

# ağ güvenliği- web security 2

Bir siteyi frame/iframe olarak yüklemeysek direkt tek bir yapı olarak yüklendiyse tüm içeriğin domain aynı olmalı.

attacker.com banka.com'a link verebilir, img kendi sitesinde src="bank.com/img" gibi dahil edebilir. Ama dosyanın içeriğini görüp düzenleyemez.

## SOP- Origins

JS, XMLHttpRequest veya JQuery ile bir şeylerin yüklenmesi için istek yollayabilir.

### Malicious XMLHttpRequests

```
// running on attacker.com
$.ajax({url: "https://bank.com/account",
  success: function(result){
    $("#div1").html(result);
  }
});

// Will this request run?
// Should attacker.com be able to see Bank Balance?
```

You can only read data from **GET** responses if they're from the same origin (or you're given permission by the destination origin to read their data)

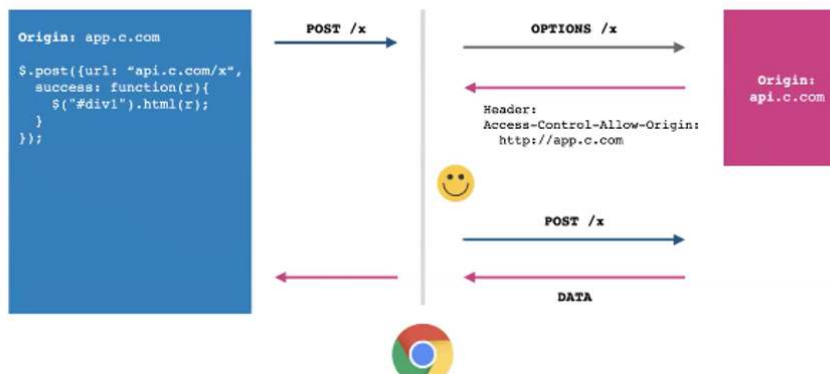
You cannot make **POST/PUT** requests to a different origin... unless you are granted permission by the destination origin (*usually*, caveats to come later)

XMLHttpRequests requests (both sending and receiving side) are policed by **Cross-Origin Resource Sharing (CORS)**

## ACAO(Access-Control-Allow-Origin)

Farklı originden gelen JS isteği için(GET) okuma izni

# CORS Success



Wildcard Origins: Her GET isteğine açık. Public bir şey yapıyorsan okey.

## SOP - Cookies

domain, path kontrolü yapılır.

# Scoping Example

name = cookie1  
value = a  
domain = login.site.com  
path = /

name = cookie2  
value = b  
domain = site.com  
path = /

name = cookie3  
value = c  
domain = site.com  
path = /my/home

cookie domain is suffix of URL domain  $\wedge$  cookie path is a prefix of URL path

	Cookie 1	Cookie 2	Cookie 3
<a href="#">checkout.site.com</a>	No	Yes	No
<a href="#">login.site.com</a>	Yes	Yes	No
<a href="#">login.site.com/my/home</a>	Yes	Yes	Yes
<a href="#">site.com/account</a>	No	Yes	No

# Setting Cookie Scope

Websites can set a scope to be any suffix of domain and prefix of path

- ✓ cs155.stanford.edu can set cookie for cs155.stanford.edu
- ✓ cs155.stanford.edu can set cookie for stanford.edu
- ✗ stanford.edu cannot set cookie for cs155.stanford.edu
  
- ✓ website.com/ can set cookie for website.com/
- ✓ website.com/login can set cookie for website.com/
- ✗ website.com cannot set cookie for website.com/login

## No Domain Cookies

Domain set edilmez. Cookie POST'u yollayan hostname haricinde kimseye gitmez.

### Cookie SOP Policy

cs.stanford.edu/zakir cannot see cookies for cs.stanford.edu/dabo  
(cs.stanford.edu cannot see cookies for cs.stanford.edu/zakir either)

Are Dan's Cookies safe from Zakir? **No, they are not.**

```
const iframe = document.createElement("iframe");
iframe.src = "https://cs.stanford.edu/dabo";
document.body.appendChild(iframe);
alert(iframe.contentWindow.document.cookie);
```

Zakir can access frame's cookies by DOM SOP

Çakışma oldu.

## HttpOnly Cookies(Secure Cookies)

Bu çakışmayı ortadan kaldırmak için HttpOnly Cookies kullanırız. DOM üzerinden erişim kapatılıyor. Buradaki problem de network attackerlar. Bunu çözmek için de sadece ve sadece HTTPS kullanıldığında HttpOnly Cookies yolluyoruz.

## Cookies & Web Attacks

### Cross-Site Request Forgery(CSRF)

CSRF bir saldırı yöntemi. Farklı sayfaya gerçekleştirmek istemeyeceği isteklerde bulunuluyor.

Cookie based authenticationlar değişikliğe sebep olan istekleri(POST vs.) önlemede yeterli olmuyor. Yaygınca kullanılan dört teknik var.

## Referer Validation

Adrese göderen bir önceki sayfanın kaydı tutuluyor.

https://bank.com	→	https://bank.com	✓
https://attacker.com	→	https://bank.com	✗
	→	https://bank.com	??

Geldiği alan boşsa ne yapacağız? Bunu için secret validation token var.

## Secret Validation Token

Formdaki gizli alanlarda bir değer gönderiliyor bu değer kontrol edilerek request gerçekleştiriliyor. Bu değerın statik olmaması lazım yoksa saldırgan kopyalar.

How do we come up with a token that user can access but attacker can't?

- ✗ Set static token in form
  - attacker can load the transfer page out of band
- ✓ Send session-specific token as part of the page
  - attacker cannot access because SOP blocks reading content

## Custom HTTP Header

### Force CORS Pre-Flight

Requests that required and passed CORS Pre-Flight check are safe

- Typical **GETs** and **POSTs** don't require Pre-Flight even if **XMLHttpRequest**

Can we force the browser to make Pre-Flight check? And tell the server?

- You can add custom header to **XMLHttpRequest**
  - Forces Pre-Flight because custom header
  - Never sent by the browser itself when performing normal **GET** or **POST**

Typically developers use **X-Requested-By** or **X-Requested-With**

## sameSite Cookies

Browser cookie'yi scope içerisinde mutlaka gönderiyor. Bir req direct site değil cross site req ise gönderilmemesini sağlayacak bir option.

a-) Strict Mode

Never send, bağlantılı link varsa bile.

### b-)Lax Mode

Bağlantılı link varsa GET'e izin veriyor ama POST veya PUT'a izin vermiyor.

CSRF ile authentication dışında internet trafiğine saldırılar da olabilir. Girdiğimiz bir link DNS ayarlarımızı değiştirerek trafiğimize hijacking yapabilir. Bir adrese gittiğimizi sanarken aslında saldırganın gitmemizi istediği adrese gidiyoruz.

## SQL Injection

En kritik saldırı.

Bir input ile bir komutun çalıştırılması hedefleniyor. SQL sorgularında da bunu yapabiliyoruz.

```
$u = $_POST['login']; // zakir'--
$pp = $_POST['password']; // 123

$sql = "SELECT id FROM users WHERE uid = '$u' AND pwd = '$p'";
//      "SELECT id FROM users WHERE uid = 'zakir'-- AND pwd..."
$rs = $db->executeQuery($sql);
//      (No Error)
if ($rs.count > 0 {
    // Success!
}
```

'- - ile kalan SQL kodu comment satırı oldu böylece şifre ne olursa olsun kullanıcı giriş yapabildi

## Causing Damage

```
$u = $_POST['login']; // '; DROP TABLE [users] --
$pp = $_POST['password']; // 123

$sql = "SELECT id FROM users WHERE uid = '$u' AND pwd = '$p'";
//      "SELECT id FROM users WHERE uid = ''DROP TABLE [users]--"
$rs = $db->executeQuery($sql);
// No Error...(and no more users table)
```

Tablo silindi

# Escaping Database Server

```
$u = $_POST['login']; // '; exec xp_cmdshell 'net user add usr pwd'--
$pp = $_POST['password']; // 123

$sql = "SELECT id FROM users WHERE uid = '$u' AND pwd = '$p'";
//      "SELECT id FROM users WHERE uid = '';
//      exec xp_cmdshell 'net user add usr pwd123'-- "

$rs = $db->executeQuery($sql);
// No Error...(and with a resulting local system account)
```

Sistem kullanıcısı tanımlandı

Never trust user input

Parametre işlenmeden önce komut vb. varsa diye temizlenmesi gerekiyor.

## Cross Site Scripting(XSS)

Uygulama güvenilmeyen veriyi alıp browser'a yolladığında gerçekleşir

### Command/SQL Injection

attacker's malicious code is  
executed on app's server

### Cross Site Scripting

attacker's malicious code is  
executed on victim's browser

## Embedded Script

`https://google.com/search?q=<script>alert("hello")</script>`

```
<html>
<title>Search Results</title>
<body>
  <h1>Results for <?php echo $_GET["q"] ?></h1>
</body>
</html>
```

Sent to Browser

```
<html>
<title>Search Results</title>
<body>
  <h1>Results for <script>alert("hello")</script></h1>
</body>
</html>
```

Input script olabilir.

Embedded Script ile cookie çalınabilir. Sizin sunucunuz cookie gönderme isteğinde bulunduğu için güvenlik önlemlerini aşarak saldırgan siteye cookie yollanabilir.

XSS'i engellemek için yapılacak yegane şey:

## Content Security Policy