal you own. However, access to resources in AWS requires explicit permissions in identity-based policies, so they are still a necessary component of providing access, but not as part of establishing data perimeter guardrails.

SCPs are primarily going to use the `aws:ResourceOrgID` condition for achieving only trusted resources and `aws:SourceIp`, `aws:SourceVpc`, and `aws:ViaAWSService` as the conditions for achieving only expected networks as well as for managing exceptions required in the policies. Refer to **service_control_policies** in the data perimeter policy examples repo for more details.

Resource control policies

RCPs are a type of organization policy that you can use to control the maximum available permissions for resources in your organization. When you use RCPs with supported services, you can limit the scope of access to identities or source networks in your organization. Similarly to SCPs, RCPs do not apply to SLRs. RCPs, however, can be used to limit access by AWS service principals.

Although you can define similar controls with resource-based policies, the solutions in this document favor using RCPs. They are a more scalable way to implement permissions guardrails on resources in your organization than creating and managing individual policies for every resource you own. RCPs help central admin teams to enforce security invariants and provide developers more freedom to manage resource-based policies. Developers can use resource-based policies to grant permissions as required by their applications. They are commonly used to provide cross-account access, and can be used to authorize external AWS credentials or anonymous access. Resource-based policies can also be used to apply data perimeter guardrails on resources that are not yet supported by RCPs. For a list of services that support resource-based policies, refer to AWS services that work with IAM. See the *AWS Organizations User Guide* for the List of AWS services that support RCPs.

Resource control policies are primarily going to use:

- `aws:PrincipalOrgID`, `aws:PrincipalIsAWSService`, and `aws:SourceOrgID` conditions to achieve only trusted identities
- `aws:SourceIp`, `aws:SourceVpc`, `aws:ViaAWSService`, and `aws:PrincipalIsAWSService` as conditions for achieving only expected networks, as well as for managing exceptions required in the policies

Refer to **resource_control_policies** in the data perimeter policy examples repo for more details.

VPC endpoint policies

These policies are a special type of resource-based policy that you attach to an endpoint that controls access to resources when they are accessed through that VPC endpoint. An endpoint policy does not override or replace IAM user policies or service-specific policies (such as Amazon S3 bucket policies). It is a separate policy for controlling access from the endpoint to the specified service. In VPC networks, traffic is routed to VPC endpoints automatically if you are using an AWS-provided domain name system (DNS).

For on-premises networks, you can also route AWS traffic through VPC endpoints if they are connected to AWS with AWS Direct Connect or a VPN. For services that have AWS PrivateLink interface endpoints, you can route traffic to those endpoints directly from an on-premises network. For a list of services that support VPC endpoints and VPC endpoint policies, refer to AWS services that integrate with AWS PrivateLink.

VPC endpoint policies are primarily going to use the `aws:PrincipalOrgID` and `aws:PrincipalIsAWSService` conditions to achieve only trusted identities and the `aws:ResourceOrgID` condition to achieve only trusted resources. Refer to **vpc_endpoint_policies** in the data perimeter policy examples repo for more details.

Note

For a VPC endpoint policy, it's important to explicitly specify the AWS organization ID in the condition. **DO NOT** use a condition like this:

```
"Condition": {
  "StringNotEquals": {
    "aws:ResourceOrgID": "${aws:PrincipalOrgID}"
  }
}
```

If an unintended principal in a different AWS organization tries to access a resource in their own organization over the VPC endpoint, this condition won't deny the action because the values for both keys in the request context, `aws:ResourceOrgID` and `aws:PrincipalOrgID`, will be the unintended principal's organization ID. This is different from an SCP because SCPs are only applied to principals that are part of your organization,

but a VPC endpoint policy can apply to any principal. Instead, use a condition like the following in a VPC endpoint policy:

```
"Condition": {
  "StringNotEquals": {
    "aws:ResourceOrgID": "<my-org-id>",
    "aws:PrincipalOrgID": "<my-org-id>"
  }
}
```

Summary

The following table outlines how the different types of policies are used to achieve the perimeter objectives to support the necessary authorization conditions.

Authorization condition	Perimeter objective	Policy type used	Primary IAM condition(s) used
Only trusted identities	Ensure that my resources can be accessed by only **trusted identities**.	RCPs	`aws:PrincipalOrgID` , `aws:PrincipalIsAWSService` , `aws:SourceOrgID`
	Ensure that only **trusted identities** are allowed from my networks.	VPC endpoint policies	`aws:PrincipalOrgID` , `aws:PrincipalIsAWSService`
Only trusted resources	Ensure that my identities can access only **trusted resources**.	SCPs	`aws:ResourceOrgID`

Authorization condition	Perimeter objective	Policy type used	Primary IAM condition(s) used
	Ensure that only **trusted resources** can be accessed from my networks.	VPC endpoint policies	`aws:ResourceOrgID`
Only expected networks	Ensure that my identities can access resources only from **expected networks**.	SCPs	`aws:SourceIp` , `aws:SourceVpc` , `aws:ViaAWSService`
	Ensure my resources can only be accessed from **expected networks**.	RCPs	`aws:SourceIp` , `aws:SourceVpc` , `aws:ViaAWSService` , `aws:PrincipalIsAWSService`

Perimeter implementation

This section describes the complete perimeter solution by evaluating each perimeter authorization condition and how the different policy types are used to achieve it. Each section will describe the overall solution for that objective, provide links to detailed policy examples, explain how exceptions can be implemented, and demonstrate how the controls prevent the unintended access pattern.

Only trusted identities

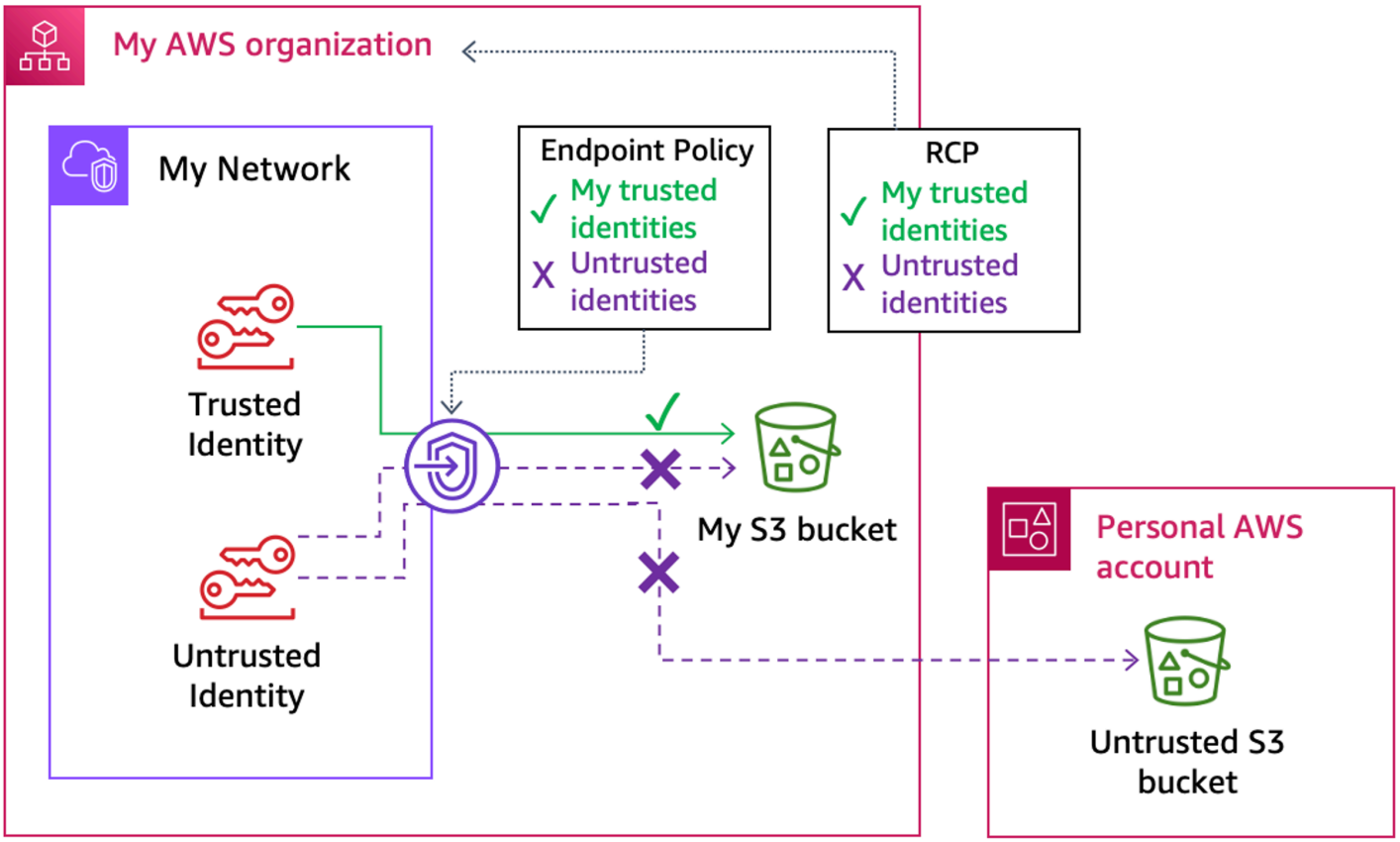
The objectives for this condition ensure that only *trusted identities* can access *my resources* and only *trusted identities* are allowed from *my networks*. You'll use RCPs and VPC endpoint policies to constrain which principals are allowed access.

The primary way to ensure IAM principals belong to *my AWS* is by specifying the `aws:PrincipalOrgID` IAM policy condition in those policies. This requires that the principal being considered during the authorization of access to a resource you own or originating from a network you own (regardless of the resource owner) belongs to your AWS organization.

You can implement a more granular restriction with the `aws:PrincipalAccount` or `aws:PrincipalOrgPaths` IAM policy conditions as well.

When implementing data perimeter guardrails, it's still important to ensure that your resource policies only allow the intended access. You can use IAM Access Analyzer for supported resources to identify resource-based policies that are too permissive. You can also use IAM Access Analyzer to validate if your RCPs help prevent unintended access, regardless of the resource-based policies applied.

The following diagram demonstrates how these controls prevent untrusted identities from accessing your resources or using your networks.



Preventing untrusted identities in an RCP and a VPC endpoint policy

In certain cases, AWS services might use an AWS service principal instead of an IAM role to interact with your resources. A service principal is not part of your AWS organization like IAM roles are. These are intended access patterns, but they need to be explicitly allowed in your RCPs and, in some cases, VPC endpoint policies. For example, AWS CloudTrail uses the service principal `cloudtrail.amazonaws.com` to deliver logs to your Amazon S3 bucket, which requires an exception in your RCP statements that enforce `aws:PrincipalOrgID` or related condition keys. Similarly, when you use a presigned URL for wait condition signaling from a VPC with AWS CloudFormation, you need to allow the `cloudformation.amazonaws.com` service principal. In Allow statements, the service principal can be listed explicitly. However, you might also have Deny statements that the service principal needs to be exempt from. You can't use a `NotPrincipal` statement with an AWS service principal; instead, use the `aws:PrincipalIsAWSService` condition to exempt service principals from Deny statements.

Use the `aws:SourceOrgID` condition key in your RCPs to ensure that AWS services access your resources only on your behalf. To continue with the CloudTrail example, you can require that the CloudTrail trail that delivers logs to your S3 bucket belongs to an account within your organization.

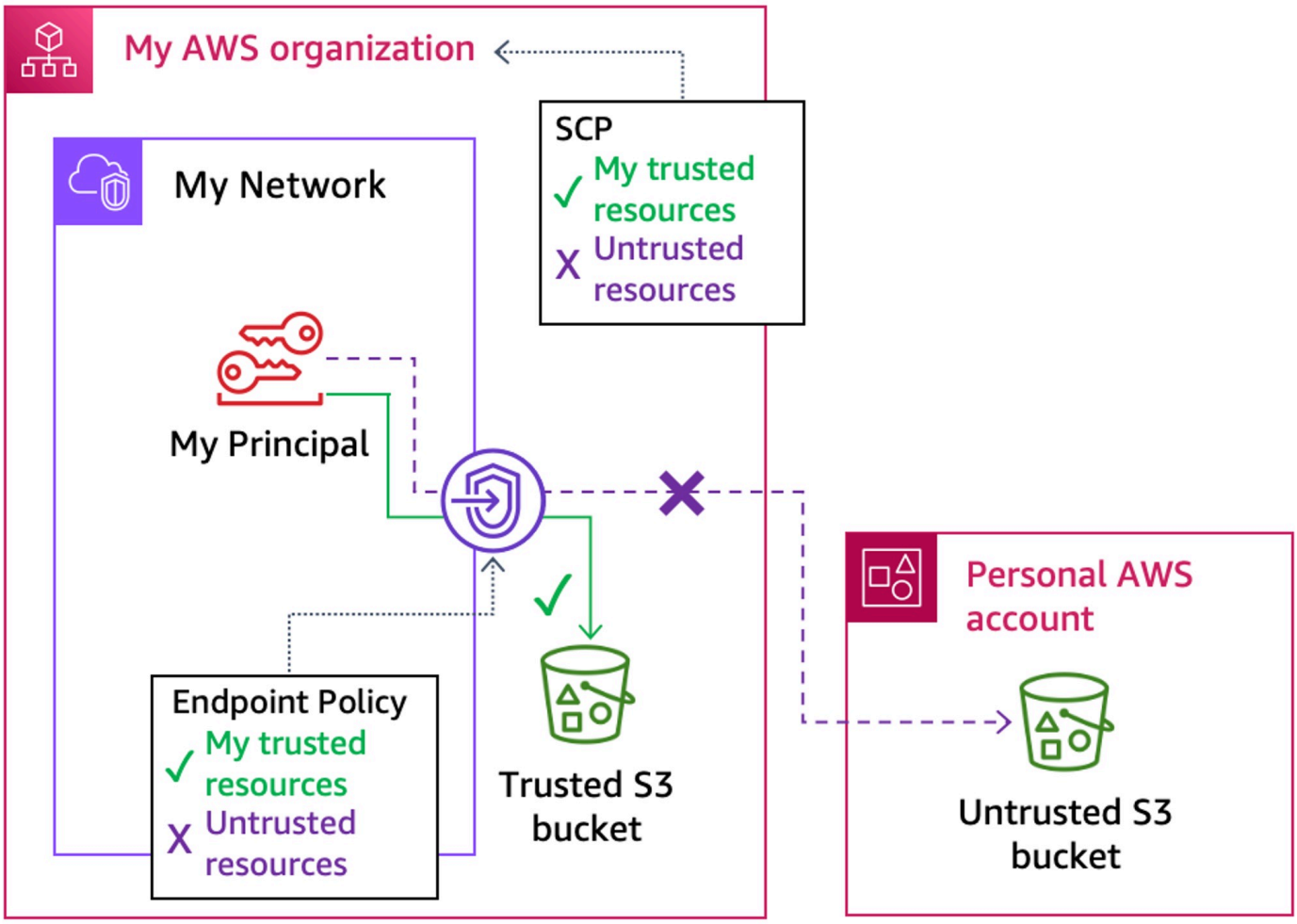
Alternatively, you can use `aws:SourceOrgPaths` or `aws:SourceAccount` condition keys to enforce more granular controls when mitigating cross-service confused deputy risk.

Refer to the AWS data perimeter policy examples repo for templates and instructions on how to create the required RCPs and VPC endpoint policies to achieve this objective as well as create any required exceptions. Specifically, look at the example resource_control_policies and the example vpc_endpoint_policies.

Only trusted resources

The objectives for this condition ensure that *my principals* can only access *trusted resources* and that access from *my networks* only targets *trusted resources* (regardless of the principal involved). You'll use SCPs and VPC endpoint policies to constrain which resources are allowed to be accessed.

The primary way to ensure targeted resources belong to *my AWS* is by specifying the `aws:ResourceOrgID` IAM policy condition in SCPs and VPC endpoint policies. This ensures that the resource being considered during authorization, either being accessed directly or through a VPC endpoint, belongs to your AWS organization. The following diagram demonstrates how these policies prevent access to an untrusted resource.



Preventing access to untrusted resources with SCPs and VPC endpoint policies

Using VPC endpoint policies in this way can be considered a defense in depth approach. This is because implementing the controls for the ??? objectives apply a policy on each endpoint that ensures only my principals can access resources from my networks. Then, the SCP used for this objective always applies to these principals to constrain what resources they can access. This indirectly accomplishes the same outcome as applying an `aws:ResourceOrgID` condition to your VPC endpoint policies.

In some cases, you might need to directly access resources outside of *my AWS*. These could be Amazon S3 buckets that AWS provides for things like Amazon Linux packages, CloudWatch agent installation, or public data repositories. They could also be resources like public SSM parameters. When you use `cfn-hup`, it supports the `on.command` hook, which allows you to use Amazon SQS messages to invoke the `cfn-hup` actions. This is used by AWS Elastic Beanstalk environments where the `cfn-hup` daemon retrieves an AWS CloudFormation specific credential to query an

Amazon SQS queue owned by AWS. Remember that SCPs don't apply to SLRs or AWS service principals, so when you need to create exceptions to allow them to access resources from your VPCs, you'll need to use VPC endpoint policies. For example, in order to use AWS CloudFormation wait condition signaling, which uses an Amazon S3 presigned URL to a bucket owned by AWS, you have to create an exception to allow this access in a VPC endpoint policy.

You will need to use SCPs and VPC endpoint policies together to create the necessary exceptions. In VPC endpoint policies, you can list the trusted resources explicitly in the resource element of an Allow statement. However, in an SCP, you can only list specific resources in a Deny statement. In this case, we don't want to deny the resources; we want to allow them. Instead, you can amend the trusted resource guardrail SCP that requires `aws:ResourceOrgID`. This statement can use a `NotAction` element to exempt specific actions from meeting the `aws:ResourceOrgID` condition. This combined approach allows you to exempt specific actions in your SCPs from the `aws:ResourceOrgID` guardrail and ensures that only specific resources can be accessed with those permissions by explicitly allowing them in a VPC endpoint policy.

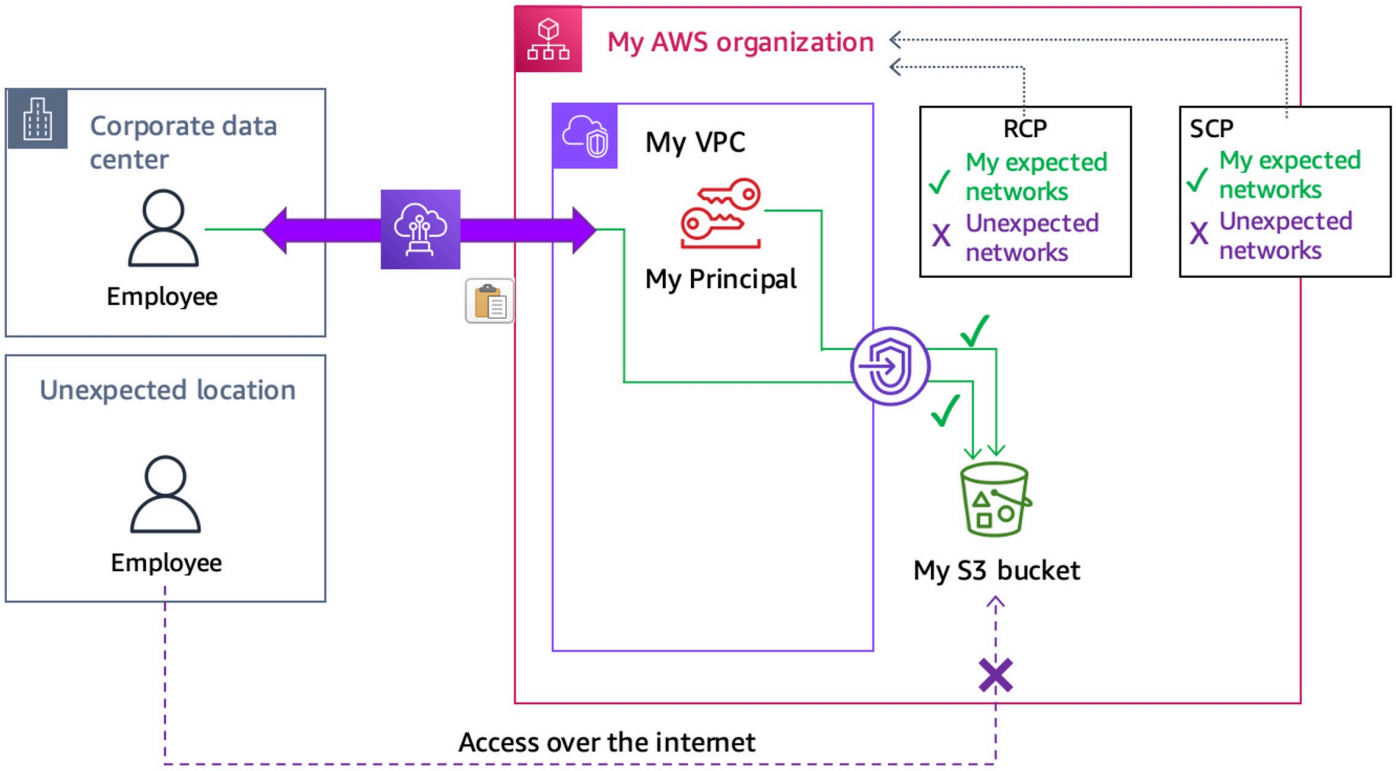
In a few other scenarios, you can use services that use your credentials on your behalf to interact with resources that are not part of your organization. For example, AWS Service Catalog and AWS Data Exchange will write data to an Amazon S3 bucket that you don't own. Amazon Athena can also write data to a bucket you don't own. In the former cases, AWS owns the buckets those services store the data in, but in the latter case with Athena, the bucket could be owned by anyone. For that reason, you might not want to create a broad exception to accessing external resources with the `aws:ViaAWSService` condition. Not all of the resources that can be accessed through an AWS service should be considered a trusted resource. Instead, you can use the `aws:CalledVia` condition to explicitly allow services to behave this way.

Refer to the AWS data perimeter policy examples repo for templates and instructions on how to create the required SCPs and VPC endpoint policies to achieve this objective as well as manage any required exceptions. Specifically look at the resource_perimeter_policy and the vpc_endpoint_policies.

Only expected networks

This final condition's objectives ensures that only *expected networks* can be the source of requests from *my principals* or to *my resources*. You'll use SCPs and RCPs to constrain which networks are allowed for access.

Within an SCP, you can define the expected networks by using an IP address with the `aws:SourceIp` condition key or a VPC identifier with the `aws:SourceVpc` condition. The following diagram shows how these policies prevent access from unexpected network locations.



Preventing access from unexpected networks with SCPs and RCPs

Applying a similar constraint with an RCP can also be considered a defense in depth approach. This is because applying the ??? objectives using an RCP helps ensure that only my principals can access those resources. Then, the network boundary SCP always applies to these principals to constrain the networks they can access resources from. This indirectly accomplishes the same outcome as applying an `aws:SourceIp` condition or `aws:SourceVpc` condition in your RCPs.

There are several scenarios where AWS will act on your behalf with your IAM credentials from networks that AWS owns that will require exceptions to these policies. For example, you can use AWS CloudFormation to define a template of resources for which AWS orchestrates the creation, update, and deletion. The initial request to create a AWS CloudFormation stack will originate from an expected network, but the subsequent requests for each resource in the template are made by the AWS CloudFormation service in an AWS network using your credentials with forward access sessions (FAS) or a service role for AWS CloudFormation that you've specified. This situation also occurs when you use Athena to run queries on CloudTrail logs. The `aws:ViaAWSService` IAM

policy condition provides a way to implement an exception for scenarios where FAS is used in requests made by AWS on your behalf.

When AWS uses a service principal to interact with your resources, the source of that interaction is typically a network owned by AWS. In order to allow this access, you need to create an exception to the expected network control in your RCPs only since SCPs don't apply to service principals. A typical example of this pattern is CloudTrail log delivery to your Amazon S3 bucket.

The last consideration in implementing network perimeter controls is AWS services that operate in compute environments that are not part of your network. For example, Lambda functions and SageMaker AI Studio Notebooks both provide an option to run in AWS-owned networks.

Some of these services provide a configuration option for running the service in your VPC as well. If you want to use the same VPC network boundary for these services, you should monitor and, where possible, enforce it using the VPC configuration.

For example, customers can enforce AWS Lambda function deployments and updates to use Amazon Virtual Private Cloud (Amazon VPC) settings with IAM condition keys, use AWS Config Rules to audit this configuration, and then implement remediation with AWS Config Remediation Actions and AWS Systems Manager Automation documents.

It is important to note that not all AWS services are hosted as an AWS-owned endpoint authorized with IAM; for example, Amazon Relational Database Service (Amazon RDS) databases. Instead, these services expose their *data plane* inside a customer VPC.

The data plane is the part of the service that provides the day-to-day functionality of that thing. For MySQL RDS, it would be the IP address of the RDS instance on port 3306. Network controls such as firewalls or security groups should be used as part of your network boundary to prevent access to AWS services that are hosted in customer VPCs, but are not authorized with IAM credentials. Additionally, customers should leverage alternative authentication and authorization systems to access those services, such as AWS Secrets Manager for Amazon RDS access, when possible.

Refer to the data perimeter policy examples repo for templates and instructions on how to create the required SCPs and RCPs to achieve this objective as well as create any required exceptions. Specifically, look at the network perimeter example in service_control_policies and resource_control_policies for details on how to write policies for this objective as well as manage exceptions. Also, look at the **data_perimeter_governance** examples in service_control_policies for preventing non-VPC deployments of Lambda and SageMaker AI.

Mobile devices

In on-premises networks, there are some resources that are physically static, such as servers. Other resources such as laptops, however, are inherently mobile and can connect to networks outside of your control.

For example, a laptop could be connected to a corporate network when accessing data, which is temporarily stored locally, but then joins a public Wi-Fi network and sends the data to a personal Amazon S3 bucket. This network access pattern could allow access to untrusted resources by bypassing corporate network controls and is a use case that you will need to consider with care.

Customers have generally tried to solve this problem with preventative controls such as always-on VPNs to keep devices connected to a corporate network. They also use detective controls (including agents) to monitor traffic and identify when preventative controls are disabled.

However, these controls aren't fool-proof. There is still some risk that the device could join non-corporate networks. Virtual Desktop Infrastructure (VDI) is typically implemented when the risk of being able to operate a device outside of a controlled network is unacceptable and requires forcing access to AWS resources from non-mobile assets.

Amazon WorkSpaces offers a VDI solution that can be used to require users, developers, and data scientists to use a static asset to interact with AWS resources that is subject to the same network controls as other resources in AWS VPCs. VDI solutions can also be operated by customers natively using Amazon Elastic Compute Cloud (Amazon EC2) instances in a VPC.

Additional considerations

There are a few additional considerations for specific scenarios when building a data perimeter on AWS.

Amazon S3 resource considerations

Amazon S3 is widely used to store and present publicly available website content and public data sets. Access to this content is typically performed anonymously, meaning that the HTTP requests do not have an authorization header or query string parameter generated from AWS credentials.

You might need this anonymous access for users to browse internet websites from VPC networks or on-premises networks that are routed through VPC endpoints. It is also used for workloads that might need to access public data (such as package repositories hosted on Amazon S3 or agent

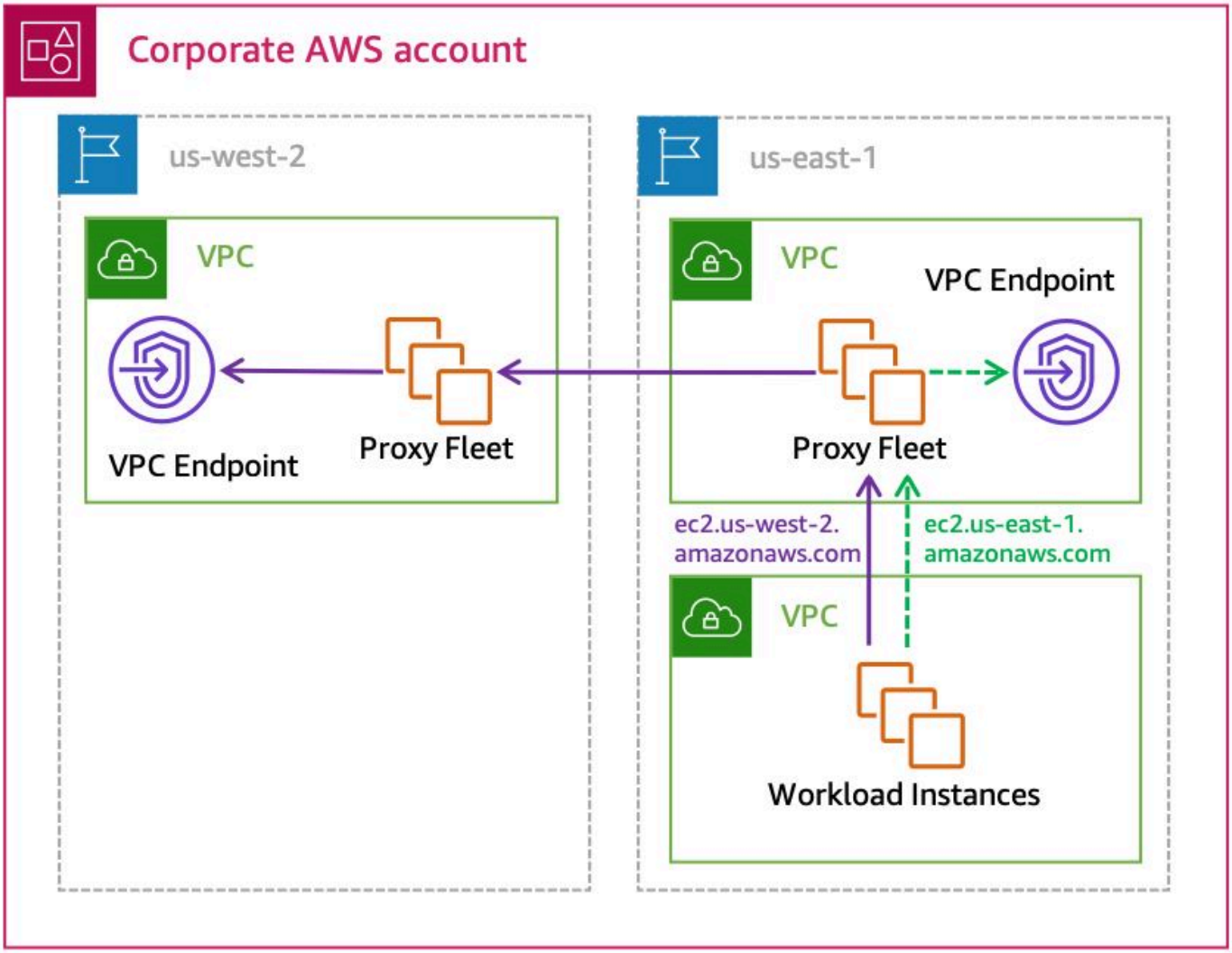
downloads). In order to allow this type of access, you can allow anonymous `GetObject` API calls in your VPC endpoint policies. This is true whether the Amazon S3 content is being accessed using the virtual or path style endpoints or is being accessed via an Amazon S3 website endpoint.

Access to all other Amazon S3 APIs should be authenticated. Refer to the S3 endpoint policy example, which includes details of how to allow anonymous `GetObject` API calls while enforcing authentication and permissions guardrails for trusted resources for the remainder of Amazon S3 actions. The example lists specific AWS resources, but you might also need unauthenticated access to resources where you do not know the bucket or object names (for example, downloading static content on a website in your browser). For those scenarios, you might want to use a dedicated Amazon S3 VPC endpoint that allows unauthenticated `GetObject` API calls for all Amazon S3 resources that is just used for networks where you expect this scenario to be required.

Cross-Region requests

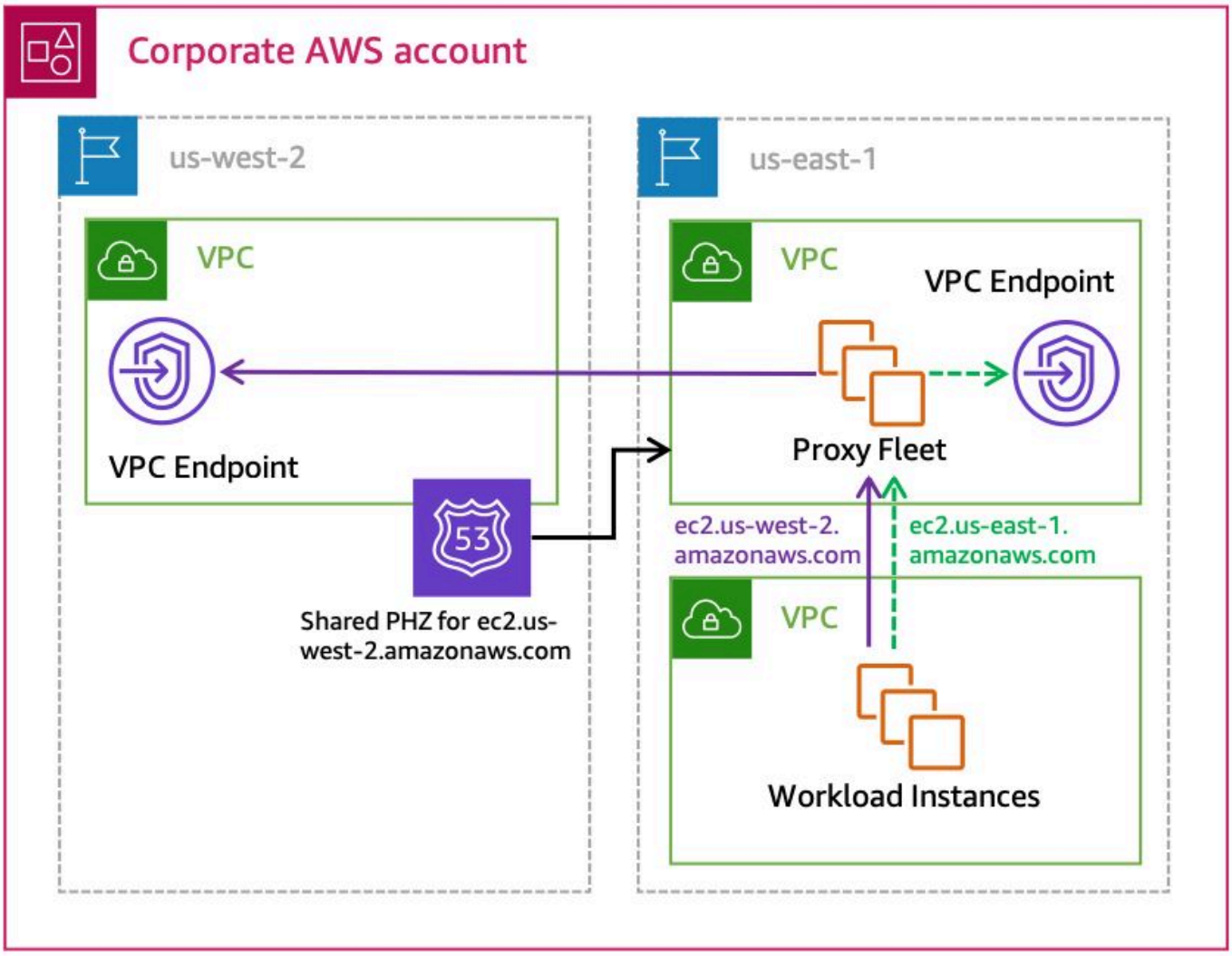
VPC endpoints only support routing AWS API calls to service endpoints that are in the same Region as the VPC endpoint itself. For example, an Amazon S3 VPC endpoint in a VPC in us-east-1 only supports routing traffic for requests made to Amazon S3 buckets in us-east-1. A call to `PutObject` for a bucket in us-west-2 would not traverse the VPC endpoint and would not have the endpoint policy applied to the request. To ensure the intended security controls are applied consistently, you can handle cross-Region requests in three ways:

- Prevent cross-Region API calls using a proxy. This does not require inspecting Transport Layer Security (TLS) and can be done by looking at the hostname in the `CONNECT` request. You can also use Server Name Indication (SNI) because the hostname presented in the `ClientHello` includes the AWS Region in the domain name of the URL (with the exception of global services).
- Use a centralized VPC endpoint approach for each Region and share Route 53 private hosted zones (PHZ) for the endpoints. This allows instances to resolve the out-of-Region endpoint domain name locally and send their request directly to the VPC endpoint. This pattern is described in more detail in Centralized access to VPC private endpoints.
- If you use HTTP/S proxies in your environment, you can use them to forward out-of-Region requests. There are two variations for this option:
  - **Use proxy-chaining** - The proxy in the local Region forwards traffic to a peer proxy running in a VPC in the destination Region. The out-of-Region proxy delivers the traffic to the appropriate VPC endpoint in its Region. See ??? for an example proxy configuration that implements this proxy-chaining solution. The following diagram demonstrates a high-level reference architecture.



Using proxy-chaining to send out-of-Region requests through VPC endpoints

- **Implement a centralized VPC endpoint approach using shared Route 53 private hosted zones** - The local proxy uses AWS-provided VPC DNS and sends the request directly to the out-of-Region VPC endpoint. This eliminates the need to configure proxy-chaining, but does require the creation of a PHZ for each endpoint that will need to be shared with every VPC hosting a proxy. Refer to Cross Region endpoint access for more details.



Forwarding out-of-Region requests using a shared Route 53 PHZ

Preventing access to temporary credentials

Except for the cases of credential theft or leakage, the only other way for an unintended entity to gain access to temporary credentials derived from IAM roles that are part of *my AWS* is through misconfigured IAM role trust policies.

IAM role trust policies define the principals that you trust to assume an IAM role. A role trust policy is a required resource-based policy that is attached to a role in IAM. The principals that you can specify in the trust policy include users, roles, accounts, and services. If the trusted entity is a service principal, as a best practice, the trust policy should not trust more than one AWS service in order to apply least privilege.

Additionally, RCPs can be configured to help ensure that no one from outside your account or organization can be authorized to assume the role. If the trusted entity is an IAM principal, such as a role or user, the policy can use either the `aws:PrincipalOrgID`, `aws:PrincipalOrgPaths`, or `aws:PrincipalAccount` condition. Exceptions can be created for known, external, expected accounts and they should use the `sts:ExternalId` condition.

Refer to resource_control_policies for more details.

Resource sharing and external targets

The final consideration are services that allow resource sharing or targeting external resources. With these services, you cannot use a condition such as `aws:ResourceOrgID` because the resource being evaluated in the policy belongs to your AWS organization, but its configuration specifies a resource that does not.

Instead, you will need to use different approaches to prevent sharing resources with AWS accounts outside of your organization. AWS has several services that allow you to share a resource with another account or target a resource in another account. Some options include:

- **Amazon Machine Images (AMIs)** - AMIs can be shared with other accounts or made public with the `ModifyImageAttribute` API. You can deny this action in an SCP and create an exception for a privileged IAM principal if required.
- **Amazon EBS Snapshots** - Amazon EBS Snapshots can be shared with other accounts or made public with the `ModifySnapshotAttribute` API. You can deny this action in an SCP and create an exception for a privileged IAM principal if required.
- **Amazon RDS Snapshots** - Amazon RDS Snapshots can be shared with other accounts or made public with the `ModifyDBSnapshotAttribute` API. You can deny this action in an SCP and create an exception for a privileged IAM principal if required.
- **AWS Resource Access Manager (AWS RAM)** - AWS RAM is a service that allows sharing various types of resources with other AWS accounts. Sharing with AWS RAM can be constrained to your AWS organization using the `ram:RequestedAllowsExternalPrincipals` IAM condition.
- **Amazon CloudWatch Logs Subscription Filters** - You can send CloudWatch Logs to cross account destinations. You can deny this action in an SCP and create an exception for a privileged IAM principals using a principal tag.
- **Amazon EventBridge Targets** - You can add targets to an EventBridge rule that are in different accounts than the event bus. Use the `events:TargetArn` IAM condition to limit which accounts

can be used as targets for EventBridge rules or create an exception for privileged IAM principals using a principal tag.

AWS RAM allows for conditioning on sharing outside of your AWS organization in an SCP. For the rest of the services, you can use IAM policies to prevent the actions altogether. However, this isn't always practical. In these cases, you can either allow just privileged roles to take those actions, or you can build detective controls and remediate the configuration using services such as AWS Config or EventBridge.

For example, an EventBridge rule can look for `PutSubscriptionFilter` events and invoke a Lambda function to evaluate the destination Amazon Resource Name (ARN) in the subscription. If the ARN is for a resource not in the AWS organization, the function can remove the subscription.

Refer to data perimeter governance policies in service_control_policies for additional examples of APIs that can be used to share resources and sample policies for helping prevent resource sharing.

AWS Management Console

The AWS Management Console can be configured to use a Private Access option. This capability allows you to prevent users from signing in to unintended AWS accounts from within your network. You can use it to limit access to the management console to only a specified set of AWS accounts or organizations. AWS Management Console Private Access helps ensure only trusted identities are allowed from your expected networks. Refer to the documentation for supported AWS Regions, service consoles, and features. Instructions for implementing VPC endpoint policies with the `aws:PrincipalOrgID` condition to enforce only trusted identities and considerations for enforcing only expected networks with `aws:SourceVpc` can also be found in the documentation.

Conclusion

This paper has reviewed how to implement a data perimeter on AWS using SCPs, RCPs, and VPC endpoint policies. These controls are used to ensure **Only Trusted Identities**, **Only Trusted Resources**, and **Only Expected Networks** are allowed access to *my AWS*.

The following is a list of the recommendations made throughout this paper as part of achieving the perimeter's six objectives.

- Use the `aws:PrincipalOrgID` condition in RCPs and VPC endpoint policies to help prevent untrusted identities. Use the `aws:PrincipalIsAWSService` and `aws:SourceOrgID` conditions to restrict access by AWS services so that it is only on your behalf.
- Use the `aws:ResourceOrgID` condition in SCPs to help prevent your IAM principals from accessing untrusted resources. Additionally, add this condition to your VPC endpoint policies as a defense in depth approach. Create exceptions using a `NotAction` list in your SCPs and list explicit resources that should be trusted in your VPC endpoint policies. Use the `aws:CalledVia` condition to allow specific AWS services to access resources you don't own.
- Use an SCP to prevent access from unexpected network locations. Additionally, add similar policy statements to your RCPs as a defense in depth approach. Use the `aws:ViaAWSService` condition to create exceptions when AWS acts on your behalf using your credentials.
- Audit your policies to ensure that permissions guardrails are applied to help prevent misconfiguration. Use IAM Access Analyzer to review resource-based policy configuration and effective permissions on your resources.
- Block all outbound internet access, except for required AWS endpoints and allowed external services that are dependencies for your workloads. This prevents data movement to non-AWS destinations, out-of-Region AWS endpoints, and unintended VPC hosted data plane services (like Amazon RDS instances).
- Route out-of-Region requests through VPC endpoints so that the network boundary controls are consistently applied.
- Where AWS provides an option to run a resource publicly or inside a customer-owned VPC, use the VPC configuration (that is, Amazon OpenSearch Service (OpenSearch Service), Amazon SageMaker AI notebooks, and AWS Lambda) and turn off the public access options (for example, Amazon Redshift and Amazon RDS) in order to use network controls.
- Configure RCPs to limit access to your IAM roles so that they can only be assumed by trusted identities.

- Prevent external resource sharing and targeting external resources with an SCP.
- Use the AWS Management Console Private Access feature to help ensure users can only access intended AWS accounts and organizations from your expected networks.

Appendix A – Proxy configuration example

The following configuration is for a Squid-based proxy running in us-east-1 with peers in us-west-2 and eu-west-1. It denies all other traffic for the `amazonaws.com` domain, but allows all other domains to be forwarded normally. This configuration will need to be kept up to date with global AWS services that do not use a Region name in their domain name.

```
    cache_effective_user squid
prefer_direct off
nonhierarchical_direct off

## Define acls for local networks that are forwarding here
acl rfc_1918 src 10.0.0.0/8        # RFC1918 possible internal network
acl rfc_1918 src 172.16.0.0/12     # RFC1918 possible internal network
acl rfc_1918 src 192.168.0.0/16    # RFC1918 possible internal network
acl localnet src fc00::/7          # RFC 4193 local private network range
acl localnet src fe80::/10         # RFC 4291 link-local (directly plugged) machines
acl localnet src 127.0.0.1         # localhost loopback

## Additional ACLs
acl ssl_ports port 443             # ssl
acl safe_ports port 80             # http
acl safe_ports port 21             # ftp
acl safe_ports port 443            # https
acl safe_ports port 70             # gopher
acl safe_ports port 210            # wais
acl safe_ports port 1025-65535     # unregistered ports
acl safe_ports port 280            # http-mgmt
acl safe_ports port 488            # gss-http
acl safe_ports port 591            # filemaker
acl safe_ports port 777            # multiling http
acl CONNECT method CONNECT

## Define acls for amazonaws.com
acl aws_domain dstdomain .amazonaws.com
acl us_east_1 dstdomain .s3.amazonaws.com
acl us_east_1 dstdomain .sts.amazonaws.com
acl us_east_1 dstdomain .cloudfront.amazonaws.com
acl us_west_2 dstdomain .globalaccelerator.amazonaws.com
acl us_east_1 dstdomain .iam.amazonaws.com
acl us_east_1 dstdomain .route53.amazonaws.com
```

```
acl us_east_1 dstdomain .queue.amazonaws.com
acl us_east_1 dstdomain .sdb.amazonaws.com
acl us_east_1 dstdomain .waf.amazonaws.com
acl us_east_1 dstdomain .us-east-1.amazonaws.com
acl us_east_2 dstdomain .us-east-2.amazonaws.com
acl us_west_2 dstdomain .us-west-2.amazonaws.com
acl eu_west_1 dstdomain .eu-west-1.amazonaws.com
acl us_east_1_alt dstdom_regex \.us-east-1\..*?\.amazonaws.com
acl us_east_2_alt dstdom_regex \.us-east-2\..*?\.amazonaws.com
acl us_west_2_alt dstdom_regex \.us-west-2\..*?\.amazonaws.com
acl eu_west_1_alt dstdom_regex \.eu-west-1\..*?\.amazonaws.com

## Deny access to anything other than SSL
http_access deny !safe_ports
http_access deny CONNECT !ssl_ports

## Now specify the cache peer for each Region
never_direct allow us_east_2
never_direct allow us_east_2_alt
never_direct allow us_west_2
never_direct allow us_west_2_alt
never_direct allow eu_west_1
never_direct allow eu_west_1_alt
cache_peer us-east-2.proxy.local parent 3128 0 no-query proxy-only name=cmh
cache_peer_access cmh allow us_east_2
cache_peer_access cmh allow us_east_2_alt
cache_peer us-west-2.proxy.local parent 3128 0 no-query proxy-only name=pdx
cache_peer_access pdx allow us_west_2
cache_peer_access pdx allow us_west_2_alt
cache_peer eu-west-1.proxy.local parent 3128 0 no-query proxy-only name=dub
cache_peer_access dub allow eu_west_1
cache_peer_access dub allow eu_west_1_alt

# Only allow cachemgr access from localhost
http_access allow localhost manager
http_access deny manager

## Explicitly allow approved AWS Regions so we can block
## all other Regions using .amazonaws.com below
http_access allow rfc_1918 us_east_1
http_access allow rfc_1918 us_east_2
http_access allow rfc_1918 us_west_2
http_access allow rfc_1918 eu_west_1
http_access allow rfc_1918 us_east_1_alt
```

```
http_access allow rfc_1918 us_east_2_alt
http_access allow rfc_1918 us_west_2_alt
http_access allow rfc_1918 eu_west_1_alt

## Block all other AWS Regions
http_access deny aws_domain

## Allow all other access from local networks
http_access allow rfc_1918
http_access allow localnet

## Finally deny all other access to the proxy
http_access deny all

## Listen on 3128
http_port 3128

## Logging
access_log stdio:/var/log/squid/access.log
strip_query_terms off
logfile_rotate 1

## Turn off caching
cache deny all

## Enable the X-Forwarded-For header
forwarded_for on

## Suppress sending squid version information
httpd_suppress_version_string on

## How long to wait when shutting down squid
shutdown_lifetime 30 seconds

## Hostname
visible_hostname aws_proxy

## Prefer ipv4 over v6
dns_v4_first on
```

Contributors

Contributors to this document include:

- Michael Haken, Principal Solutions Architect, Amazon Web Services
- Tatyana Yatskevich, Principal Solutions Architect, Amazon Web Services

Further reading

For additional information, refer to:

- Data perimeter on AWS
- Blog series - Establishing a data perimeter on AWS
- Blog - IAM makes it easier for you to manage permissions for AWS services accessing your resources
- Data Perimeter Workshop
- Data Perimeter Policy Examples
- Centralized access to VPC private endpoints

Document history

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Whitepaper updated	Updated with guidance on how to use resource control policies (RCPs) and `aws:SourceOrgID` condition key to establish a data perimeter	November 13, 2024
Whitepaper updated	Updated to centralize all policy examples in the AWS policy example repo	June 12, 2023
Whitepaper updated	Updated to add guidance for using `aws:ResourceOrgID` , added additional policy examples, and organizational updates	April 26, 2022
Whitepaper updated	Content and policy example updates	September 8, 2021
Initial publication	Whitepaper first published	July 1, 2021

Note

To subscribe to RSS updates, you must have an RSS plug-in enabled for the browser that you are using.

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.