# Week 3, Lecture 1

## COMP6[48]43

Hamish Cox

Mar 4th, 2025

# Week 2 Recap

Last week we looked at authenticating users…

- initially, using passwords, MFA and/or SSO,
- and for subsequent requests, using a token stored in a cookie.

But while we now know who a user is, we also need to check that they are allowed to access/do whatever they are trying to.

# Let’s review what we left off with last week

We had some password authentication.

Let’s add a page to view a list of users.

This should only be visible to ‘admins’.

# What does a permissions check look like?

```
if some_user_or_session_property ≠ expected_value:
    return abort(403)
```

# What does a permissions check look like?

```
if session.get("role") ≠ "admin":
    return abort(403)
```

# WYCTOAC

(Whatever-You-Can-Think-Of Access Control)

There's a lot of acronyms for various different access control models:

- ACLs (Access Control Lists)
- RBAC (Role Based Access Control)
- ABAC (Attribute Based Access Control)
- RuBAC (Rule Based Access Control)
- DAC, MAC, ReBAC...

At the end of the day these definitions really don't matter. They get mixed and matched all the time.

# Example: AWS

## It's Role-Based:

**EdDisco**[redacted]

Delete

**Summary**

Edit

**Creation date**
September 12, 2021, 23:12 (UTC+10:00)

**Last activity**
8 minutes ago

**ARN**
arn:aws:iam:[redacted]role/EdDisco[redacted]

**Maximum session duration**
1 hour

Permissions | Trust relationships | Tags | Last Accessed | Revoke sessions

**Permissions policies** (2) Info

You can attach up to 10 managed policies.

Simulate | Remove | Add permissions

Search

Filter by Type: All types

1

| Policy name | Type | Attached entities |
|---|---|---|
| AWSLambdaBasicExecutionRole | AWS managed | 1 |
| UpdateFnServiceRoleDefaultPolicy39CF5F23 | Customer inline | 0 |

# Example: AWS

Wait, but the permissions on those roles can have rules:

| | | |
|---|---|---|
| UpdateFnServiceRoleDefaultPolicy39CF5F23 | Customer inline | 0 |

**UpdateFnServiceRoleDefaultPolicy39CF5F23**

Copy JSON  Edit

```
1  {
2      "Version": "2012-10-17",
3      "Statement": [
4          {
5              "Action": "secretsmanager:GetSecretValue",
6              "Resource": "arn:aws:secretsmanager:ap-southeast-2[redacted]secret:ed-discor[redacted]",
7              "Effect": "Allow"
8          },
9          {
10             "Action": [
11                 "dynamodb:PutItem",
12                 "dynamodb:Scan"
13             ],
14             "Resource": "arn:aws:dynamodb:ap-southeast-2[redacted]table/EdDisco[redacted]",
15             "Effect": "Allow"
16         }
17     ]
18 }
```

# Example: AWS

Possibly quite complicated ones:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "StartStopIfTags",
            "Effect": "Allow",
            "Action": [
                "ec2:StartInstances",
                "ec2:StopInstances",
                "ec2:DescribeTags"
            ],
            "Resource": "arn:aws:ec2:region:account-id:instance/*",
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/Project": "DataAnalytics",
                    "aws:PrincipalTag/Department": "Data"
                }
            }
        }
    ]
}
```

Allow starting or stopping instances and describing the tags

All instances in the specified region and account id only

Only those instances with tag Project : DataAnalytics

Only by the Principals with tag Department : Data

# Example: AWS

## And you can attach them directly to users:

**HamishCox** Info

Delete

**Summary**

ARN
arn:aws:iam:: [redacted] user/HamishCox

Created
May 22, 2020, 22:04 (UTC+10:00)

Console access
Enabled with MFA

Last console sign-in
Today

Access key 1
[redacted] Active
Used 164 days ago. 826 days old.

Access key 2
Create access key

Permissions | Groups (2) | Tags | Security credentials | Last Accessed

**Permissions policies** (2)

Permissions are defined by policies attached to the user directly or through groups.

Remove | Add permissions

Search

Filter by Type: All types

1

| Policy name | Type | Attached via |
|---|---|---|
| AdministratorAccess | AWS managed - job function | Group Administrator |
| Billing | AWS managed - job function | Group BillingAccess |

# Example: AWS

## Or to certain kinds of resources:

**Bucket policy**

Edit Delete

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. Learn more

**Public access is blocked because Block Public Access settings are turned on for this bucket**
To determine which settings are turned on, check your Block Public Access settings for this bucket. Learn more about using Amazon S3 Block Public Access

Copy

```
{
  "Version": "2012-10-17",
  "Id": "AccessControl",
  "Statement": [
    {
      "Sid": "AllowSSLRequestsOnly",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::cdk-hnb659fds-assets[redacted]ap-southeast-2",
        "arn:aws:s3:::cdk-hnb659fds-assets[redacted]ap-southeast-2/*"
      ],
      "Condition": {
        "Bool": {
          "aws:SecureTransport": "false"
        }
      }
    }
  ]
```

# Example: AWS

And then those resources might then have their own access control models:

## Access control list (ACL)

Edit

Grant basic read/write permissions to other AWS accounts. Learn more

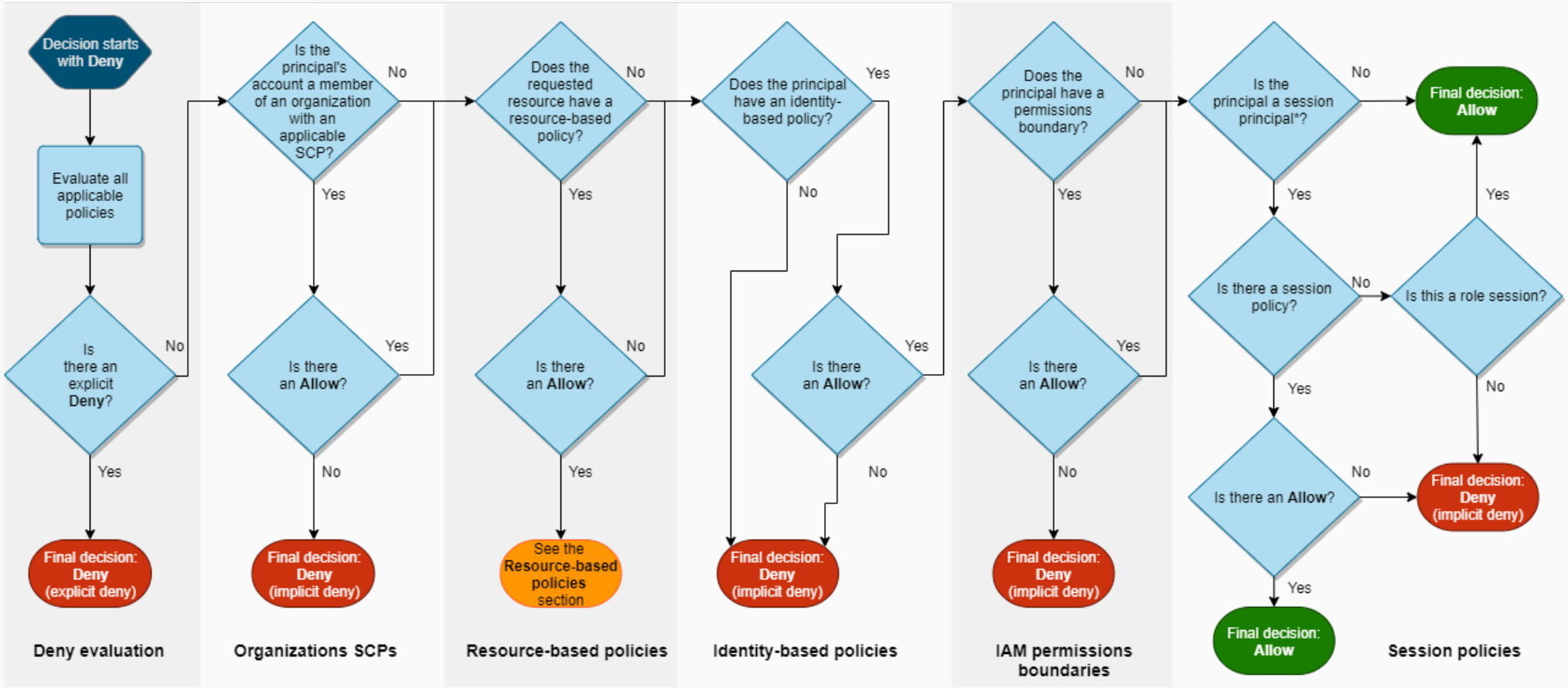
**Public access is blocked because Block Public Access settings are turned on for this bucket**
To determine which settings are turned on, check your Block Public Access settings for this bucket. Learn more about using Amazon S3 Block Public Access

**The console displays combined access grants for duplicate grantees**
To see the full list of ACLs, use the Amazon S3 REST API, AWS CLI, or AWS SDKs.

| Grantee | Objects | Bucket ACL |
|---|---|---|
| **Bucket owner (your AWS account)**<br>Canonical ID: 71a0d85fba411b49f965[redacted] | List, Write | Read, Write |
| **Everyone (public access)**<br>Group: http://acs.amazonaws.com/groups/global/AllUsers | - | - |
| **Authenticated users group (anyone with an AWS account)**<br>Group: http://acs.amazonaws.com/groups/global/AuthenticatedUsers | - | - |
| **S3 log delivery group**<br>Group: http://acs.amazonaws.com/groups/s3/LogDelivery | - | - |

# Example: AWS

If in doubt, here's a handy simple guide:



# This seems complicated and easy to mess up

it is :)

AWS is a special brand of insane that I bring up because I've had to deal with it recently, but any system with enough users will likely gravitate towards a more complex system that is easy to mess up.

# Where does this mess up occur?

It's typically either:

- the application developer (e.g. a bug in the permission check code),
- or the administrator setting up their permissions (e.g. selecting something wrong in the admin panel).

# Vulnerabilities

These basically boil down to:

- misconfigured permissions,
- storing authorization information (e.g. a role) insecurely on the client-side,
- missing permission/authorization checks.

In particular: In-Direct Object Reference (IDOR)

# Delegating Authorization

Remember Single Sign On?

What if we (as an application developer) wanted to access the provider's resource(s) on our user's behalf?