

Vulnerabilitats web

Interfícies d'usuari

Aleix Llusà Serra

Enginyeria de Sistemes TIC
Universitat Politècnica de Catalunya
<http://epsem.upc.edu>

5 de maig de 2023



Aquesta obra està subjecta a una llicència de [Creative Commons "Reconeixement-CompartirIgual 4.0 Internacional"](https://creativecommons.org/licenses/by-sa/4.0/).

- CSRF: Cross-site request forgery
- CORS i SOP: Cross-origin resource sharing i Same Origin Policy
- XSS: Cross site scripting

- Cross-site request forgery (CSRF)
- Explota la confiança d'un lloc web amb el navegador de l'usuari
- Seguretat implementada pel servidor

- 1 Feu login al vostre banc → cookie
- 2 El banc fa transferències anant a la URL
GET `https://banc.test/transferencia?d=ES00000&q=500`
- 3 Un atacant us envia per correu o té una web amb un enllaç a
GET `https://banc.test/transferencia?d=COMPTEATACANT&q=999`
- 4 Si hi cliqueu, el navegador té la sessió desada al vostre banc i per tant s'executa la transferència.
- 5 L'enllaç serà camuflat en l'href, l'atacant hi posarà un text genèric:
`Més informació`
- 6 La URL també es pot camuflar ``, `<form action=URL>`
- 7 En formularis l'atacant pot fer servir tant GET com POST

- 1 Atacant us envia enllaç camuflat GET o POST
`https://g.com/login?u=ATACANT&PASS_ATACANT`
- 2 Si cliqueu, estareu loggejats a g.com amb les seves credencials
- 3 Es registren totes les accions que feu a g.com al seu compte: històrics de cerca?, contrasenyes emmagatzemades?, targetes emmagatzemades?

- GET i HEAD sense problemes CSRF: segons HTTP han de ser segurs; no poden canviar l'estat a l'aplicació
- POST:
 - dades application/x-www-form-urlencoded i multipart/form-data
PERILL CSRF!: cal aplicar mesures anti CSRF
 - dades enviades amb format json o XML, sense problemes CSRF: només* (***normalment**) es poden cridar des d'Ajax, el qual està protegit per SOP i CORS.
- Altres (PUT, DELETE...) sense problemes CSRF: només* (***normalment**) es poden cridar de d'Ajax, el qual està protegit per SOP i CORS.
- A més: Cal sessió de login establerta i cal que la URL tingui efectes directes en l'estat. Si POST `https://banc.test/transferencia` obre una nova pàgina amb un botó de confirmació, l'atacant no ho pot veure i l'usuari ja no completarà el procediment.

Cal controlar els POST amb `application/x-www-form-urlencoded` i `multipart/form-data`. Hi ha diverses opcions, algunes:

- El servidor comprova la capçalera HTTP referer: la URL des d'on es fa la sol·licitud és coneguda?
- Cookies amb SameSite Strict i Secure: només s'envien navegant dins del mateix origen de la cookie i per HTTPS. No s'envien navegant al mateix origen des d'un lloc extern (p.ex. quan es clica a un hiperenllaç)
- CSRF tokens: servidor genera token aleatori que dóna a l'usuari i l'usuari l'ha d'enviar en fer el submit al formulari. L'atacant mai pot descobrir aquest token.

- Cross-origin resource sharing (CORS)
- Explota les sol·licituds externes des d'Ajax o similars
- Seguretat implementada entre el navegador i el servidor
- El navegador per defecte implementa Same Origin Policy (SOP)
- El navegador pot demanar excepcions CORS al servidor

1 Visito una web que és `https://atacant.test`

2 L'atacant podria haver escrit un codi:

```
<script>xhr.post('https://banc.test/transferencia')</script>
```

3 Si teniu sessió oberta amb el banc, aquest Ajax enviarà la petició amb les vostre cookies, perill?

4 No, tranquils! el navegador bloqueja la petició perquè no compleix Same Origin Policy: `atacant.test` \neq `banc.test`

5 Qualsevol petició Ajax està restringida per defecte a SOP

- Però bé podria ser legítim que hi hagués una aplicació a `appexterna.test` que treballés amb la API de `api.banc.test`
- Aleshores, el CORS permet excepcions a SOP
- GET, HEAD, POST[urlencoded o form-data]: CORS-safelisted perquè, com s'ha vist abans, ja han d'estar protegits contra CSRF
- CORS-safelisted

```
GET https://api.banc.test/transferencia  
ORIGIN: appexterna.test
```

```
200 Ok
```

```
Access-Control-Allow-Origin: https://appexterna.test
```

o retorna error si `api.banc.test` no permet l'accés des de `appexterna.test`

- No CORS-safelisted (POST, PUT, DELETE...): preflight

```
OPTIONS https://api.banc.test/transferencia
```

```
ORIGIN: appexterna.test
```

```
Access-Control-Request-Method: POST
```

```
200 Ok
```

```
Access-Control-Allow-Origin: https://appexterna.test
```

```
Access-Control-Allow-Methods: POST, PUT, DELETE
```

o servidor retorna error si `banc.test` no permet l'accés des de `appexterna.test`

- Si preflight és correcte:

```
POST https://banc.test/transferencia
```

- Cross site scripting (XSS)
- Explota la confiança d'un usuari amb un lloc web
- Seguretat implementada pel desenvolupador web
- L'atacant injecta un codi javascript dins d'un lloc web i per tant es pot saltar els CORS i part del CSRF

- Hi ha una pàgina a `https://banc.test` que admet inserir comentaris públics
- L'atacant insereix el comentari

```
<script>xhr.post('https://banc.test/transferencia')</script>
```

- Tenim sessió oberta al banc i visitem aquesta pàgina de comentaris
- El SOP és correcte **Perill!!!**, l'Ajax té accés a les cookies strict **Perill total!!!**

- Hi ha una pàgina a `https://banc.test/cerca?t=bondia` que permet cerques
- Si no es troba el text de cerca la pàgina elabora una resposta inserint el text demanat:

No s'ha trobat bondia

- L'atacant us envia l'enllaç camuflat
`https://banc.test/cerca?t=<script>xhr.post('https://banc.test/transferencia')</script>`

No s'ha trobat

```
<script>xhr.post('https://banc.test/transferencia')</script>
```

- El SOP és correcte **Perill!!!**, l'Ajax té accés a les cookies strict **Perill total!!!**

- Cal parsejar totes les entrades d'usuaris perquè no hi hagi injecció JS