CORS & Web Security – Interview Questions & Answers

1️⃣ What is the Same-Origin Policy, and why does it exist?

✅ **Answer:**

The **Same-Origin Policy (SOP)** is a security mechanism enforced by web browsers that prevents scripts running on one origin (protocol + domain + port) from interacting with resources on a different origin.

🔹 **Why it exists?**

- Prevents malicious websites from making unauthorized requests on behalf of a user.
- Protects sensitive user data, such as authentication tokens and session cookies.
- Helps mitigate cross-site attacks like **Cross-Site Request Forgery (CSRF)**.

2️⃣ How does CORS enable cross-origin requests?

✅ **Answer:**

CORS (Cross-Origin Resource Sharing) is a browser security feature that allows controlled access to resources from a different origin by using HTTP headers.

- When a browser makes a **cross-origin request**, the server must send **CORS headers** in its response to indicate whether the request is allowed.

The most important header is:

```
Access-Control-Allow-Origin: https://example.com
```

- This header tells the browser that `example.com` is allowed to access the resource.

3️⃣ What is a preflight request, and when is it required?

✅ **Answer:**

A **preflight request** is an **OPTIONS** request that browsers send **before** making a cross-origin request to check if the actual request is permitted.

🔹 **When is it required?**

- When the request **uses HTTP methods other than GET, HEAD, or POST** (e.g., PUT, DELETE).
- When the request **includes custom headers** (e.g., `Authorization`, `X-Custom-Header`).
- When the request **sends non-simple content types**, such as `application/json`.

🔹 **Example Flow:**

1️⃣ Browser sends an **OPTIONS** request.
2️⃣ Server responds with allowed methods and headers.
3️⃣ If approved, browser makes the actual request.

Example response from server:

```
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Methods: GET, POST, PUT
Access-Control-Allow-Headers: Authorization, Content-Type
```

4️⃣ How do you configure CORS headers on a server?

✅ **Answer:**

To configure CORS, the server must return proper **CORS headers** in its response.

🔹 **Example in Express.js (Node.js)**

```
app.use((req, res, next) => {
  res.setHeader('Access-Control-Allow-Origin', '*'); // Allow all
origins
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT,
DELETE');
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type,
Authorization');
  next();
});
```

- `Access-Control-Allow-Origin`: Specifies which domains can access the resource.

- `Access-Control-Allow-Methods`: Defines allowed HTTP methods.
- `Access-Control-Allow-Headers`: Lists allowed custom headers.

🔹 **In Nginx Configuration:**

```
add_header 'Access-Control-Allow-Origin' '*';
add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
add_header 'Access-Control-Allow-Headers' 'Authorization,
Content-Type';
```

This allows **all origins** (`*`), but it can be restricted to specific domains for better security.

5️⃣ What are common security risks associated with CORS?

✅ **Answer:**

While CORS is essential for enabling cross-origin requests, improper configuration can lead to security risks:

- **Overly permissive CORS settings (`Access-Control-Allow-Origin: *`)**
    - Allows any website to access the resource, increasing the risk of data leaks.
- **Allowing credentials without proper restrictions (`Access-Control-Allow-Credentials: true`)**
    - Can expose sensitive user sessions to unauthorized third-party sites.
- **Exposing sensitive HTTP headers (`Access-Control-Allow-Headers`)**
    - Misconfiguring this can expose private API keys, authentication tokens, or user data.

🔹 **How to mitigate risks?**

✅ Restrict origins to trusted domains.
✅ Avoid `Access-Control-Allow-Origin: *` when credentials are involved.
✅ Implement **proper authentication & authorization** mechanisms.

6️⃣ What are alternatives to CORS for handling cross-origin requests?

✅ **Answer:**

Instead of CORS, some architectures use:

1️⃣ **Reverse Proxy:**

- A backend server (e.g., Nginx) acts as an intermediary, making requests on behalf of the client.
- The client interacts only with the proxy, eliminating the need for CORS.

Example in Nginx:

```
location /api/ {
  proxy_pass http://backend-server.com/;
}
```

-

2️⃣ **JSONP (JSON with Padding) [Deprecated]**

- Used before CORS was widely supported.
- Works by dynamically loading a `<script>` tag instead of using AJAX.
- **Not secure**, as it exposes APIs to cross-site attacks.

3️⃣ **API Gateway Handling CORS**

- API Gateways (e.g., AWS API Gateway, Kong) can **manage CORS policies centrally**, ensuring secure and consistent configurations.

7️⃣ How do API Gateways and Reverse Proxies help with CORS?

✅ **Answer:**

🔹 **Reverse Proxies:**

- The **client requests the proxy**, which then forwards the request to the backend.

- Since the client and proxy are on the **same origin**, **CORS restrictions do not apply**.

Example setup in Nginx:

```
location /api/ {
  proxy_pass http://backend-service.com/;
}
```

-

🔹 **API Gateways:**

- API Gateways **control CORS policies** across multiple backend services.
- They allow **fine-grained control** over which origins, methods, and headers are permitted.
- Example in AWS API Gateway:
    - Configure **CORS settings** in API Gateway to allow specific origins.

✅ **Benefits:**
✔ Reduces **security risks** from misconfigured CORS headers.
✔ **Centralized control** over access policies.
✔ Improves **performance & request handling**.