🔷 How does S3 handle permissions and security?

- **Identity-Based Access**:
  ➔ AWS **IAM policies** control **who** (users, roles) can access **what** in S3.
- **Resource-Based Access**:
  ➔ **Bucket policies** and **Access Control Lists (ACLs)** control **who** can access **specific buckets or objects**.
- **Authentication and Authorization**:
  ➔ Every request to S3 is authenticated (signed with credentials) and authorized (allowed or denied based on policies).
- **Encryption**:
  ➔ You can encrypt data at rest and in transit.
- **Versioning, Logging, and MFA (Multi-Factor Authentication) Deletes**:
  ➔ Extra features to protect data from accidental or malicious deletion.

🔷 What is an S3 Bucket Policy? How is it different from an IAM Policy?

Aspect	Bucket Policy	IAM Policy
Attached To	Specific **S3 bucket** (resource-based)	Specific **IAM user/role/group** (identity-based)
Scope	Controls access **to that bucket** and its objects	Controls access **of the user/role** to AWS resources
Who it Affects	Any user (even outside your AWS account) if allowed	Only AWS users and roles inside your account
Example	"Allow anyone to download files from this	"Allow this user to upload files to any

Aspect	Bucket Policy	IAM Policy
	bucket"	bucket"

Summary:

- **Bucket Policy**: Attached to **the bucket**.
- **IAM Policy**: Attached to **the user/role**.

#	Use Case Title	Policy Type	Description	Minimal JSON / Steps
1	**Public Read-Only Access to Static Website**	Bucket Policy	Allows public to read website files in the bucket.	`"Principal": "*", "Action": "s3:GetObject"`
2	**Allow Specific IP Range for Uploads Only**	Bucket Policy	Allow uploads only from specific IPs.	`"Condition": { "IpAddress": { "aws:SourceIp": "203.0.113.0/24" }}`
3	**Cross-Account Access for Logs Storage**	Bucket Policy	Another AWS account can write logs.	`"Principal": { "AWS": "arn:aws:iam::123456789012:root" }`
4	**Deny Deletes from Non-MFA Authenticated Users**	Bucket Policy	Prevent deletes unless MFA is used.	`"Effect": "Deny", "Action": "s3:DeleteObject", "Condition": { "Bool": { "aws:MultiFactorAuthPresent": "false" }}`
5	**Allow CloudFront Access via OAI**	Bucket Policy	Restrict read to CloudFront OAI only.	`"AWS": "arn:aws:iam::cloudfront:user/CloudFront Origin Access Identity E1234567"`

#	Use Case Title	Policy Type	Description	Minimal JSON / Steps
6	**Developer Upload Access to Dev Bucket**	IAM Policy	Devs can upload to `/dev/` prefix only.	`"Resource": "arn:aws:s3:::my-bucket/dev/*", "Action": "s3:PutObject"`
7	**Read-Only Access to All S3 Buckets**	IAM Policy	Auditors get list/read access.	`"Action": ["s3:ListBucket", "s3:GetObject"], "Resource": "*"`
8	**Time-Limited Access via Conditions**	IAM Policy	Temporary access within date range.	`"Condition": { "DateGreaterThan": { "aws:CurrentTime": "2025-05-01T00:00:00Z" }, "DateLessThan": { "aws:CurrentTime": "2025-05-31T23:59:59Z" }}`
9	**Restrict Access to Certain Buckets by Tag**	IAM Policy	Access only if bucket has tag `Environment=Production`.	`"Condition": { "StringEquals": { "s3:ResourceTag/Environment": "Production" }}`
10	**Allow Backup**	IAM	Full access for	`"Resource":`

#	Use Case Title	Policy Type	Description	Minimal JSON / Steps
	Tool Full Access to Backup Bucket	Policy	automated backups.	`"arn:aws:s3:::backup-bucket/*", "Action": "s3:*"`

Let's take **one practical demo example** to **explain everything you asked**, including:

- Identity-based access
- Resource-based access
- Authentication
- Encryption
- Versioning, Logging, MFA delete
- Bucket Policy vs IAM Policy

I'll explain as if you are **doing it on AWS Console step-by-step**, so it's easy to imagine.

Scenario:

You have an S3 bucket called `company-documents`.
You want:

- Only a specific IAM User (`s3-user`) to upload documents.
- Allow **everyone on the internet** to **download** a document (`public-read`).
- Ensure documents are encrypted at rest.
- Enable versioning and MFA delete for extra safety.

🔷 Step 1: Identity-Based Access (IAM Policy)

✅ Create an IAM user called `s3-user`.

✅ Attach this IAM Policy to allow upload to `company-documents`:

```
json
---------------
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
```

```
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::company-documents/*"
    }
  ]
}
```

- **Meaning**:
  ➔ `s3-user` can upload files (`PutObject`) and set ACLs for objects.
  ➔ Only inside the `company-documents` bucket.

✅ This is **Identity-Based Access** because **policy is attached to user**.

🔷 Step 2: Resource-Based Access (Bucket Policy)

✅ In the S3 console, **go to** `company-documents` → **Permissions** → **Bucket Policy**.

✅ Add this **Bucket Policy** to allow **anyone** to download (GET) objects:

```
json
---------------
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowPublicRead",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::company-documents/*"
    }
  ]
}
```

- **Meaning**:
  ➔ Any user (`Principal: "*"`) can download files (`GetObject`) from the bucket.

✅ This is **Resource-Based Access** because **policy is attached to the bucket**.

🔷 Step 3: Authentication and Authorization

✅ Now when someone uploads a file:

- AWS checks IAM credentials (access key + secret key) of `s3-user`.
- Validates the IAM policy ➔ Upload is **allowed** ✅.

✅ When someone downloads a file:

- No authentication needed (because public access is allowed in bucket policy).

✅ Behind the scenes:

- Every request (upload/download) is either **authenticated** or **checked against permissions**.

🔷 Step 4: Encryption (at Rest and in Transit)

✅ **Encryption at Rest**:

- In S3 bucket settings ➔ Enable **Default Encryption** ➔ Choose **SSE-S3** (AWS manages keys).

Now, every file uploaded will be automatically encrypted at rest.

✅ **Encryption in Transit**:

- Always use HTTPS URLs for uploads/downloads (default if using AWS Console, SDK, CLI).

🔷 Step 5: Enable Versioning, Logging, MFA-Delete

✅ **Versioning**:

- In the bucket properties ➔ Enable **Versioning**.

✅ Now if someone uploads a file with the same name, S3 keeps **both versions**!

✅ **MFA-Delete**:

- You can enable MFA Delete (extra setting inside Versioning).
- Means: **To delete an object**, user must provide an MFA token (OTP).

✅ **Logging**:

- Enable **Server Access Logging** → Logs will be written to another S3 bucket.

📚 Final Mapping:

Feature	Where it applies	Demo Step
IAM Policy (Identity Based)	User (s3-user)	Step 1
Bucket Policy (Resource Based)	Bucket (company-documents)	Step 2
Authentication	Request signed with IAM credentials	Step 3
Authorization	IAM + Bucket Policy evaluation	Step 3
Encryption at Rest	SSE-S3	Step 4
Encryption in Transit	HTTPS	Step 4

Feature	Where it applies	Demo Step
Versioning	Multiple versions of objects	Step 5
MFA Delete	Require MFA for deletes	Step 5
Logging	Server access logs	Step 5

In One Line:

- **IAM Policy** ➔ Controls who can upload.
- **Bucket Policy** ➔ Controls who can download.
- **Authentication** ➔ All upload requests are signed and checked.
- **Encryption** ➔ Data is encrypted both while moving (in transit) and staying (at rest).
- **Versioning + MFA-Delete** ➔ Protects against accidental/malicious deletion.

Scenario (Same as Before):

Bucket: `company-documents`
User: `s3-user`
Goal: Upload (private user), Download (public), Encryption, Versioning.

Step-by-Step CLI Demo

Step 1: Create a New S3 Bucket

```
bash
---------------
aws s3api create-bucket --bucket company-documents --region us-east-1
```

✅ This creates a new bucket called `company-documents`.

Step 2: Enable Default Encryption (SSE-S3)

```
bash
---------------
aws s3api put-bucket-encryption --bucket company-documents --server-side-
encryption-configuration '
{
    "Rules": [
        {
            "ApplyServerSideEncryptionByDefault": {
                "SSEAlgorithm": "AES256"
            }
```

```
        }
    ]
}'
```

✅ Now **all files** uploaded are **encrypted at rest** automatically using AES256.

Step 3: Enable Versioning

```
bash
---------------
aws s3api put-bucket-versioning --bucket company-documents --versioning-
configuration Status=Enabled
```

✅ Now **every file version is saved** even if updated multiple times.

Step 4: Add Public Read Bucket Policy

```
bash
---------------
aws s3api put-bucket-policy --bucket company-documents --policy '
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::company-documents/*"
    }
  ]
}'
```

✅ Anyone can **download** (`GetObject`) files publicly.

Step 5: Upload a Test File (Private Upload)

✅ Create a small test file:

```
bash
---------------
echo "Hello, AWS S3!" > testfile.txt
```

✅ Upload the file to S3:

```
bash
---------------
aws s3 cp testfile.txt s3://company-documents/testfile.txt
```

✅ After upload, because of **bucket policy**, the file is now **publicly downloadable** by anyone using:

```
bash
---------------
https://company-documents.s3.amazonaws.com/testfile.txt
```

✅ (You can open this URL directly in browser! 🎯)

Summary of CLI actions:

Step	Command	What it did
1	`create-bucket`	Created `company-documents` bucket
2	`put-bucket-encryption`	Enabled default server-side encryption
3	`put-bucket-versioning`	Enabled object versioning
4	`put-bucket-policy`	Made objects public-read
5	`cp` upload	Uploaded a test file

---

1. Which AWS policy allows the public to read files from a static website hosted on an S3 bucket?

A. IAM Policy with `"Action": "s3:GetObject"`
B. Bucket Policy with `"Principal": "*", "Action": "s3:GetObject"`
C. IAM Role with `"Action": "s3:ListBucket"`
D. VPC Endpoint Policy with `"Effect": "Allow"`

Correct Answer: B

---

2. What is the purpose of the following Bucket Policy snippet?

```
json
CopyEdit
"Condition": {
  "IpAddress": {
    "aws:SourceIp": "203.0.113.0/24"
  }
}
```

A. Allow uploads only from the Dev team
B. Deny access to all except the given IP
C. Allow uploads only from a specific IP range
D. Allow access only during a specific time

Correct Answer: C

---

3. Which policy type should be used to restrict developers to uploading only to the `/dev/` prefix in an S3 bucket?

A. Bucket Policy
B. SCP (Service Control Policy)
C. IAM Policy with `"Action": "s3:PutObject"` and `"Resource": "arn:aws:s3:::my-bucket/dev/*"`
D. VPC Policy

Correct Answer: C

---

4. An auditor needs to view and list all objects across all S3 buckets. What IAM policy would fulfill this?

A. `"Action": "s3:PutObject"` on all buckets
B. `"Action": ["s3:GetObject", "s3:ListBucket"], "Resource": "*"`
C. `"Action": "s3:GetBucketAcl", "Resource": "*"`
D. `"Action": "s3:DeleteObject", "Resource": "*"`

Correct Answer: B