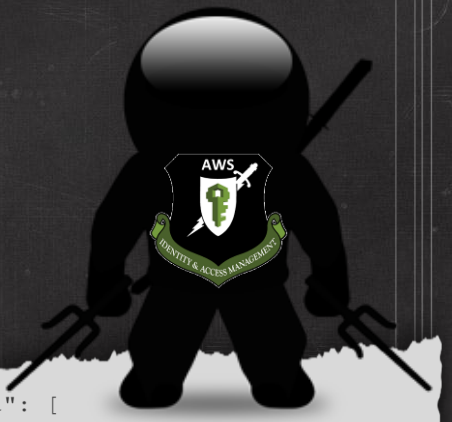
AWS re:Invent

Mastering Access Control Policies

Jeff Wierer, Identity and Access Management

November 13, 2013




```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:Get*", "s3:List*"],
      "Resource": "*"
    }
  ]
}
```

amazon web services

Goals

- Know more about securing your AWS resources
- Get a deeper understanding of the policy language
- Learn some tips and tricks for most frequently asked tasks
- Keep this a lively session via demos
  - Amazon S3
  - AWS IAM
  - Amazon EC2
  - Amazon DynamoDB

Before getting too deep… Let's level set on Identity and Access Management

Why IAM?

- One of customers' biggest concerns when moving to the cloud

CONTROL

- What do I do if…
  - I want to control "Who can do what"?
  - I want to implement security best practices?
  - I want to be at least as secure as on premises?
  - One of my employees leaves the company?

IAM Provides Granular Control to your AWS Account

You can grant or deny access by defining:

- Who can access your resources
- What actions they can take
- Which resources they can access
- How will they access your resources

This is described using a policy language

The Access Control Policy Language

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:Get*", "s3:List*"],
      "Resource": "*"
    }
  ]
}
```

The Policy Language is about Authorization

- Two facets:
  - **Specification**: *defining* access policies
  - **Enforcement**: *evaluating* policies

Specification

Policies

- JSON-formatted documents
- Contain statements (permissions) which specify:
  - What actions a principal can perform
  - Which resources can be accessed

S3 Read-Only Access

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:Get*", "s3:List*"],
      "Resource": "*"
    }
  ]
}
```

Example of an IAM user/group/role access policy

Anatomy of a statement

```
{
 "Statement":[{
   "Effect":"effect",
   "Principal":"principal",
   "Action":"action",
   "Resource":"arn",
   "Condition":{
     "condition":{
       "key":"value" }
     }
    }
   ]
}
```

Principal
Action
Resource
Conditions

Conditions on request-time metadata
- IP Address
- UserAgent
- date/time

```
Effect: Allow
Principal:123456789012:user/bob
Action: s3:*
Resource: jeff_bucket/*
Condition: Referer = example.com
```

```
Effect: Deny
Principal:123456789012:user/jim
Action: s3:DeleteBucket
Resource: jeff_bucket
Condition: Referer = example.com
```

Principal - Examples

- An entity that is allowed or denied access to a resource
- Principal element required for resource-based policies

```
<!-- Everyone (anonymous users) -->
"Principal":"AWS":"*.*"

<!-- Specific account or accounts -->
"Principal":{"AWS":"arn:aws:iam::account-number-without-hyphens:root" }
"Principal":{"AWS":"account-number-without-hyphens"}

<!-- Individual IAM user -->
"Principal":"AWS":"arn:aws:iam::account-number-without-hyphens:user/username"

<!-- Federated user (using web identity federation) -->
"Principal":{"Federated":"www.amazon.com"}
"Principal":{"Federated":"graph.facebook.com"}
"Principal":{"Federated":"accounts.google.com"}

<!-- Specific role -->
"Principal":{"AWS":["arn:aws:iam::account-number-without-hyphens:role/rolename"]}

<!-- Specific service -->
 "Principal":{"Service":["ec2.amazonaws.com"]}
```

Action - Examples

- Describes the type of access that should be allowed or denied
- Statements must include either an Action or NotAction element

```
<!-- EC2 action -->
"Action":"ec2:StartInstances"

<!-- IAM action -->
"Action":"iam:ChangePassword"

<!-- S3 action -->
"Action":"s3:GetObject“

<!-- Specify multiple values for the Action element-->
"Action":["sqs:SendMessage","sqs:ReceiveMessage"]

<--Use wildcards (* or ?) as part of the action name.  This would cover Create/Delete/List/Update-->
"Action":"iam:*AccessKey*"
```

Understanding NotAction

- Lets you specify an exception to a list of actions
- Can sometimes result in shorter policies than using Action and denying many actions
- Example: Let's say you want to allow everything but IAM APIs

```
{
  "Version": "2012-10-17",
  "Statement": [ {
      "Effect": "Allow",
      "NotAction": "iam:*",
      "Resource": "*"
    }
  ]
}
```

This is **not a Deny**. A user could still have a separate policy that grants IAM:*

Notice the difference?

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": "iam:*",
      "Resource": "*"
    }
  ]
}
```

If you want to prevent the user from **ever** being able to call IAM APIs use an explicit deny

Resource - Examples

- The object or objects that are being requested
- Statements must include either a Resource or a NotResource element

```
<-- S3 Bucket -->
"Resource":"arn:aws:s3:::my_corporate_bucket/*“

<-- SQS queue-->
"Resource":"arn:aws:sqs:us-west-2:account-number-without-hyphens:queue1"

<-- IAM user -->
"Resource":"arn:aws:iam::account-number-without-hyphens:user/Bob"

<-- Multiple DynamoDB tables -->
"Resource":["arn:aws:dynamodb:us-west-2:account-number-without-hyphens:table/books_table",
            "arn:aws:dynamodb:us-west-2:account-number-without-hyphens:table/magazines_table"]

<-- All EC2 instances for an account in a region -->
 "Resource": "arn:aws:ec2:us-east-1:account-number-without-hyphens:instance/*"
```

Resource-Based Policies vs. IAM Policies

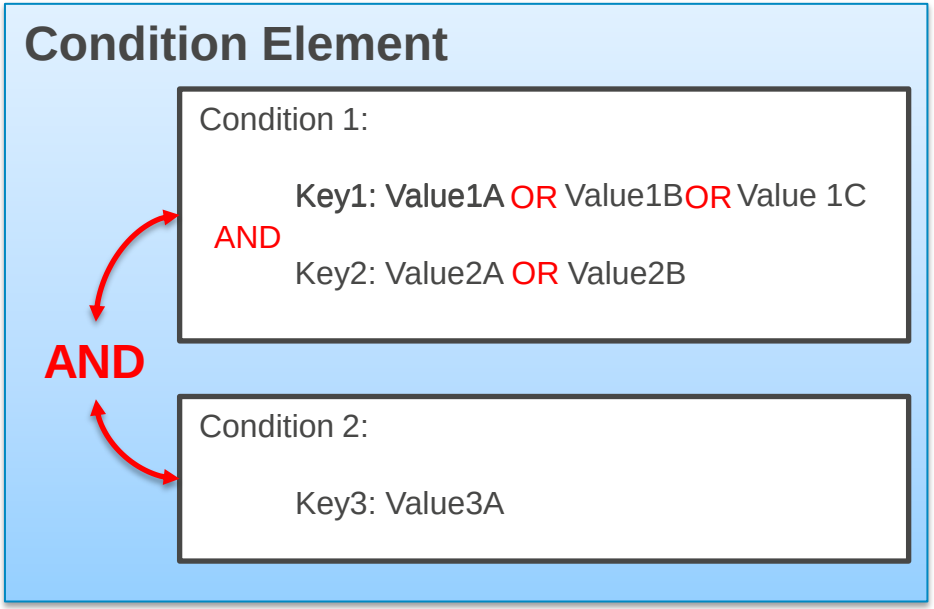
- IAM policies live with
  - IAM Users
  - IAM Groups
  - IAM Roles
- Some services allow storing policy with resources
  - S3 (bucket policy)
  - SNS (topic policy)
  - SQS (queue policy)

Principal required here

```
{
 "Statement":
  {
    "Sid":"Queue1_SendMessage",
    "Effect": "Allow",
    "Principal": {"AWS": "111122223333"},
    "Action": "sqs:SendMessage",
    "Resource":
       "arn:aws:sqs:us-east-1:444455556666:queue1"
  }
}
```

Conditions

- Conditions are optional
- Condition element can contain multiple conditions
- Condition keys can contains multiple values
- If a single condition includes multiple values for one key, the condition is evaluated using logical OR
- multiple conditions (or multiple keys in a single condition) the conditions are evaluated using logical AND



Condition Example

```
"Condition" :  {
   "DateGreaterThan" : {"aws:CurrentTime" : "2013-08-16T12:00:00Z"},
   "DateLessThan": {"aws:CurrentTime" : "2013-08-16T15:00:00Z"},
   "IpAddress" : {"aws:SourceIp" : ["192.0.2.0/24", "203.0.113.0/24"]}
}
```

AND

OR

Allows a user to access a resource under the following conditions:

- The time is after 12:00 p.m. on 8/16/2013
- The time is before 3:00 p.m. on 8/16/2013
- The request comes from an IP address in the 192.0.2.0 /24 or 203.0.113.0 /24 range

Policy Variables

Policy Variables

- Example use cases
  - Allows users to self-manage their own credentials
  - Easily set up user access to “home folder” in S3
  - Manage EC2 resources using tags
- Benefits
  - Reduces the need for user specific policies
  - Simplifies overall management
- Variables based on request context
  - Existing keys (aws:SourceIP, DateTime, etc.)
  - New keys (aws:username, aws:userid, aws:principaltype, others)
  - Provider-specific keys (graph.facebook.com:id, www.amazon.com:user_id)

The Anatomy of a Policy with Variables

```
{
  "Version": "2012-10-17",
  "Statement": [{
     "Effect": "Allow",
     "Action": ["s3:ListBucket"],
     "Resource": ["arn:aws:s3:::myBucket"],
     "Condition":
       {"StringLike":
          {"s3:prefix":["home/${aws:userid}/*"]}
        }
    },
    {
      "Effect":"Allow",
      "Action":["s3:*"],
      "Resource": ["arn:aws:s3:::myBucket/home/${aws:userid}",
                   "arn:aws:s3:::myBucket/home/${aws:userid}/*"]
    }
  ]
}
```

New Version is required

Variable in conditions

Variable in resource ARNs

Grants a user a home directory in S3 that can be accessed programmatically

Creating an S3 Home Directory

Demo

Giving a User a Home Directory From S3 Console

```
{
"Version": "2012-10-17",
 "Statement": [
   {"Sid": "AllowGroupToSeeBucketListInTheManagementConsole",
     "Action": ["s3:ListAllMyBuckets", "s3:GetBucketLocation"],
     "Effect": "Allow",
     "Resource": ["arn:aws:s3:::*"]},
   {"Sid": "AllowRootLevelListingOfThisBucketAndHomePrefix",
     "Action": ["s3:ListBucket"],
     "Effect": "Allow",
     "Resource": ["arn:aws:s3:::myBucket"],
     "Condition":{"StringEquals":{"s3:prefix":["","home/"],"s3:delimiter":["/"]}}},
   {"Sid": "AllowListBucketofASpecificUserPrefix",
     "Action": ["s3:ListBucket"],
     "Effect": "Allow",
     "Resource": ["arn:aws:s3:::myBucket"],
     "Condition":{"StringLike":{"s3:prefix":["home/${aws:username}/*"]}}},
   {"Sid":"AllowUserFullAccesstoJustSpecificUserPrefix",
      "Action":["s3:*"],
      "Effect":"Allow",
      "Resource": ["arn:aws:s3:::myBucket/home/${aws:username}",
                   "arn:aws:s3:::myBucket/home/${aws:username}/*"]}
 ]
}
```

Necessary to access the S3 console

Allows listing all objects in a folder + its subfolders

Allows modifying objects in the folder + subfolders

Allowing an IAM User to Self-manage Secrets

Demo

Grant a User Access to the IAM Console

```
{
  "Version": "2012-10-17",
  "Statement": [{
     "Sid": "ViewListOfAllUsers",
     "Action": ["iam:ListUsers"],
     "Effect": "Allow",
     "Resource": ["arn:aws:iam::123456789012:user/*"]
    },
    {
     "Sid": "AllowUserToSeeListOfOwnStuff",
     "Action": ["iam:GetUser","iam:GetLoginProfile",
         "iam:ListGroupsForUser","iam:ListAccessKeys"],
     "Effect": "Allow",
     "Resource": ["arn:aws:iam::123456789012:user/${aws:username}"]
    }
  ]
}
```

- Underneath the covers the IAM console calls these APIs
- Keep in mind the user will be able to view *limited* details about all users
- The IAM user will **not** be able to modify the other IAM users settings
- Alternatively, use the CLI

Allow IAM User to “Self-manage” from Console

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Action": ["iam:*AccessKey*","iam:*SigningCertificate*"],
   "Effect": "Allow",
   "Resource": ["arn:aws:iam::123456789012:user/${aws:username}"]
  }
 ]
}
```

Edit these actions if you want to modify user permissions

Allowing an IAM user to self-manage vMFA

Demo

Allow User to Manage Own Virtual MFA from IAM Console

```
{
  "Version": "2012-10-17",
  "Statement": [
   {
      "Action": ["iam:CreateVirtualMFADevice","iam:DeleteVirtualMFADevice"],
      "Effect": "Allow",
      "Resource": "arn:aws:iam::123456789012:mfa/${aws:username}"
   },
   {
      "Action": ["iam:DeactivateMFADevice",
                 "iam:EnableMFADevice",
                 "iam:ListMFADevices",
                 "iam:ResyncMFADevice"],
      "Effect": "Allow",
      "Resource": "arn:aws:iam::123456789012:user/${aws:username}"
   },
   {
      "Action": ["iam:ListVirtualMFADevices"],
      "Effect": "Allow",
      "Resource": "arn:aws:iam::123456789012:mfa/*"
   }
 ]
}
```

Amazon EC2 Resource Permissions

What Changes with EC2 Permissions

- Previously policies applied to all EC2 resources
- Permissions can now be set per-resource
- Ex: assign which users can stop, start, or terminate a particular instance

EC2 Policies Before Resource Permissions

```
{
   "Statement": [{
      "Effect": "Allow",
      "Action": ["ec2:TerminateInstances"],
      "Resource":"*"
   }
  ]
}
```

EC2 Policies After Resource Permissions

```
{
   "Statement": [{
      "Effect": "Allow",
      "Action": ["ec2:TerminateInstances"],
      "Resource":"*",
      "Condition": {
        "StringEquals": {"ec2:ResourceTag/department": "dev"}
      }
    }
  ]
}
```

EC2 Policies After Resource Permissions

```
{
   "Statement": [{
      "Effect": "Allow",
      "Action": ["ec2:TerminateInstances"],
      "Resource":
        "arn:aws:ec2:us-east-1:123456789012:instance/*",
      "Condition": {
        "StringEquals": {"ec2:ResourceTag/department": "dev"}
      }
    }
  ]
}
```

EC2 Policies After Resource Permissions

```
{
   "Statement": [{
      "Effect": "Allow",
      "Action": ["ec2:TerminateInstances"],
      "Resource":
        "arn:aws:ec2:us-east-1:123456789012:instance/i-abc12345"
     }
  ]
}
```

Supported Resource Types

Supports many different resource types, including:

- Customer gateway
- DHCP options set
- Image
- Instance
- Instance profile
- Internet gateway
- Key pair
- Network ACL
- Network interface
- Placement group
- Route table
- Security group
- Snapshot
- Subnet
- Volume
- VPC

APIs Currently Supported

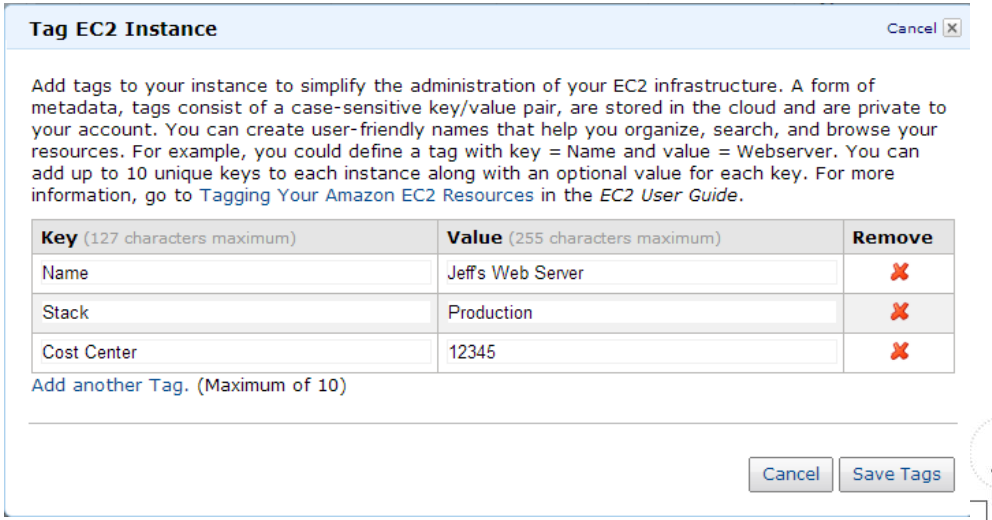
Type of Resource	Actions
EC2 Instances	StartInstances, StopInstances, RebootInstances, TerminateInstances, RunInstance[1]
Customer gateway	DeleteCustomerGateway
DHCP Options Sets	DeleteDhcpOptions
Internet Gateways	DeleteInternetGateway
Network ACLs	DeleteNetworkAcl, DeleteNetworkAclEntry
Route Tables	DeleteRoute, DeleteRouteTable
Security Groups	AuthorizeSecurityGroupEgress, AuthorizeSecurityGroupIngress, DeleteSecurityGroup, RevokeSecurityGroupEgress, RevokeSecurityGroupIngress
Volumes	AttachVolume, DeleteVolume, DetachVolume

[1]Coming Soon

Accurate as of 11/13/2013

Categorize Your Resources

- Use tags as a resource attribute
  - Allows user-defined models
  - “Prod”/”Dev”
  - “Cost Center X”
  - “Department Y”



Using Amazon EC2 resource-level permissions

Demo

Locking Down Access to EC2 Instances

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
    "Sid": "THISALLOWSEC2READACCESS",
    "Effect": "Allow",
    "Action": ["ec2:Describe*","elasticloadbalancing:Describe*",
               "cloudwatch:ListMetrics","cloudwatch:GetMetricStatistics",
              "cloudwatch:Describe*","autoscaling:Describe*"],
     "Resource": "*"
     },
     {
    "Sid": "THISLIMITSACCESSTOOWNINSTANCES",
    "Effect": "Allow",
    "Action": ["ec2:RebootInstances","ec2:StartInstances",
              "ec2:StopInstances","ec2:TerminateInstances"],
    "Resource":"arn:aws:ec2:us-east-1:123456789012:instance/*",
    "Condition": {"StringEquals":
                  {"ec2:ResourceTag/Owner": "${aws:username}"}}
      }
    ]
}
```

New Version is required here because we're using variables

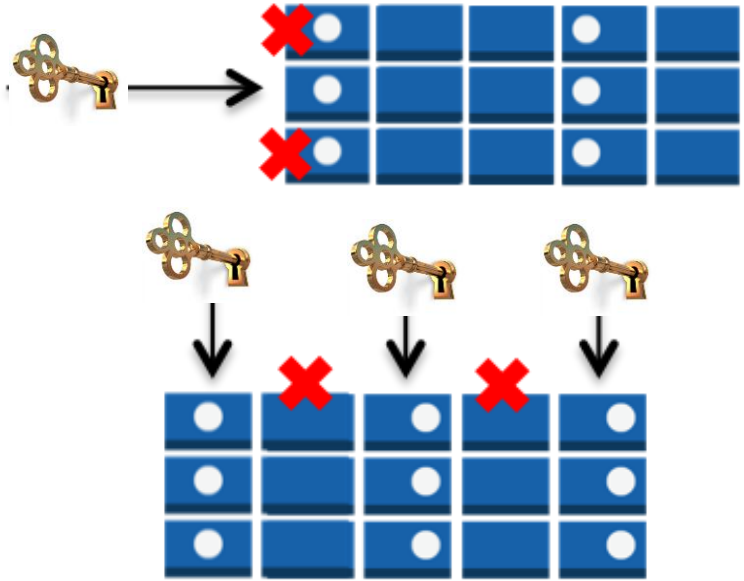
Allows seeing everything from the EC2 console.

Allowed only if this tag condition is true

Use variables for the owner tag

Amazon DynamoDB
Fine-Grained Access Control New

Enables Sub-table and Per-action Access Control



Horizontal or vertical access control

```
GetItem
BatchGetItem
Query
```

```
GetItem
BatchGetItem
Query
PutItem
UpdateItem
BatchWriteItem
```

Read-only or read-write access

DynamoDB Fine-Grained Access Control

- Grant or deny access to individual items by hiding tables or index information
  - Horizontally by matching primary key values
  - Vertically by controlling which attributes are visible
- Use policy conditions to define level of access
  - `dynamodb:LeadingKeys` – access items where the hash key value matches a unique identifier (ex: aws:userid policy variable)
  - `dynamodb:Attributes` – allows access to only a subset of attributes
  - `StringEqualsIfExists` clause – ensures the app must always provide a list of attributes to act opon
- You must include all primary and index key attributes if you use `dynamodb:Attributes`

Configuring Fine-Grained Access Control

Demo

Example: Restricting Access to a Table

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem", "dynamodb:BatchGetItem","dynamodb:Query",
        "dynamodb:PutItem", "dynamodb:UpdateItem", "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"],
      "Resource": ["arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"],
      "Condition": {
          "ForAllValues:StringEquals": {
             "dynamodb:LeadingKeys":  ["${www.amazon.com:user_id}"],
             "dynamodb:Attributes": [
                 "UserId","GameTitle","Wins","Losses",
                 "TopScore","TopScoreDateTime"]
          },
          "StringEqualsIfExists": {"dynamodb:Select": "SPECIFIC_ATTRIBUTES"}
        }
      }
    ]
}
```

New Version is required

Hash key value must match the user's ID. Results will be horizontally filtered.

Only return these attributes. Results will be vertically filtered.

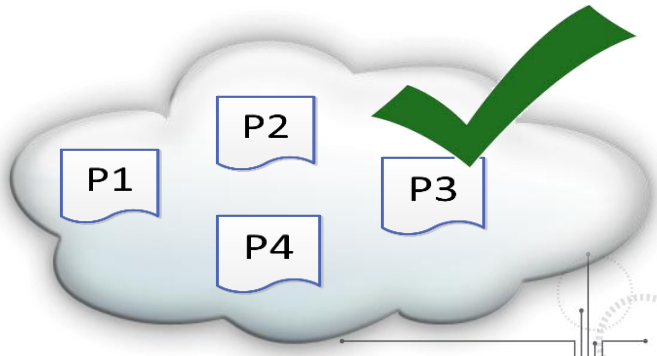
App must specify attributes. Cannot request all.

Note that `Scan` is not included, because `Scan` would provide access to **all** of the leading keys

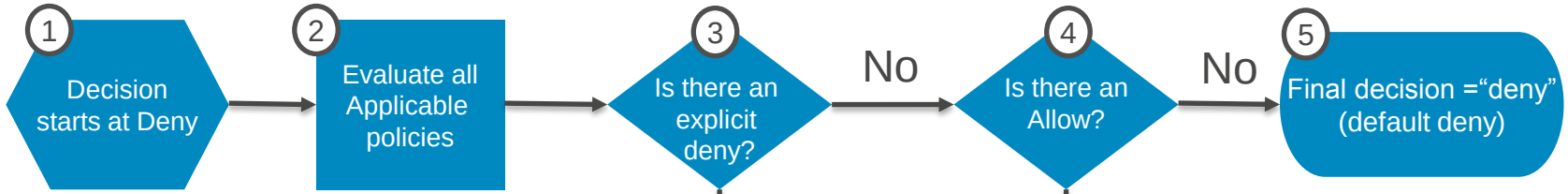
Let’s Finish Up with Enforcement

Policy Enforcement

- Remember policies can come from multiple places
  - IAM users, roles, and groups
  - AWS resources (S3, SQS, & SNS)
  - Passed through federated users
- Well-defined evaluation logic
  - A request can be allowed or denied
  - “Deny” trumps “Allow”
  - If not allowed, request is denied by default
  - Permissions are union of all policies



Determining if a Request is Allowed or Denied



- AWS retrieves all policies associated with the user and resource
- Only policies that match the action & conditions are evaluated

- If a policy statement has a deny, it trumps all other policies

- Access is granted if there is an explicit allow and no deny

- By default, a implicit (default) deny is returned

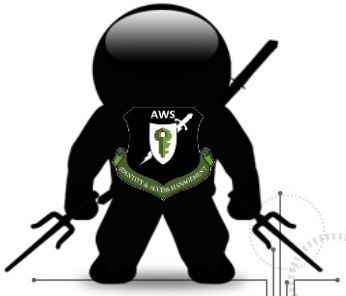
Testing Policies Using the Policy Simulator

Demo

https://policysim.aws.amazon.com

Summary

- IAM provides access control for your AWS account
- Use the policy language to allow or deny granular access to AWS resources
  - Users are denied access by default
  - Denys trump allow
- All policies (user, group, resource-based) are evaluated for authorization
- Use policy variables - they make life better!
  - Simplifies policy management
  - Reduces the need for individual user policies
- We're continuously enabling more granular control
  - EC2 / RDS Resource-level permissions
  - DynamoDB fine-grained access control



Additional Resources

- IAM detail page: http://aws.amazon.com/iam
- AWS forum: https://forums.aws.amazon.com/forum.jspa?forumID=76
- Documentation: http://aws.amazon.com/documentation/iam/
- AWS Security Blog: http://blogs.aws.amazon.com/security
- Twitter: @AWSIdentity

All IAM-Related Sessions at re:Invent

ID	Title	Time, Room
CPN205	Securing Your Amazon EC2 Environment with AWS IAM Roles and Resource-Based Permissions	Wed 11/13 11am, Delfino 4003
SEC201	Access Control for the Cloud: AWS Identity and Access Management (IAM)	Wed 11/13 1.30pm, Marcello 4406
SEC301	TOP 10 IAM Best Practices	Wed 11/13 3pm, Marcello 4503
SEC302	Mastering Access Control Policies	Wed 11/13 4.15pm, Venetian A
SEC303	Delegating Access to Your AWS Environment	Thu 11/14 11am, Venetian A

Come talk security with AWS	**Thu 11/14 4pm, Toscana 3605**

AWS re:Invent

Please give us your feedback on this presentation

SEC302

As a thank you, we will select prize winners daily for completed surveys!

amazon web services