Ataques de CORS o Cross-Origin Resource Sharing

El Cross-Origin Resource Sharing (CORS) es un mecanismo de seguridad utilizado por los navegadores web para permitir o restringir las solicitudes de recursos (como archivos JavaScript, CSS, imágenes, etc.) desde un origen (dominio, protocolo y puerto) a otro origen diferente al de la página web actualmente visualizada. Fue introducido para abordar los problemas de seguridad asociados con las solicitudes de recursos entre orígenes distintos en aplicaciones web.

Cuando un navegador realiza una solicitud de recursos a un servidor web, incluye información sobre el origen desde el cual se originó la solicitud. El servidor web, en función de esa información, puede decidir si permitir o bloquear la solicitud según la política de CORS configurada.

Gran parte de los sitios web operan bajo la política del mismo origen, conocida en inglés como **same-origin policy (SOP)**. Esta política se rige bajo una filosofía muy simple: impedir la carga de datos procedentes de otros servidores. Todos los archivos tienen que compartir la misma fuente.

Al igual que la política del mismo origen (SOP), el **CORS** restringe todas aquellas peticiones cuyo origen difiere del dominio al que se realiza la petición. Pero, al contrario que la SOP, deja abierta la puerta a la colaboración en momentos puntuales.

De este modo, se previene la **infiltración de contenido malicioso** por parte de terceros. Si no se filtrasen las peticiones según su procedencia, se le estarían abriendo las puertas de par en par a los atacantes. Y estos tendrían libertad plena para **introducir malware en las páginas**.

Aquí hay una explicación más detallada de cómo funciona el CORS:

1. **Solicitud de origen cruzado (Cross-Origin Request):** Una solicitud de origen cruzado es una solicitud HTTP realizada por un navegador desde la página web actual a un origen diferente al de la página web actual. Por ejemplo, si una página web alojada en `https://www.ejemplo.com` intenta hacer una solicitud a `https://api.ejemplo.com`, esto se consideraría una solicitud de origen cruzado.
2. **Cabeceras CORS:** Cuando se realiza una solicitud de origen cruzado, el navegador agrega cabeceras CORS a la solicitud HTTP. Estas cabeceras incluyen información sobre el origen desde el cual se originó la solicitud (`Origin`) y el tipo de solicitud que se está realizando (por ejemplo, `GET`, `POST`, `PUT`, `DELETE`, etc.).
3. **Respuesta CORS:** El servidor web al que se envía la solicitud de origen cruzado puede responder con cabeceras CORS específicas que indican si se permite o no la solicitud desde el origen dado. Estas cabeceras incluyen principalmente `Access-Control-Allow-Origin`, que especifica los orígenes permitidos para realizar solicitudes, y otras cabeceras

relacionadas con el control de acceso, como `Access-Control-Allow-Methods`, `Access-Control-Allow-Headers`, `Access-Control-Allow-Credentials`, etc.

4. **Política CORS:** La política CORS se configura en el servidor web para determinar qué solicitudes de origen cruzado son permitidas y cuáles son bloqueadas. Esto puede basarse en el origen de la solicitud, los métodos HTTP utilizados, las cabeceras enviadas con la solicitud, entre otros factores.

Ejemplos de código con CORS

Curl

```
curl -i -X OPTIONS tecnocratica.net/api/dns \
-H 'Access-Control-Request-Method: GET' \
-H 'Access-Control-Request-Headers: Content-Type, Accept' \
-H 'Origin: http://api.tecnocratica.net'
```

Apache

```
<IfModule mod_headers.c>
    <FilesMatch "\.(ttf|ttc|otf|eot|woff|font.css|css|js|gif|png|jpe?g|svg|svgz|ico|webp)$">
        Header set Access-Control-Allow-Origin "*"
    </FilesMatch>
</IfModule>
```

Nginx

```
location ~ \.(ttf|ttc|otf|eot|woff|font.css|css|js|gif|png|jpe?g|svg|svgz|ico|webp)$ {
    add_header Access-Control-Allow-Origin "*";
}
```

El CORS es esencial para proteger las aplicaciones web contra ataques de origen cruzado, como el Cross-Site Request Forgery (CSRF) y el Cross-Site Script Inclusion (XSSI). Sin embargo, es importante configurarlo correctamente para evitar problemas de seguridad, como la exposición accidental de datos sensibles. Los desarrolladores web deben comprender cómo funciona CORS y configurarlo adecuadamente en sus servidores para garantizar la seguridad de sus aplicaciones.

Revision #5
Created 4 February 2024 08:50:16 by etaboada
Updated 4 February 2024 09:29:03 by etaboada