Building an AWS Perimeter

July 1, 2021

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.

Contents

Introduction 1

Perimeter Objectives 2

- AWS Services 3
- Summary 4

Perimeter Overview 5

- Identity Boundary 5
- Resource Boundary 10
- Network Boundary 12
- Preventing Access to Internal Credentials 16
- Cross-Region Requests 14

Conclusion 16

Appendix 1 – IAM Guardrail Policy Examples 17

Appendix 2 – Network Boundary SCP 22

Appendix 3 – Resource Policy Example 24

Appendix 4 – VPC Endpoint Policy Examples 26

- Preventing Unintended Principals 26
- Preventing Unintended Resource Access 28

Appendix 5 – IAM Role Trust Policy Example 30

Appendix 6 - Example Proxy Configuration 31

Contributors 34

Document Revisions 34

Abstract

Many organizations want to create in AWS the same kind of perimeter protections they use in on-premises environments. This paper outlines the best practices and available services for creating a perimeter around your identities, resources, and networks in AWS.

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 (S3) buckets and Amazon Simple Queue Service (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 is a grouping 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" to happen. It refers to the access patterns among your identities, resources, and networks that should be allowed. Using those three elements, we want to make the following assertion to define our perimeter's goal: access is allowed if - and only if[1] - the identity is intended, the resource is intended, and the network is intended.

If any of these conditions are false, then the access inside the perimeter is "unintended" and should be denied. In order to build the perimeter, each component (identity, resource, and network) must implement a boundary to ensure that the necessary conditions are true.

This paper discusses these boundaries in terms of preventing unintended access patterns. It is designed to help customers understand how to use them to create a complete AWS perimeter as part of their responsibility in the AWS Shared Responsibility Model.

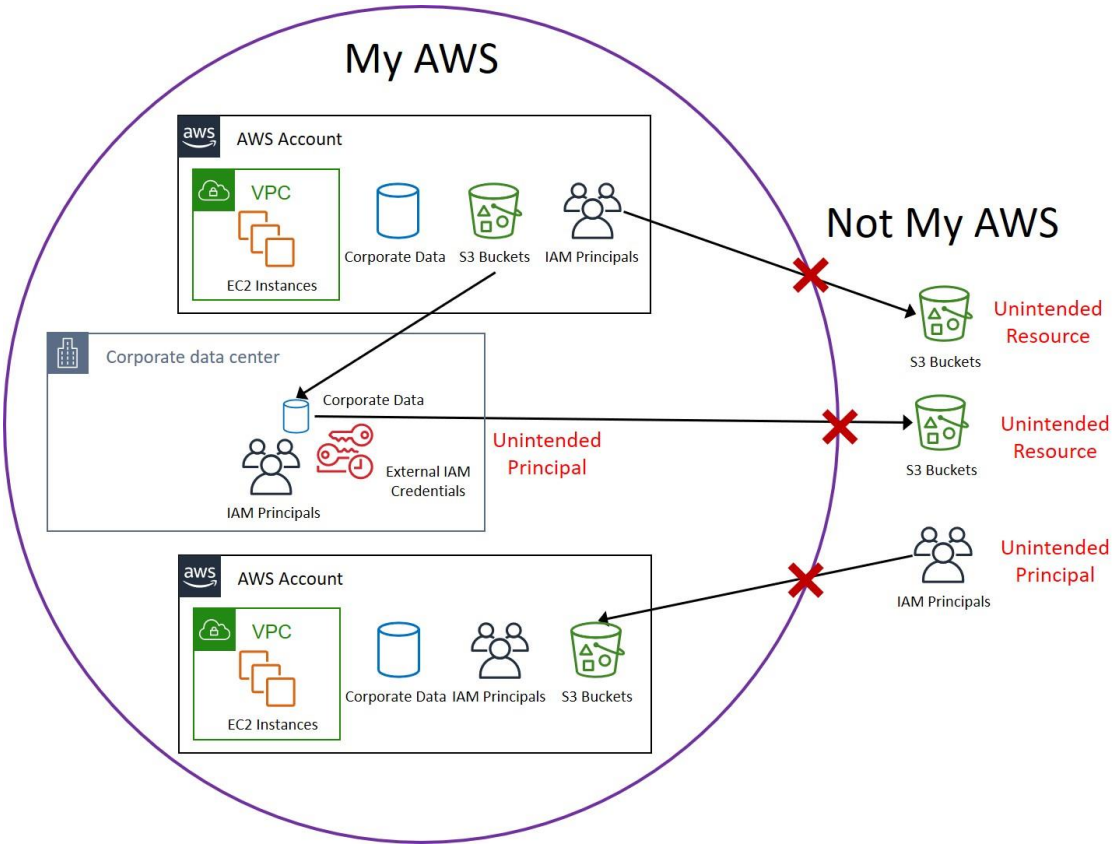

***Figure 1 - AWS perimeter**: A high-level depiction of defining a perimeter around your AWS resources to prevent interaction with unintended [AWS Identity and Access Management (IAM)] principals and unintended resources.*

Perimeter Objectives

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

1. **Only My IAM** - The [AWS Identity and Access Management] (IAM) principals in my AWS organization or AWS acting on my behalf
2. **Only My Resources** - The resources in my AWS organization or resources AWS operates on my behalf
3. **Only My Networks** - Both my VPC and on-premises networks

These are the necessary and sufficient conditions for intended access inside an AWS perimeter to be allowed.

$$\text{Access in the Perimeter} \iff (\text{Only My IAM}) \wedge (\text{Only My Resource}) \wedge (\text{Only My Network})$$

Ensuring the truth of these three conditions ultimately defines the objectives of the perimeter. It also represents three separate boundaries through which we can implement controls.

There is an **Identity Boundary** that specifies resource and network controls, a **Resource Boundary** that specifies identity and network controls, and a **Network Boundary** that specifies identity and resource controls.

Thus, each boundary supports two of the three overall objectives to prevent unintended access patterns. The following table outlines how each perimeter objective is supported in each boundary.

Boundary	Perimeter Objective	Purpose
Identity	**Only My Resources**	My IAM principals can only access resources that are part of “my AWS”
	Only My Networks	My IAM principals can only access resources from expected networks
Resource	**Only My IAM**	Only IAM principals that are part of “my AWS” can access my resources
	Only My Networks	My resources can only be accessed from expected networks
Network	**Only My IAM**	Only IAM principals that are part of “my AWS” can access resources from my networks
	Only My Resources	Only resources that are part of “my AWS” can be accessed from my networks

The next section will outline the AWS services used to implement the perimeter.

AWS Services

There are several services that will be used to create the controls in each boundary to meet each of the perimeter objectives. We will analyze these services based on how they support each boundary.

- **Identity Boundary** – The identity boundary is built from two types of policies, **IAM identity-based policies** and AWS Organizations **Service Control Policies** (SCP). Identity-based policies are applied directly to IAM principals to define their permissions. SCPs are a type of organization policy that you can use to manage permissions in your organization and are applied to your identities. SCPs offer central control over the maximum available permissions for all accounts in your organization. This control includes the ability to define the expected network locations for interaction with intended AWS resources.
- **Resource Boundary** – The resource boundary is defined through **resource-based policies**. These policies are applied to AWS resources and define which IAM principals can interact with the resource, as well as what the expected networks are for access.
- **Network Boundary** – The network boundary is defined through the use of **VPC endpoints** and **VPC endpoint policies** and must include both VPC and on-premises networks. A VPC endpoint enables a private VPC connection to supported AWS services without requiring an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Most endpoints also support applying an endpoint policy. This is an additional IAM policy that is evaluated against all requests, regardless of whether the IAM principal or resource involved in the request is part of “my AWS”. It is important because these policies are evaluated against all IAM principals, where identity-based policies and SCPs only apply to IAM principals that are part of your organization.

Objectives Summary

In summary, the objectives of the perimeter are to ensure that access involves **Only My IAM**, **Only My Resources**, and **Only My Networks**. If each boundary achieves the stated objectives for all principals, resources, and networks, then the perimeter prevents unintended principals from accessing your resources, prevents your principals from accessing unintended resources, and finally, prevents data from being moved outside the perimeter to another AWS resource. This table summarizes the boundary and objectives mapping to the AWS services used to implement them.

Boundary	Perimeter Objective	AWS Services Used
Identity	Only My Resources Only My Networks	Identity-based policies and SCPs
Resource	Only My IAM Only My Networks	Resource-based policies
Network	Only My IAM Only My Resources	VPC Endpoint Policies

Perimeter Overview

This section describes the complete perimeter solution by evaluating how each boundary is implemented using the previously mentioned AWS services. Each boundary section will describe the overall solution and how it achieves each associated perimeter objective. They will also provide an appendix reference with detailed examples and demonstrate how the boundary prevents the unintended access pattern for each objective.

Identity Boundary

The identity boundary consists of policies applied to the IAM principals that are part of “my AWS” and ensures that they only access intended resources (Only My Resources) from expected networks (Only My Networks).

Only My Resources

This objective ensures that your intended IAM principals can’t access resources in AWS accounts outside the perimeter. All access to unintended AWS resources involves cross-account access (since those resources exist in an AWS account that is not part of your AWS organization). For requests made from one AWS account to another, the requester in Account A (an account in “my AWS”) must have an identity-based policy that allows them to make a request to the resource in Account B (an account outside of

your perimeter). Also, the resource-based policy in Account B must allow the requester in Account A to access the resource.

If policies in both accounts don't allow the operation, the request fails. Thus, by implementing least privilege identity-based policies, focused on reducing standalone wildcards in Resource statements (`"Resource": "*"`) and explicitly listing allowed resources when possible, customers can significantly reduce, if not eliminate, the risk of accessing unintended resources.

In addition to least privilege identity-based policies, SCPs are designed to provide hard guardrails on what resources your IAM principals can access as a defense in depth approach to limiting resource access.

SCPs configured as deny lists can limit the scope of access to resources in specific accounts that are part of your organization by specifying resources like `"arn:aws:ec2:*:123456789012:*"` for actions that support specifying resource types and using the `s3:ResourceAccount` condition for Amazon S3 resources.

The following diagram demonstrates how this control prevents your identities from accessing unintended resources.

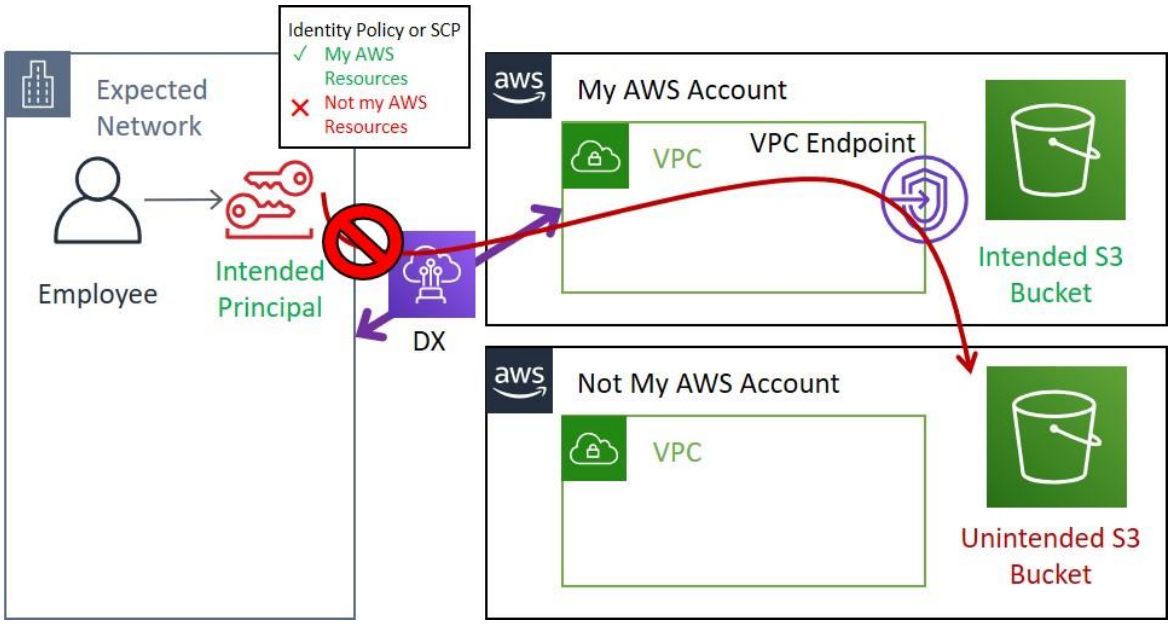

***Figure 2 - Preventing Access to Unintended Resources**: Identity-based policies and SCPs can restrict access to only intended resources.*

In some cases, you may need to interact with resources that are owned by AWS and are not a part of your AWS organization, typically an Amazon S3 bucket, AWS Systems

Manager Parameter Store parameters, or an Amazon SNS topic. These resources will need to be explicitly allowed in the policies you create. See *Appendix 1 – IAM Guardrail Policy Example,* which provides details of how to create both an identity-based policy and SCP that restricts access to resources in specific AWS accounts.

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.

Customers often need this anonymous access for users to browse internet websites. It is also used for workloads that may need to access public data (such as package repositories hosted on Amazon S3 or agent downloads). Thus, it makes it impractical to deny all anonymous access to Amazon S3 resources.

From a data movement standpoint, customers can allow anonymous `GetObject` API calls to all buckets in a policy since the API cannot be used to move data out of a customer network. 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. The *Appendix 1 – IAM Guardrail Policy Example* also includes details of how to allow anonymous `GetObject` API calls while enforcing authentication and least privilege to intended resources for the remainder of Amazon S3 actions.

Only My Networks

Customers can support this objective with an SCP, which applies to all of their intended principals and specifies the expected locations for data access. This forces data movement through your Network Boundary controls to prevent movement outside of the perimeter using unintended identities. See *Appendix 2 – Network Boundary SCP* for an example SCP that implements network controls. The following diagram shows how this control prevents access from unexpected network locations.

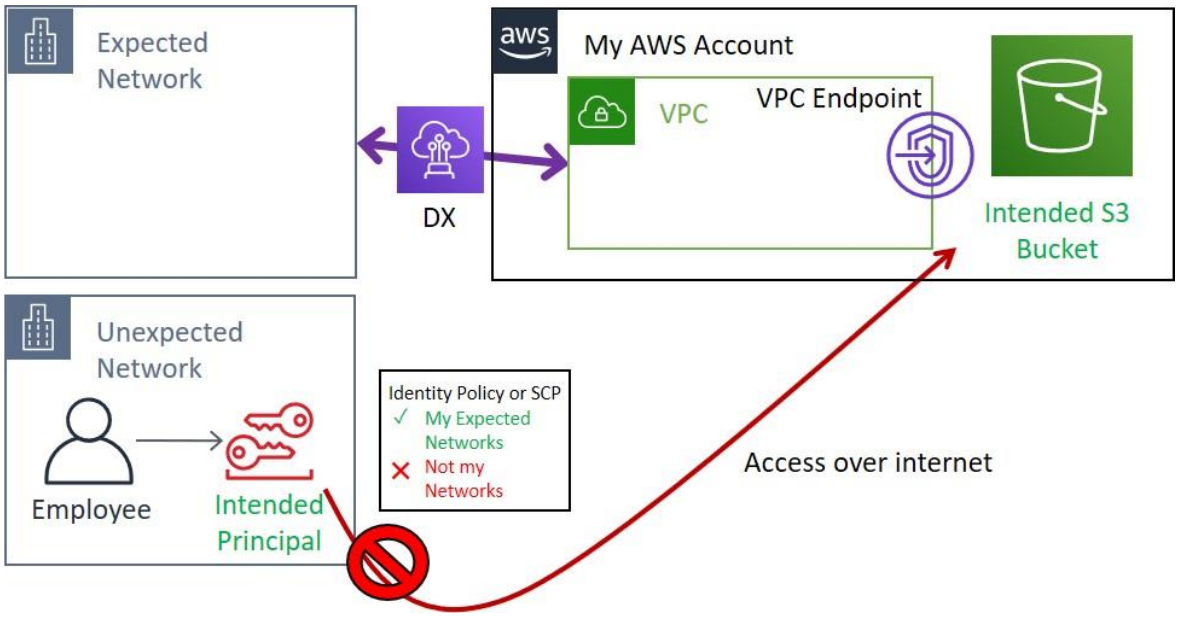

Figure 3 - Preventing Intended Principals from operating in Unexpected Networks*: An AWS Organizations SCP or identity-based policy can be used to prevent intended principals from accessing resources from unexpected networks.*

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 this control.

For example, AWS CloudFormation provides the ability for customers to define a template of resources that AWS orchestrates the creation, update, and deletion of those resources. The initial request to create a CloudFormation stack will originate from an expected network, but the subsequent requests for each resource in the template are made by the CloudFormation service in an AWS network.

The `aws:ViaAWSService` IAM policy condition provides a way to implement an exception for some of these common scenarios where your IAM credentials are used in the requests. *Appendix 2 – Network Boundary SCP* includes details on how to write such exceptions.

The last consideration in implementing network controls is AWS services that run code you provide in compute environments that are not part of your network. For example, Lambda functions or SageMaker Notebooks both provide an option to run on AWS-owned networks.

Most 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 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 like 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 identity authentication and authorization systems to access those services, like AWS Secrets Manager for RDS access, when possible.

Mobile Devices

In on-premises networks, there are some resources that are physically static, such as servers. Other resources like 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 access pattern allows access to unintended resources and is a use case that customers 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 the solution requires forcing access to AWS resources from non-mobile assets.

Amazon WorkSpaces offers a virtual desktop infrastructure (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 Boundary as other resources in AWS VPCs. VDI solutions can also be operated by customers natively using Amazon Elastic Compute Cloud (EC2) instances in a VPC.

Resource Boundary

The resource boundary consists of resource-based policies applied to the AWS resources that are part of “my AWS” and ensures that they are only accessed by intended identities (Only My IAM) from expected networks (Only My Networks).

Some resources in AWS support resource-based policies (like Amazon S3 Bucket Policies), meaning that in addition to authorization through identity-based policies, these resources can define an access policy that is directly associated with the resource. These are commonly used to provide cross-account access, and can be used to authorize external AWS credentials or anonymous access. Although resource-based policies do not allow unintended access by default, a misconfigured policy might unintentionally grant access to an unintended principal or unexpected network. To ensure that only intended principals can access your resources from expected locations, you can implement standardized resource-based policies as a compensating control against misconfiguration.

Only My IAM

The standardized control policy should limit access to intended IAM principals by specifying the `aws:PrincipalOrgId` IAM policy condition in the resource policy. You can implement a more granular restriction with the `aws:PrincipalAccount` or `aws:PrincipalOrgPaths` IAM policy conditions as well. To ensure that your resource policies only allow the intended access, customers can use IAM Access Analyzer for supported resources to identify resource-based policies that are too permissive.

In certain cases, AWS services may use an IAM principal that is outside of your organization, specifically a service principal, to perform actions on your behalf. For example, AWS CloudTrail uses the IAM service principal `cloudtrail.amazonaws.com` to deliver logs to your Amazon S3 bucket. These are intended actions, but need to be explicitly allowed in your resource-based policies. You can do this with the `aws:PrincipalIsAWSService` condition.

See *Appendix 3 – Resource Policy Example* for a template of a standard policy statement you can add to all resource-based policies to achieve the Only My IAM objective as well as create necessary exceptions in the Resource Boundary.

The following diagram demonstrates how this control prevents unintended principals from accessing your resources.

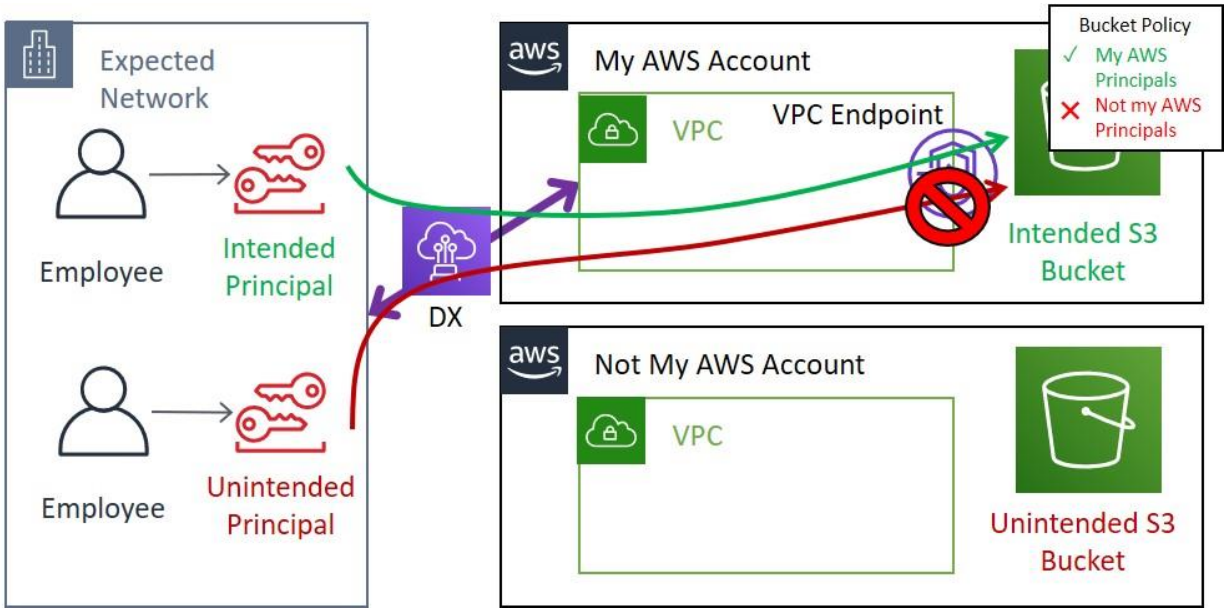

***Figure 4 - Preventing Unintended Principals**: The Amazon S3 bucket policy (a resource-based policy) only allows intended principals.*

Only My Networks

The resource-based policy might also optionally specify what network locations are expected sources for access by using the `aws:SourceIp` and `aws:SourceVpc` (or `aws:SourceVpce`) conditions. This control is optional because the Identity Boundary and Only My IAM objective control in this boundary provide the same outcome. Only your IAM principals are allowed to access this resource (as defined in the resource-based policy). The organization SCP defines the expected networks your principals are allowed to access the resource from. Those controls have indirectly also achieved the Only My Networks objective in the Resource Boundary.

The following diagram shows how this control prevents access from unexpected networks.

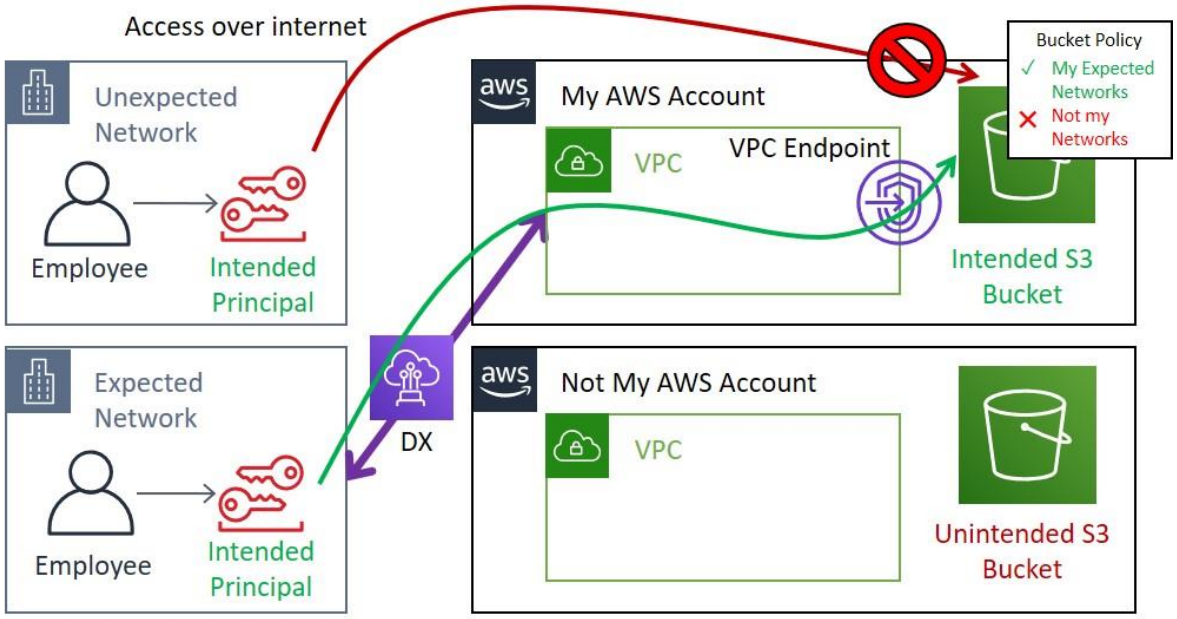

***Figure 5 - Preventing Access from Unexpected Networks**: The S3 bucket policy (a resource-based policy) prevents access from unexpected network locations.*

Network Boundary

The network boundary consists of VPC endpoint policies applied to VPC endpoints in expected networks (your VPCs) that ensure only intended identities (Only My IAM) can access intended resources (Only My Resources) from your expected networks.

This boundary's purpose is to prevent data from moving to unintended resources outside the perimeter by unintended IAM principals whom are not subject to your IAM identity-based policies or SCPs.

VPC endpoint policies provide a mechanism to prevent actions by unintended principals in both your VPC and on-premises networks. In VPC networks, traffic is routed to VPC endpoints automatically if you are using AWS provided DNS.

For on-premises networks, you can also route AWS traffic through VPC endpoints if they are connected to AWS via AWS Direct Connect or VPN. For services that have PrivateLink interface endpoints, you can route traffic to those endpoints directly from an on-premises network. When using an Amazon DynamoDB that only provides a gateway endpoint, you can use a proxy fleet as a way to route traffic from on-premises over that endpoint. This control ensures that unintended principals can't move data outside your network perimeter to other AWS locations.

Only My IAM

VPC endpoint policies can prevent the use of unintended identities by specifying the `aws:PrincipalOrgId` IAM policy condition in your network boundary. You can also implement more granular controls with the `aws:PrincipalAccount` or `aws:PrincipalOrgPaths` conditions. This condition prevents the use of any identity that is not part of your organization. See *Appendix 4 – VPC Endpoint Policy Examples* for best practices on implementing VPC endpoint policies. The following diagram shows how this control prevents access by unintended principals from expected networks.

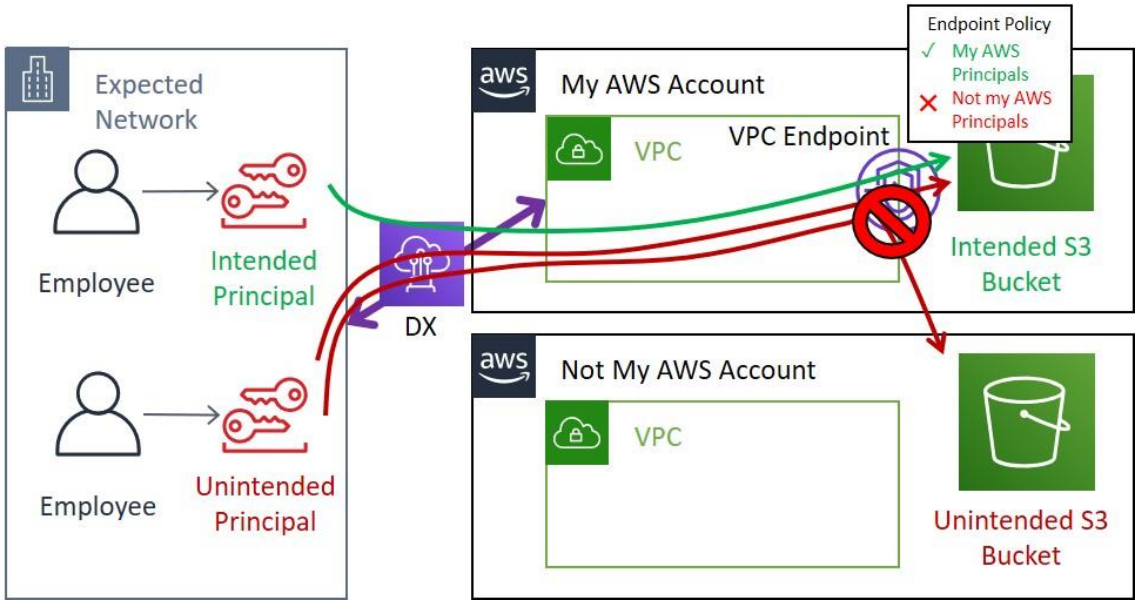

***Figure 6 - Preventing Unintended Principals from Inside an Expected Network**: Unintended principals are blocked at the VPC endpoint by using a condition statement in the endpoint policy requiring that the credentials belong to the AWS account hosting the resource or to the AWS Organization owned by the customer.*

Only My Resources

An endpoint policy can also be used to prevent access to unintended resources in a similar way that identity-based policies or SCPs do. However, this control is optional because the previous control that only allows intended identities to operate in your expected networks - combined with the SCP in the Identity Boundary that implements Only My Resources - indirectly accomplishes the same outcome of Only My Resources in the Network Boundary. The following diagram shows how this control prevents your identities from accessing unintended resources.

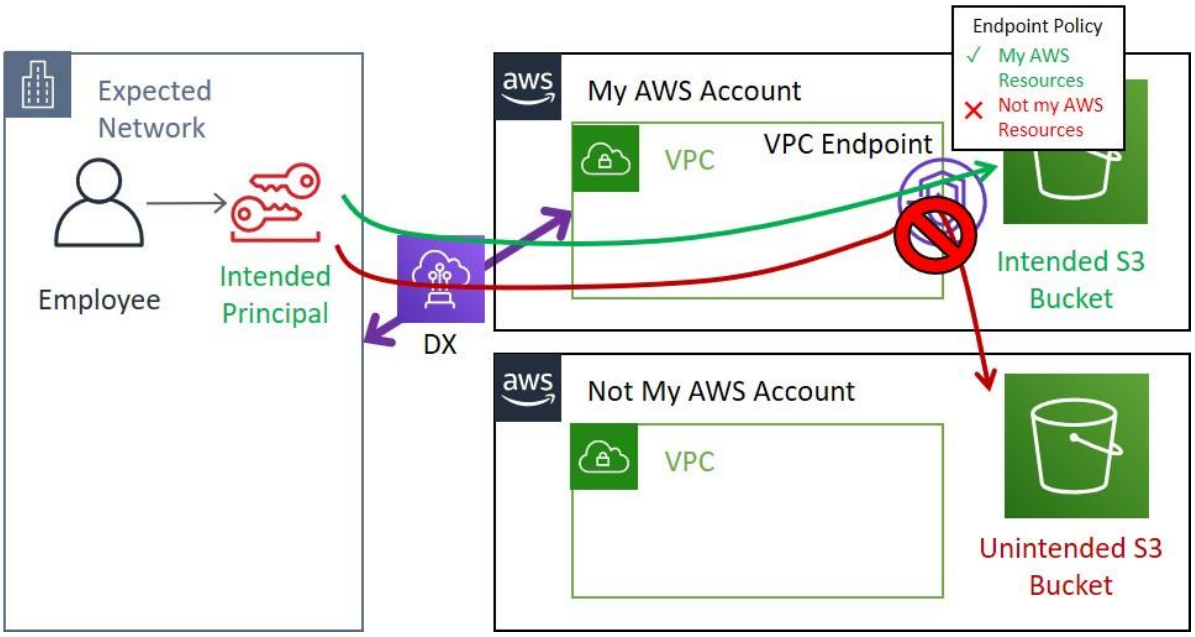

***Figure 7 - Preventing Access to Unintended Resources from an Expected Network**: Unintended resources are blocked at the VPC endpoint through using a resource statement or condition in the endpoint policy requiring that the resources belong to the AWS account of the network or belong to the AWS Organization owned by the customer.*

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 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, customers can handle cross-Region requests in two ways using a proxy.

- Prevent cross-Region API calls. This does not require inspecting TLS and can be done by looking at the hostname in the CONNECT request or, if using Server Name Indication (SNI), the hostname presented in the `ClientHello`, since the AWS Region is included in the domain name of the URL (with the exception of some services that only provide a single control plane endpoint such as IAM or Route53).
- Forward out of Region requests through the proxy. There are two options available for this solution.

- The local proxy can forward traffic to a peer proxy running in a VPC in the requested Region. The out-of-Region proxy sends the traffic to the appropriate VPC endpoint in its Region. See *Appendix 6 - Example Proxy Configuration* for an example proxy configuration that implements this *proxy-chaining* solution.
- The local proxy uses AWS-provided VPC DNS. An Outbound Amazon Route 53 Resolver Rule directs all out-of-Region domain names to an Inbound Route 53 Resolver endpoint in a VPC in the requested Region. This resolver endpoint is co-located in the same VPC, with the necessary VPC endpoints. The resolver returns the IP address of the appropriate VPC endpoint and the local proxy sends the traffic directly to the VPC endpoint in the requested Region.

The following diagram demonstrates a high-level reference architecture for managing out of Region requests with *proxy-chaining*.

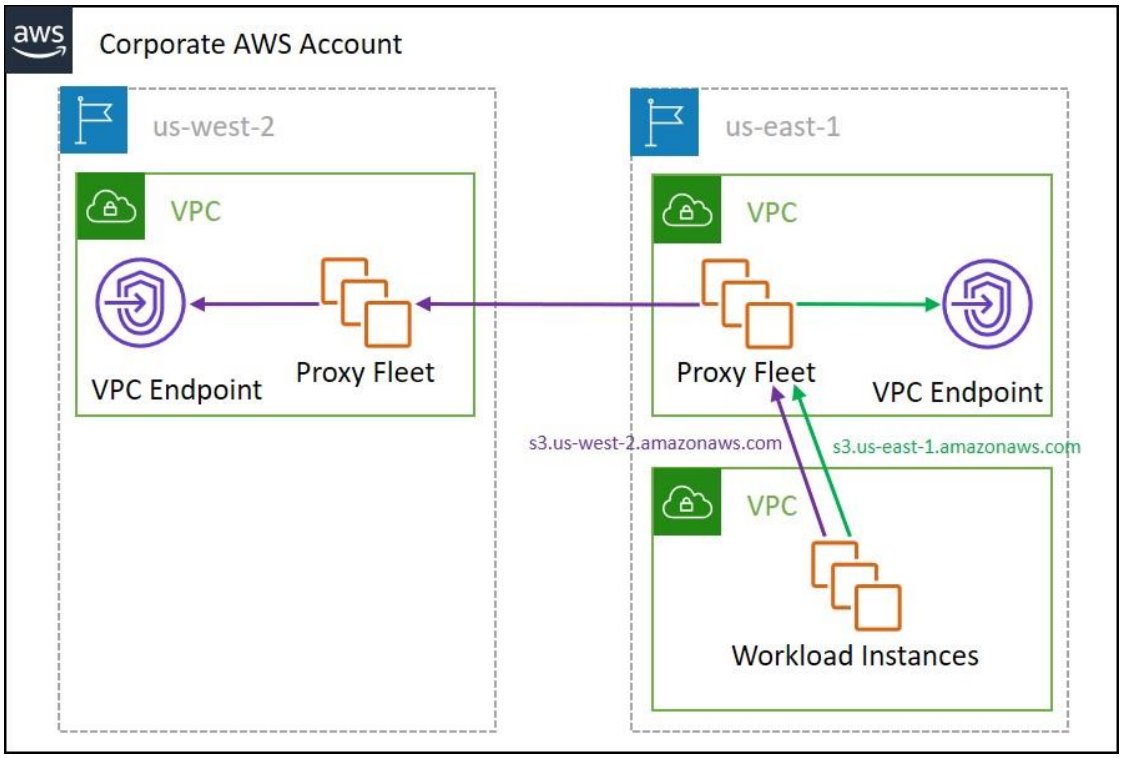

Figure 8 - Using Proxy-Chaining to Send Out-Of-Region Requests through VPC Endpoints*: Workloads send their HTTPS traffic to a proxy in the same Region. That proxy sends "in-Region" requests to the appropriate VPC endpoint and forwards "out-of-Region" requests to a peer proxy.*

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.

The trust policy can be configured to ensure that no one from outside the customer's account or organization can be authorized to assume the role. Customers should audit all IAM role trust policies and ensure either of the following are true.

- The trust policy uses either the `aws:PrincipalOrgId` or `aws:PrincipaOrgPaths` condition when the trusted entity is an IAM principal, such as a role or user. Exceptions can be created with an allow list of known, external, expected accounts and they should use the `sts:ExternalId` condition.
- The trusted entity is an AWS service, being either a service principal or IAM service-linked role. A trust policy should not trust more than one AWS service for least privilege reasons.

See *Appendix 5 – IAM Role Trust Policy Example* for more details.

Conclusion

This paper has reviewed how VPC endpoints with policies, resource and identity-based policies, and SCPs are effective tools for creating a perimeter around "my AWS".

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

- Use least privilege IAM identity-based policies and SCPs to prevent intended principals from accessing unintended resources.
- Use an SCP to limit access to resources from expected network locations.

- Add defense in depth to resources that support resource-based policies by specifying the `aws:PrincipalOrgId` condition to limit access to intended principals and optionally conditions to limit access to expected network locations. Audit all resource policies to ensure that these coarse-grained controls are applied to prevent misconfiguration. Use IAM Access Analyzer to review resource-based policy configuration. Use the `aws:PrincipalIsAWSService` condition to create exceptions in resource policies for AWS services.
- Use VPC endpoints and endpoint policies to prevent unintended principals when interacting with AWS resources from your networks by using `aws:PrincipalOrgId` as a condition in each statement. Also, use VPC endpoint policies to prevent access to unintended 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 RDS instances).
- Use the `s3:ResourceAccount` condition to limit access to buckets in specific AWS accounts.
- Where required, implement proxy-based solutions to manage out-of-Region requests so that 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 Elasticsearch Service (Amazon ES), Amazon SageMaker notebooks, and AWS Lambda) and turn off the public access options (for example, Amazon Redshift and RDS) in order to use Network Boundary controls.
- Configure IAM Role Trust Policies with condition statements, limiting access to only intended principals when the trusted entity is an IAM principal (as opposed to an AWS service principal).

Appendix 1 – IAM Guardrail Policy Examples

The following provides an example of an IAM identity-based policy that provides access to Simple Notification Service (Amazon SNS) resources belonging to a specific set of accounts (could be one or more).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SNSLimitCrossAccountAccess",
      "Action": [
        "sns:AddPermission",
        "sns:ConfirmSubscription",
        "sns:CreateTopic",
        "sns:DeleteTopic",
        "sns:GetTopicAttributes",
        "sns:ListSubscriptionsByTopic",
        "sns:ListTagsForResource",
        "sns:Publish",
        "sns:RemovePermission",
        "sns:SetTopicAttributes",
        "sns:Subscribe",
        "sns:TagResource",
        "sns:UntagResource"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:*:123456789012:*",
        "arn:aws:sns:*:987654321098:*"
      ]
    },
    {
      "Sid": "SNSAllowActionsWithoutAResource",
      "Action": [
        "sns:CheckIfPhoneNumberIsOptedOut",
        "sns:CreatePlatformApplication",
        "sns:CreatePlatformEndpoint",
        "sns:DeleteEndpoint",
        "sns:DeletePlatformApplication",
        "sns:GetEndpointAttributes",
        "sns:GetPlatformApplicationAttributes",
        "sns:GetSMSAttributes",
        "sns:GetSubscriptionAttributes",
        "sns:ListEndpointsByPlatformApplication",
        "sns:ListPhoneNumbersOptedOut",
        "sns:ListPlatformApplications",
        "sns:ListSubscriptions",
        "sns:ListTopics",
        "sns:OptInPhoneNumber",
```

```
        "sns:SetEndpointAttributes",
        "sns:SetPlatformApplicationAttributes",
        "sns:SetSMSAttributes",
        "sns:SetSubscriptionAttributes",
        "sns:Unsubscribe"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

You can use this same pattern for constraining access to just your resources for other services that support resource-based policies and cross-account access, such as Amazon Simple Queue Service (SQS), AWS Lambda and Simple Email Service (Amazon SES).

The next policy example allows anonymous access to get data from public S3 buckets, while preventing access for other actions on resources that don't belong to the specified accounts.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAnonymousGetObject",
      "Action": "s3:GetObject",
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "AllowS3InteractionWithSpecificAccounts",
      "Action": "s3:*",
      "Effect": "Allow",
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
```

```
            "s3:ResourceAccount": [
              "123456789012",
              "987654321098"
            ]
          }
        }
      }
    ]
}
```

Because the preceding policies use a condition for Amazon S3 actions and specify resources for Amazon SNS, they can't be used as an SCP. To use them as an SCP, they can be rewritten as a deny list to prevent access to unintended resources.

In this case, the example assumes the default FullAWSAccess policy is in place, so we only need to include the actions that need to be explicitly denied. You may also need to exempt certain AWS-owned Amazon S3 buckets from this policy and explicitly allow access to them.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SNSLimitCrossAccountAccess",
      "Action": [
        "sns:AddPermission",
        "sns:ConfirmSubscription",
        "sns:CreateTopic",
        "sns:DeleteTopic",
        "sns:GetTopicAttributes",
        "sns:ListSubscriptionsByTopic",
        "sns:ListTagsForResource",
        "sns:Publish",
        "sns:RemovePermission",
        "sns:SetTopicAttributes",
        "sns:Subscribe",
        "sns:TagResource",
        "sns:UntagResource"
      ],
      "Effect": "Deny",
      "NotResource": [
        "arn:aws:sns:*:123456789012:*",
        "arn:aws:sns:*:987654321098:*"
```

```
        ]
      },
      {
        "Sid": "LimitS3InteractionWithSpecificAccounts",
        "NotAction": [
          "s3:GetObject"
        ],
        "Effect": "Deny",
        "Resource": [
          "arn:aws:s3:::*",
          "arn:aws:s3:*:*:accesspoint/*",
          "arn:aws:s3:*:*:job/*",
          "arn:aws:s3:*:*:storage-lens/*",
          "arn:aws:s3-object-lambda:*:*:accesspoint/*"
        ],
        "Condition": {
          "StringNotEquals": {
            "s3:ResourceAccount" : [
              "123456789012",
              "987654321098"
            ]
          }
        }
      }
    ]
}
```

Here's a summary of how this SCP works:

- Statement 1
  - "Action": [] – This policy applies to all Amazon SNS actions that support specifying a resource type, and hence, cross-account access.
  - "NotResource": [ ] – This policy applies to all resources not listed here, effectively exempting the resources in the two identified accounts from the deny (which are explicitly allowed by the FullAWSAccess policy).
  - "Effect": "Deny" – The policy denies when the action is one of the listed items and when the resource is not one explicitly listed. Thus, this provides a guardrail to prevent access to resources that aren't specifically allow-listed in the policy.
- Statement 2

- "NotAction": "s3:GetObject" – This policy applies to all actions, except GetObject, which we still want to allow anonymously.
- "Resource": [ ] – This policy applies to all Amazon S3 resources and also constrains the actions to just those that target S3 resources.
- "Effect": "Deny" – This policy denies all S3 actions, except GetObject, to all S3 resources, unless the condition is met.
- "Condition": { } – If the S3 resources are not part of an account listed in the condition, then the action is denied, effectively only allowing authenticated access to S3 resources in the accounts specified while still allowing anonymous GetObject permissions.

This same policy can be incorporated into a VPC endpoint policy as shown in *Appendix 4 – VPC Endpoint Policy Examples*.

Appendix 2 – Network Boundary SCP

This policy can be applied once at the organization root level, in which case, you'll need to scale the `aws:SourceVpc` condition to include VPCs from all of your accounts. You can also apply this policy in a more granular way at an organizational unit or individual account level (meaning you would have multiple SCPs of this type deployed).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "NotAction": [
        "es:ESHttp*",
        "dax:PutItem",
        "dax:Query",
        "dax:Scan",
        "dax:GetItem",
        "dax:DeleteItem",
        "dax:BatchGetItem",
        "dax:BatchWriteItem",
        "dax:ConditionCheckItem",
        "iam:PassRole"
      ],
      "Resource": "*",
      "Condition": {
```

```
          "NotIpAddressIfExists": {
            "aws:SourceIp": [
              "192.0.2.0/24",
              "203.0.113.0/24"
            ]
          },
          "StringNotEqualsIfExists": {
            "aws:SourceVpc": [
              "vpc-012abc01",
              "vpc-023edf34"
            ]
          },
          "Bool": {
            "aws:ViaAWSService": "false"
          },
          "Null": {
            "aws:PrincipalTag/IpRestrictedExempt": true
          },
          "ArnNotLike": {
            "aws:PrincipalArn": "arn:aws:iam::*:role/aws:ec2-
infrastructure/*"
          }
        }
      }
    ]
}
```

This is a summary of the SCP contents. The policy denies all actions to all resources, except for the actions listed in the `NotAction` section. They are listed because Amazon DynamoDB Accelerator (DAX) and Amazon Elasticsearch Service (Amazon ES), when configured as a VPC domain, do not present a public IP address or transit a VPC endpoint and cannot be controlled with a source IP condition.

Additionally, `iam:PassRole` is a virtual permission, meaning there is not an associated API endpoint, and thus does not have a source IP address presented when policy authorization is evaluated. Because each condition operator is evaluated with a logical `AND`, every condition must evaluate to true for the policy to deny the action. Thus, any one of the conditions evaluating to false will permit the action. In that light, the conditions can be viewed as exceptions to the policy.

- `aws:SourceIp` – The action is allowed if it originates from one of the listed subnets. Customers should replace these IPs with the IPs of their NAT Gateways, EIPs, and on-premises public IP space.

- `aws:SourceVpc` – When customers have VPC endpoints implemented, they should replace these values with the VPC IDs of their own VPCs.
- `aws:ViaAWSService` – Some services, such as CloudFormation, perform actions on a user's behalf by using their credentials and will not present customer IP addresses that they have listed in the "`aws:SourceIp`" condition block. This condition allows those services to still initiate those actions without being restricted to the customer network.
- `aws:PrincipalTag` – Using a standard tag on IAM principals allows customers to exempt them from this policy if needed (for example, to support using EC2). Customers should replace the tag key with the tag that they will use to exempt principals.
- `aws:PrincipalArn` – AWS Elastic Computer Cloud (EC2) uses special infrastructure IAM roles to perform actions on customers' behalf per EC2 instance that does not present a predictable public IP address and can be safely exempted from this policy. The assumed role will look something like this: `arn:aws:sts::123456789012:assumed-role/aws:ec2-infrastructure/i-07d8bc39180cd7268`. The role name uses a "`:`" character, which is unallowed for normal roles, so it cannot be spoofed by another customer.

Appendix 3 – Resource Policy Example

The following provides an example that can be used in a resource policy to prevent access by unintended principals.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PreventUnintendedPrincipals",
      "Action": "*",
      "Effect": "Deny",
      "Principal": "*",
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        }
      }
    }
```

```
    ]
}
```

This policy is similar to the policy you would implement in a VPC endpoint policy to prevent unintended principals that do not belong to your organization.

There are expected situations when AWS uses an IAM service principal instead of an IAM role to interact with your resources. An example would be AWS CloudTrail log delivery to Amazon S3. The service principal is not part of your AWS organization like IAM roles are, so it needs to be excepted from the restriction. You cannot use a `NotPrincipal` statement with an AWS service principal, so you can instead use the `aws:PrincipalIsAWSService` condition. This provides an example of an S3 bucket policy for CloudTrail logs that ensures no one outside of your organization can access the bucket, except for the CloudTrail service principal.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PreventUnintendedPrincipalsButAllowCloudTrail",
      "Action": "*",
      "Effect": "Deny",
      "Principal": "*",
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        },
        "Bool": {
          "aws:PrincipalIsAWSService": "false"
        }
      }
    },
    {
      "Sid": "AllowCloudTrailToGetACL",
      "Action": "s3:GetBucketAcl",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "cloudtrail.amazonaws.com"
```

```
        ]
      },
      "Resource": [
        "arn:aws:s3:::bucketname"
      ]
    },
    {
      "Sid": "AllowCloudTrailToPutLogs",
      "Action": "s3:PutObject",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "cloudtrail.amazonaws.com"
        ]
      },
      "Resource": [
        "arn:aws:s3:::bucketname/AWSLogs/123456789012/*"
      ],
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": "bucket-owner-full-control"
        }
      }
    }
  ]
}
```

Appendix 4 – VPC Endpoint Policy Examples

Preventing Unintended Principals

The following is an example of a VPC Endpoint policy for Amazon DynamoDB that restricts access to credentials that are part of the customer's AWS Organization as a way to prevent unintended principals.

```
{
  "Statement": [
    {
      "Sid": "PreventUnintendedPrincipals",
      "Principal": "*",
      "Action": [
```

```
        "dynamodb:*"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        }
      }
    }
  ]
}
```

The policy could also be written with two sections in the statement, making the condition part of an explicit deny. See IAM Policy Evaluation Logic for an explanation of how these policies are evaluated.

```
{
  "Statement": [
    {
      "Sid": "PreventUnintendedPrincipals",
      "Principal": "*",
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        }
      }
    },
    {
      "Sid": "AllowAllDynamoDB",
      "Principal": "*",
      "Action": "dynamodb:*",
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Preventing Unintended Resource Access

These policies can also be combined with explicit resource statements to additionally deny access to unintended resources in the same statement.

```
{
  "Statement": [
    {
      "Sid": "PreventUnintendedPrincipalsAndResources",
      "Principal": "*",
      "NotAction": [
        "dynamodb:DescribeLimits",
        "dynamodb:DescribeReservedCapacity",
        "dynamodb:DescribeReservedCapacityOfferings",
        "dynamodb:ListBackups",
        "dynamodb:ListContributorInsights",
        "dynamodb:ListExports",
        "dynamodb:ListGlobalTables",
        "dynamodb:ListStreams",
        "dynamodb:ListTables",
        "dynamodb:PurchaseReservedCapacityOfferings"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:dynamodb:*:123456789012:*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        }
      }
    },
    {
      "Sid": "PreventUnintendedPrincipals",
      "Principal": "*",
      "Action": [
        "dynamodb:DescribeLimits",
        "dynamodb:DescribeReservedCapacity",
        "dynamodb:DescribeReservedCapacityOfferings",
        "dynamodb:ListBackups",
        "dynamodb:ListContributorInsights",
        "dynamodb:ListExports",
        "dynamodb:ListGlobalTables",
```

```
        "dynamodb:ListStreams",
        "dynamodb:ListTables",
        "dynamodb:PurchaseReservedCapacityOfferings"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        }
      }
    }
  ]
}
```

We can also create a similar Amazon S3 endpoint policy using the `s3:ResourceAccount` condition.

```
{
  "Statement": [
    {
      "Sid": "PreventUnintendedPrincipalsAndResources",
      "Principal": "*",
      "Action": "*",
      "Effect": "Allow",
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453",
          "s3:ResourceAccount": [
            "123456789012"
          ]
        }
      }
    }
  ]
}
```

Appendix 5 – IAM Role Trust Policy Example

The following is an example of a policy that ensures all principals that are allowed to assume the role are part of the customer's organization with the exception of 1 external account, 123456789012, that uses a customer-provided external id of "12345".

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgId": "o-4tkekae453"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "123456789012"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "12345"
        }
      }
    }
  ]
}
```

You can also further limit the AWS accounts inside your organization that are allowed to assume the role by listing them in the "AWS" principal field instead of the wildcard "*" in the first statement.

Appendix 6 - Example Proxy Configuration

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.

```
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

Document Revisions

Date	Description
July 2021	First publication

Notes

[1] https://en.wikipedia.org/wiki/If_and_only_if