BİLGİ GÜVENLİĞİ AKADEMİSİ

www.bga.com.tr

Web 2.0 Güvenliği

Örnek Eğitim Notu

Düzenleyen : Faruk GÜNGÖR

Javascript

- 1995 Netscape, Brendan Eich tarafından geliştirildi.
- Dinamik olmasının yanında en önemli iki özelliği;
  - Lambda
  - Closure

Javascript - Lambda

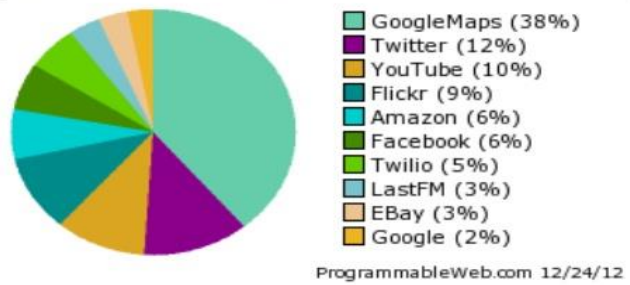
```
var ikiEkle = function (x) {
    return x + 2;
}
ikiEkle(3);     // sonuç 5
```

Javascript - Closure

```
function birEkle (y) {
        return function() {
            return y + 1;
        };
}
var x = birEkle(5);
x();      // sonuç 6
```

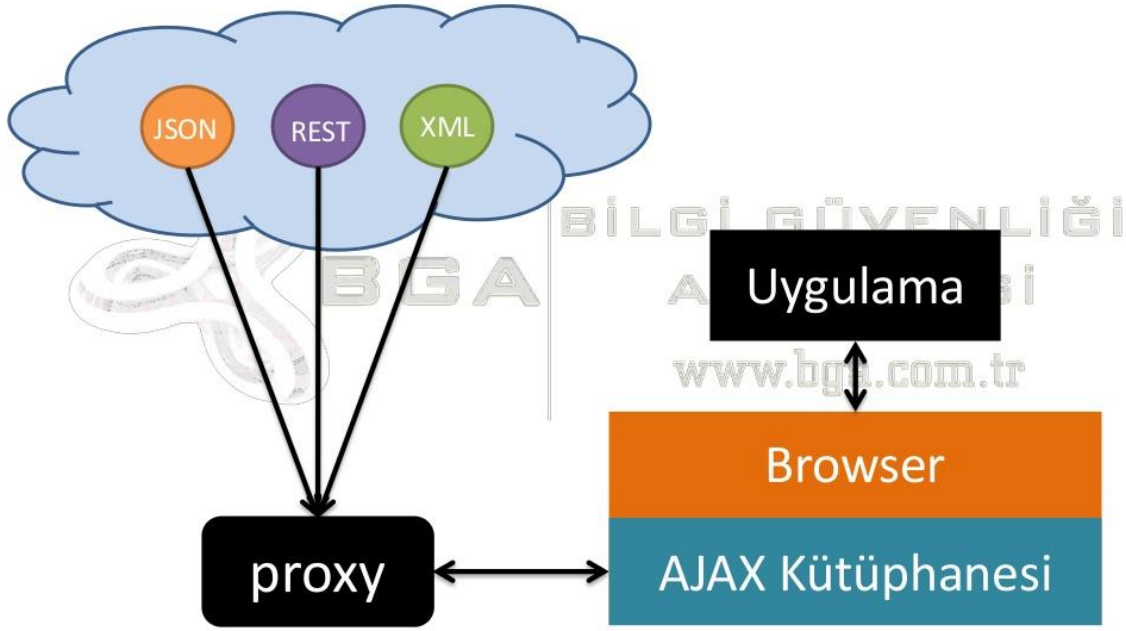
Mashup

- Diğer servislerin verilerinin birleştirilmesi ve sunulması ile oluşturulan servis tipidir.
- Başkasının aklını kullanmak
- Mashup tipleri;
  - İstemci taraflı
  - Sunucu taraflı

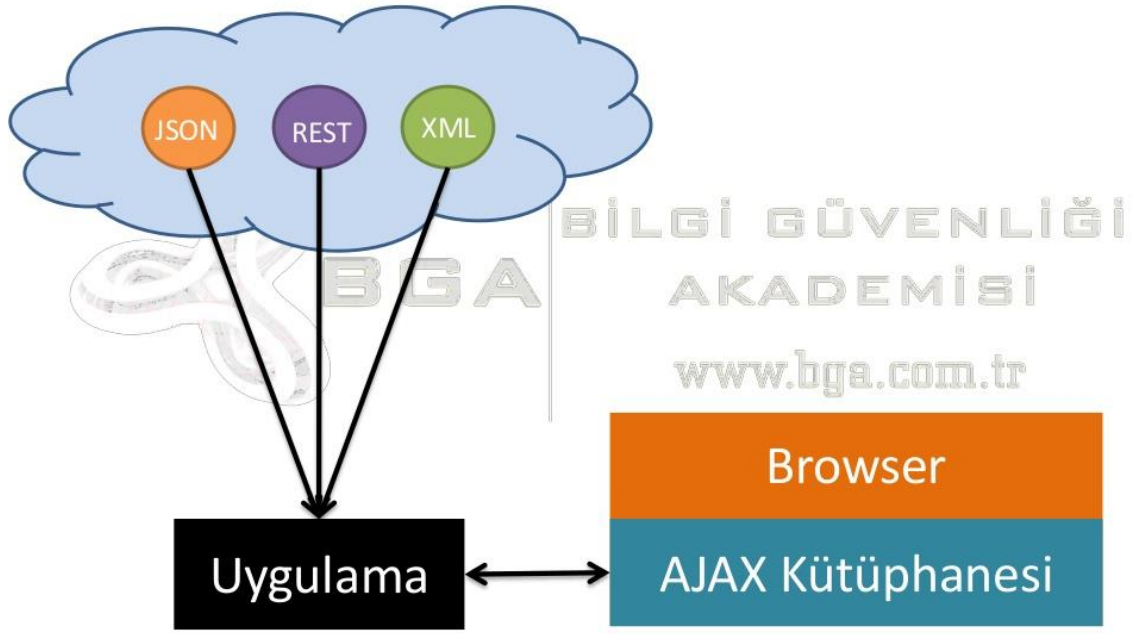


Mashup'larda Kullanılabilen Popüler Servisler

İstemci Taraflı Mashup



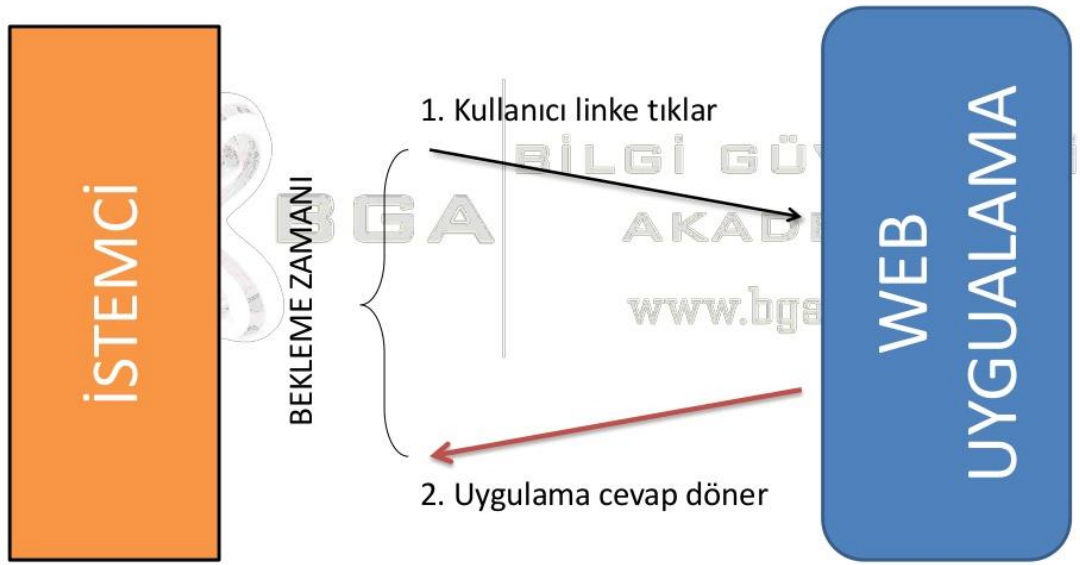
Sunucu Taraflı Mashup



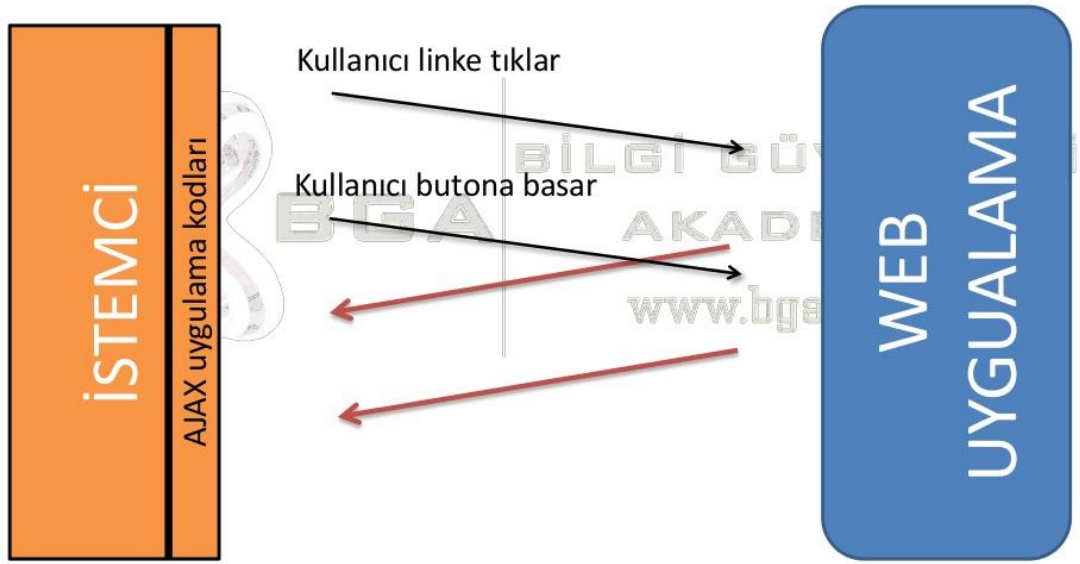
AJAX

- Asynchronous JavaScript and XML
- Asenkron web uygulamaları geliştirmek için kullanılan birbirleri ile uzaktan ilgili web geliştirme teknolojileridir;
  - Javascript
  - DOM, HTML, CSS
  - XMLHttpRequest nesnesi
  - JSON (XML daha azınlıkta)

Senkron


İSTEMCİ
BEKLEME ZAMANI
1. Kullanıcı linke tıklar
2. Uygulama cevap döner
WEB UYGUALAMA

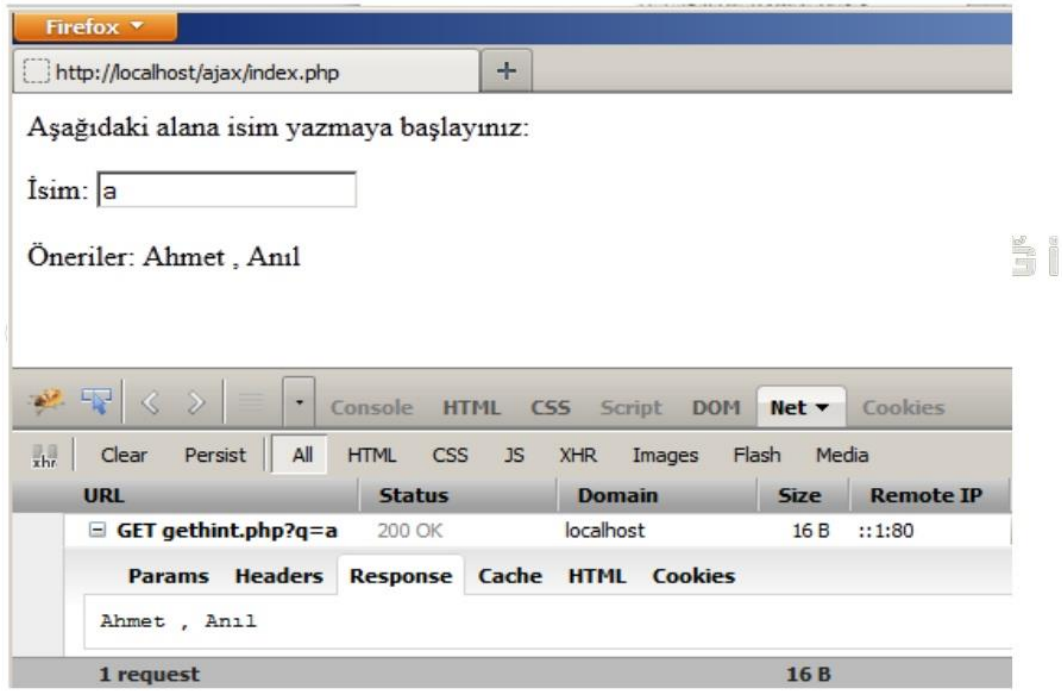
Asenkron


İSTEMCİ
AJAX uygulama kodları
Kullanıcı linke tıklar
Kullanıcı butona basar
WEB
UYGUALAMA

AJAX Örneği

```
<script type="text/javascript">
  function sH(str){
    // kod
  }
</script>
<form>
  İsim: <input type="text" onkeyup="sH(this.value)" />
</form>
Öneriler: <span id="txtHint"></span>
```

AJAX Örneği – İstek / Cevap



AJAX Veri Transfer Şekilleri

- XML
  - Yarı yapısal dil
  - Javascript kullanılan istemciler için anlamsız
- JSON
  - Javascript tabanlı bir notasyon şekli
  - Javascript kullanılan istemciler işlemeye hazır yapı

JSON

- Javascript tabanlı hafif sıklet veri iletişim formatıdır.
- XML ile karşılaştırıldığında, daha okunaklı ve anlaşılır ve daha kolay parse edilebilir.


```
{ “name” : “hagi” , “name” : “ronaldo” }
```

```
[“ulvi” , “rıza”, “metin”, “cevat” ]
```

```
{ “names” : [“feyyaz”, “cüneyt”, “müjdat”] }
```

XML vs. JSON

```xml
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
</popup>
</menu>
```

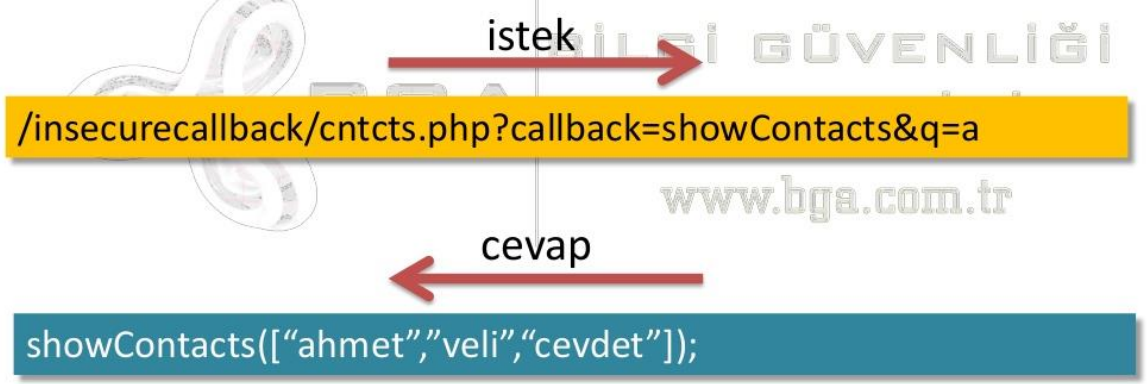
XML

```json
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"}
] }}
```

JSON

Güvensiz Callback'ler

- Dinamik script'ler ile yapılan asenkron veri aktarımı



Güvensiz Callback'ler - Saldırı

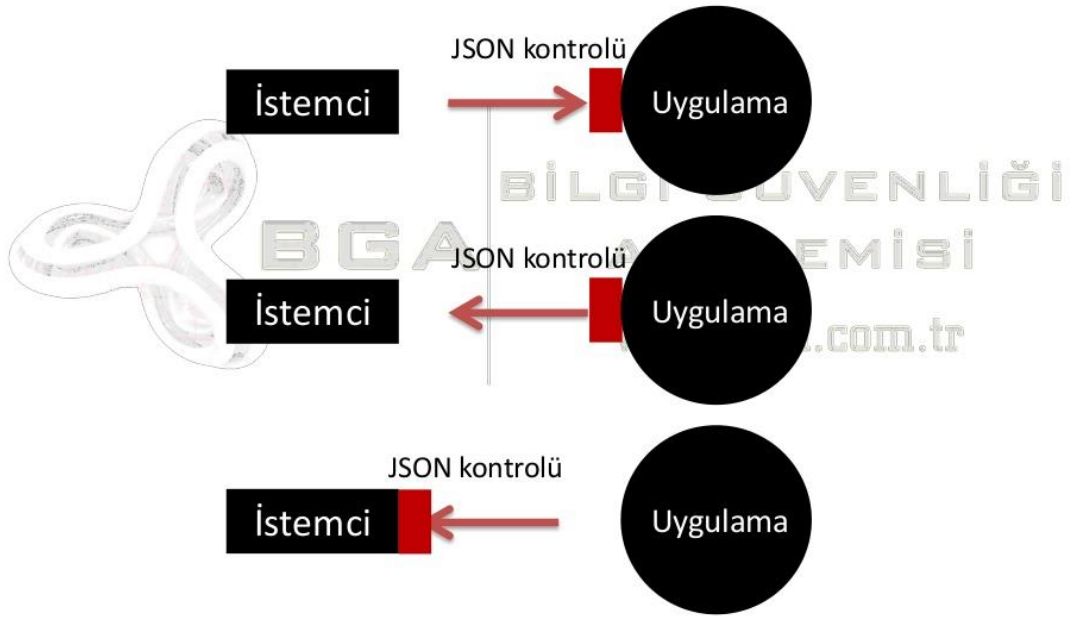
- CSRF kullanılarak kontak listesinin çalınması

```
<script type="text/javascript">
   function hijacked (contacts){
      alert(contacts);
   }
</script>
<script src="http://hedef/cntcts.php?callback=hijacked&q=*">
</script>
```

Güvenli JSON Yönetimi

- JSON verisinin işlenebilmesi için model nesnelere dönüştürülmesi gerekir.
- Dönüştürme işlemi sırasında oluşabilecek injection problemleri engellemek için JSON verileri mutlaka yapısal olarak denetlenmelidir.
  - Sunucu tarafında JSON veriyi alırken
  - Sunucu tarafında JSON veriyi istemciye gönderirken
  - İstemci tarafında JSON veriyi alırken

JSON Denetim Noktaları



JSON Denetimi - Java

```
JSONParser jsonParser = new JSONParser();

try{
   JSONObject json = (JSONObject)jsonParser.parse(jsonString);
}
catch(ParseException pe){
   // invalid JSON
}
```

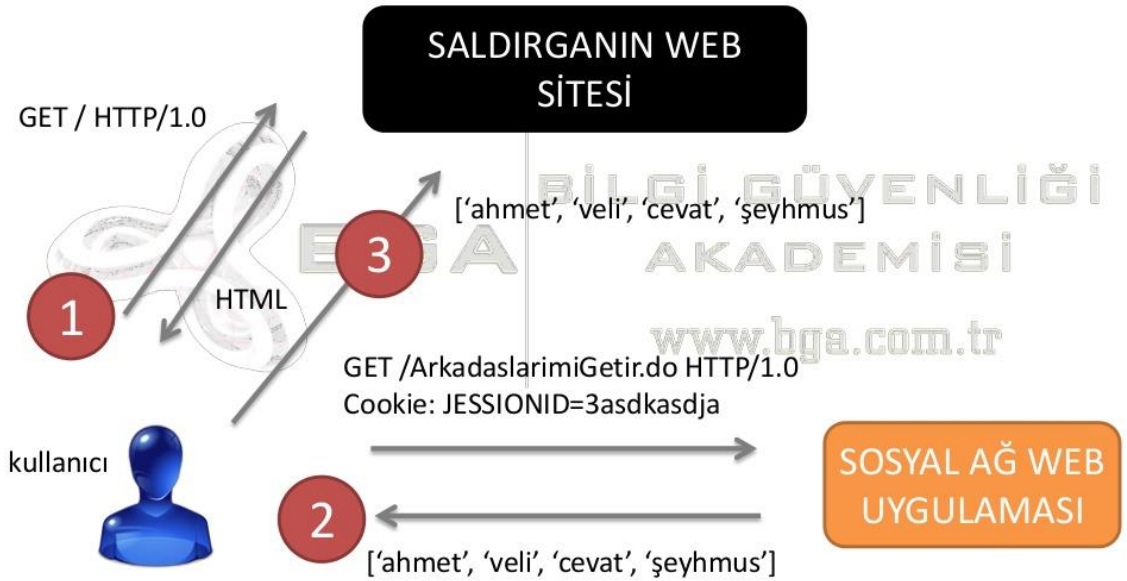
JSON Denetimi - Javascript

```
isValid = true;
try{
   obj = JSON.parse(json);
}
catch(e){
   // Invalid JSON
   isValid = false;
}
```

JSON Hijacking

- 2006'da Gmail'de bulunan bir zafiyet ile popüler olan bir zafiyettir.
- Modern browser'lar ile beraber günümüzde düşük seviyeli bir zafiyettir.

JSON Hijacking – Senaryo



JSON Hijacking - Kod

```
function Array(){
    var orjinalArray = this;
    var yeniArray = function(){
      for(var x in orjinalArray)
           calinanListe += orjinalArray[x] + ",";
       // çalınanListe'yi gönder
     }
    // this nesnesinin dolmasını bekle
     setTimeout(yeniArray, 150);
}
```

JSON Hijacking - Exploit

```
<html>
  <head>
    <script>  {Array OVERRIDE} </script>
  </head>
  <body>
    <script src="http://sosyalag.com/ArkadaslarimiGetir.do">
    </script>
   </body>
</html>
```

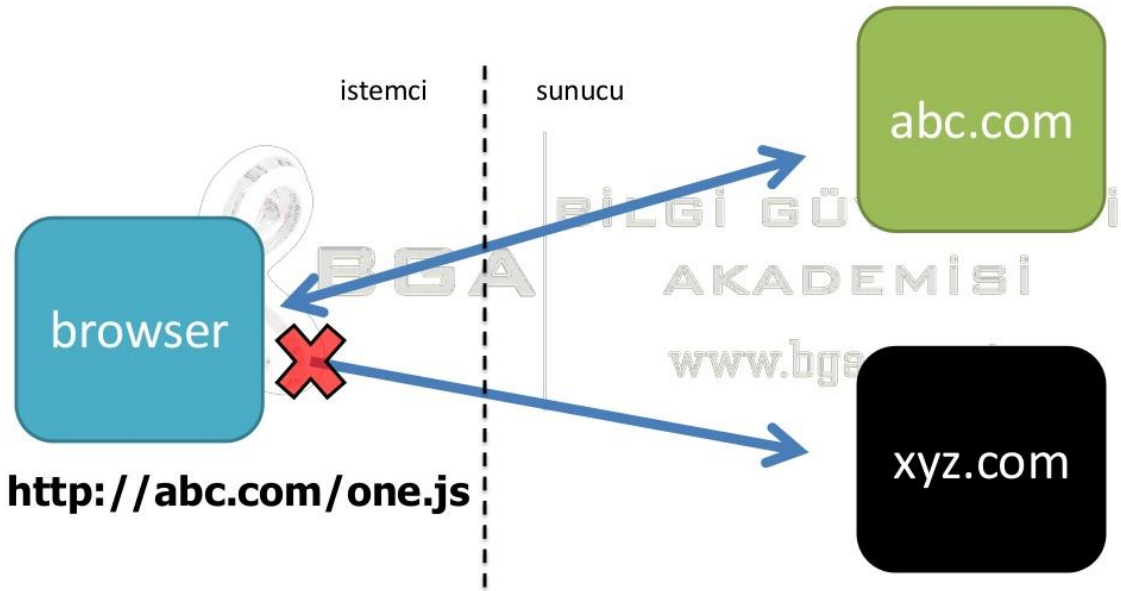
JSON/Javascript Önlemler

- Eski tarayıcılar için;
  - Hassas bilgi dönen AJAX isteklerinin GET yerine **sadece** POST ile çalıştırılması
  - JSON array yerine JSON object dönülmesi {…}
  - CSRF token'ları
  - Hassas bilgi;
    - Kullanıcı listesi, profili, mesajları, geçmişi v.b.
    - Kişisel bilgiler; adres, telefon, isim soyisim, v.b.
    - Sağlık bilgileri, finansal bilgiler v.b.

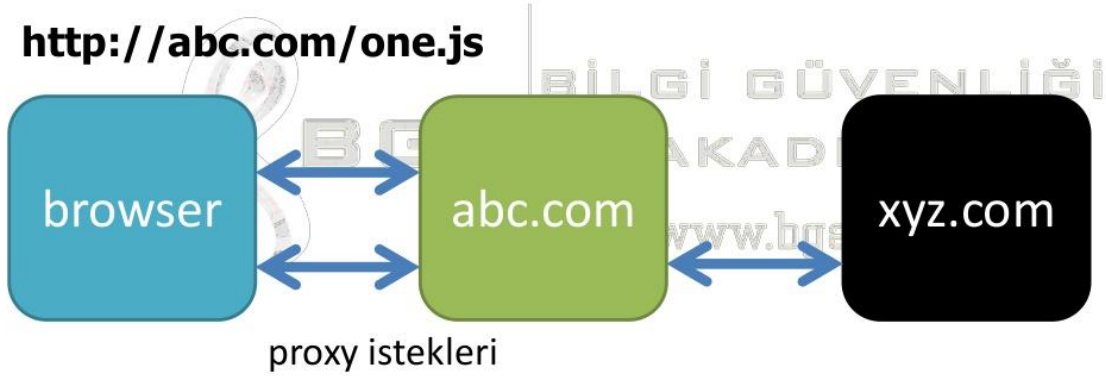
Cross Domain Access - CORS

- SOP'un yani Aynı Kaynak Politikası'nın uyguladığı *cross domain* istek kısıtlamalarına karşı geliştirilen bir W3C standardıdır.
- Kaynak: {**protokol**, **alan ismi**, **port**}

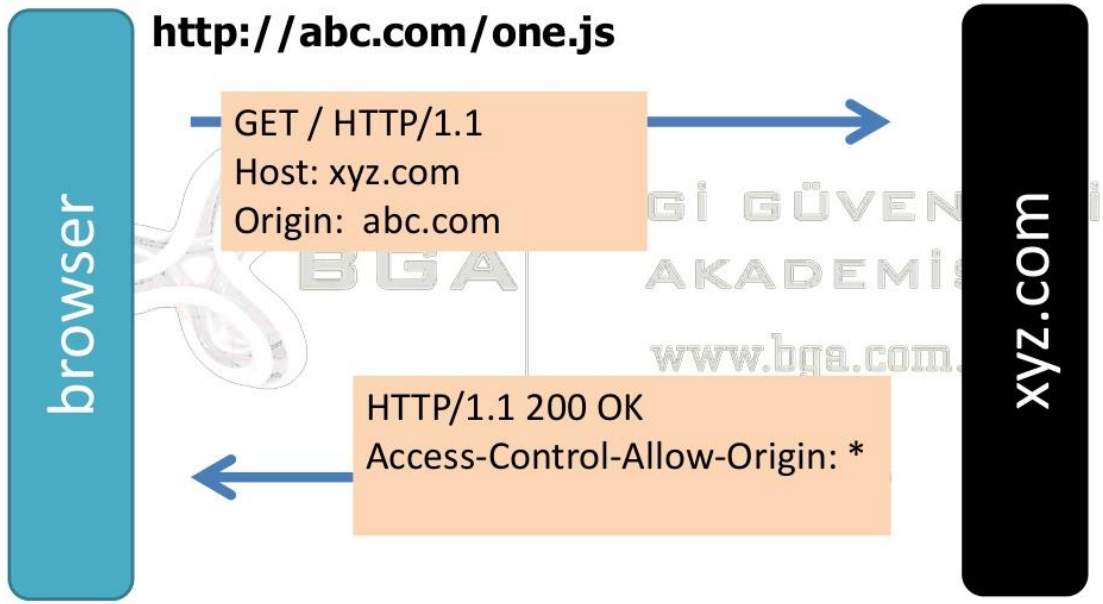
SOP



SOP ByPass



CORS Örnek


http://abc.com/one.js
browser
GET / HTTP/1.1
Host: xyz.com
Origin: abc.com
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
xyz.com

CORS - Browser Desteği

IE	Firefox	Chrome	Safari	Opera	iOS Safari
7.0	14.0	20.0	4.0	11.5	4.0-4.1
8.0	15.0	21.0	5.0	11.6	4.2-4.3
9.0	16.0	22.0	5.1	12.0	5.0-5.1
10.0	17.0	23.0	6.0	12.1	6.0
	18.0	24.0		12.5	
	19.0	25.0			

Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
	2.3		11.0		
	3.0		11.1		
	4.0		11.5		
5.0-7.0	4.1	7.0	12.0	18.0	15.0
		10.0	12.1		

Güvenli CORS Stratejileri

- *Access-Control-Allow-Origin:* * değeri dikkatli kullanılmalıdır.

Güvenli CORS Stratejileri

- *Origin: domainismi* değeri klasik yetkilendirme kontrolleri için kullanılmamalıdır.

```
String origin = request.getHeader("Origin");
if(origin!=null && origin.equals("www.abc.com"))
   // hassas bilgileri istemciye dön
else
   // normal sayfayı göster
```

Güvenli CORS Stratejileri

- *Origin* başlığının değeri hedef uygulamalarda mutlaka kontrol edilmelidir.

```
response.setHeader("Access-Control-Allow-Origin", "www.abc.com");
// isteği işlemeye devam et
```

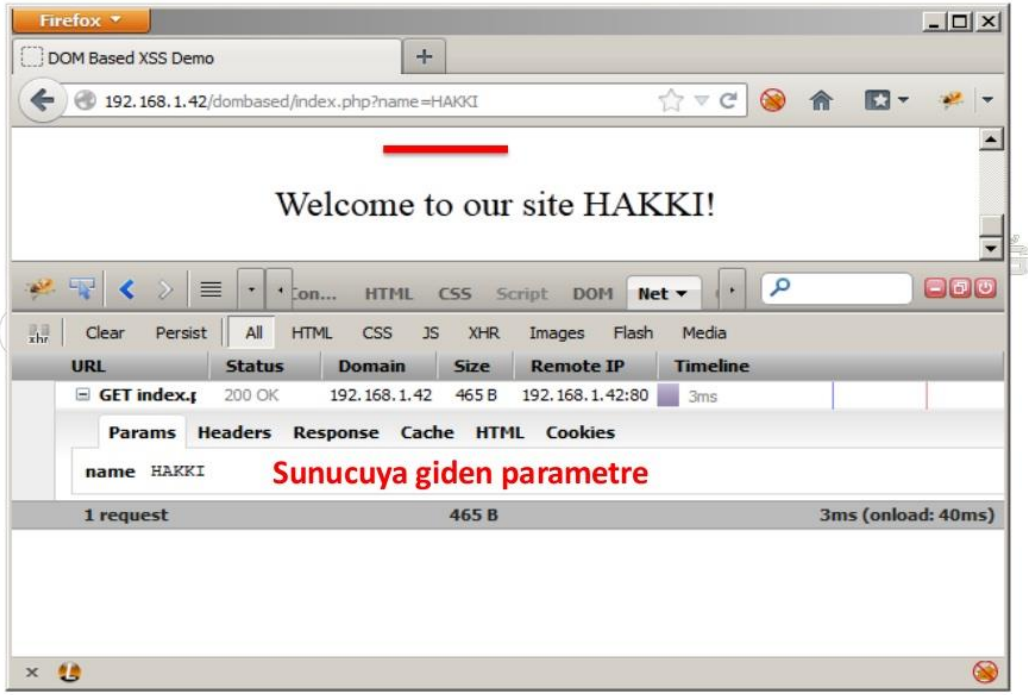

```
String origin = request.getHeader("Origin");
if(origin!=null && origin.equals("www.abc.com"))
   response.setHeader("Access-Control-Allow-Origin", "www.abc.com");
else
   return; // isteği işlemeden dön
```

DOM Tabanlı XSS

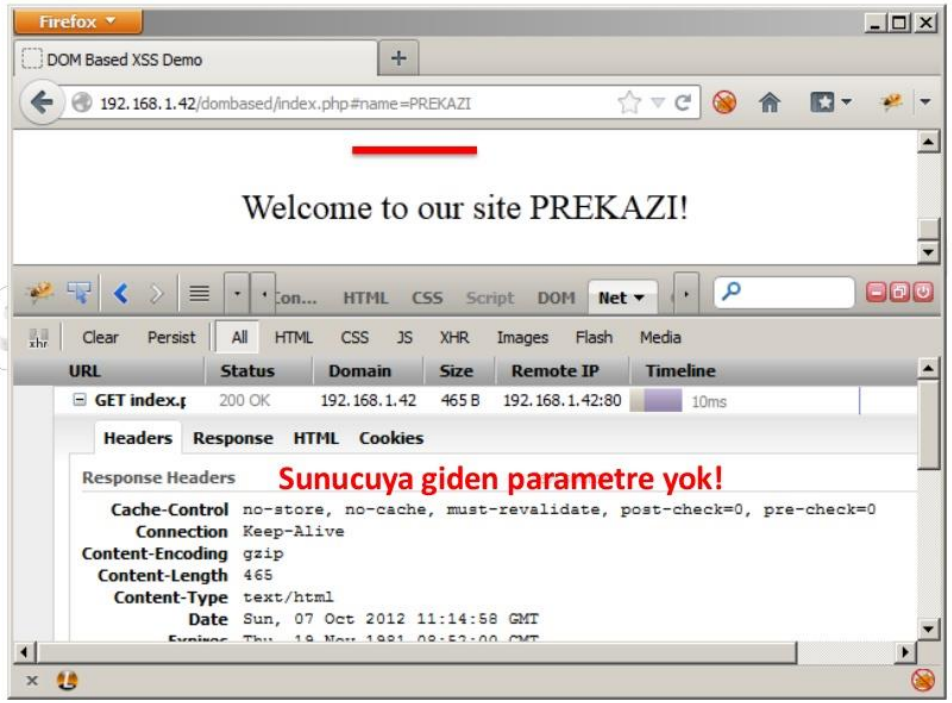
- Javascript'in neden olduğu XSS çeşididir.

```
<script>
 function printName(){
   var index = length = document.URL.length;
   if(document.URL.indexOf("name=")  !=  -1)
      index = document.URL.indexOf("name=") + 5;
   var substr = document.URL.substring(index,length);
   document.write(unescape(substr));
 }
```

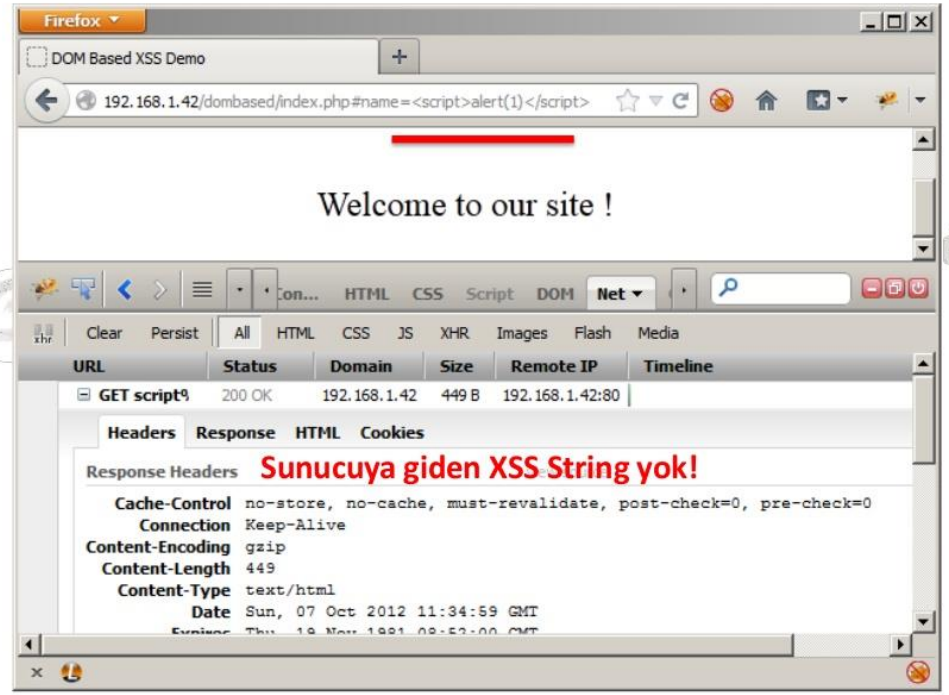
DOM Tabanlı XSS - Senaryo



DOM Tabanlı XSS - Senaryo



DOM Tabanlı XSS - Senaryo



DOM Tabanlı XSS - Önlemler

- HTML Kodlama kullanılması

```
element.innerHTML = “<%= HTMLEncode(param) %>”;
```

```
element.outerHTML = “<%= HTMLEncode(param) %>”;
```

```
document.write(“<%= HTMLEncode(param) %>”);
```

```
document.writeln(“<%= HTMLEncode(param) %>”);
```

DOM Tabanlı XSS - Önlemler

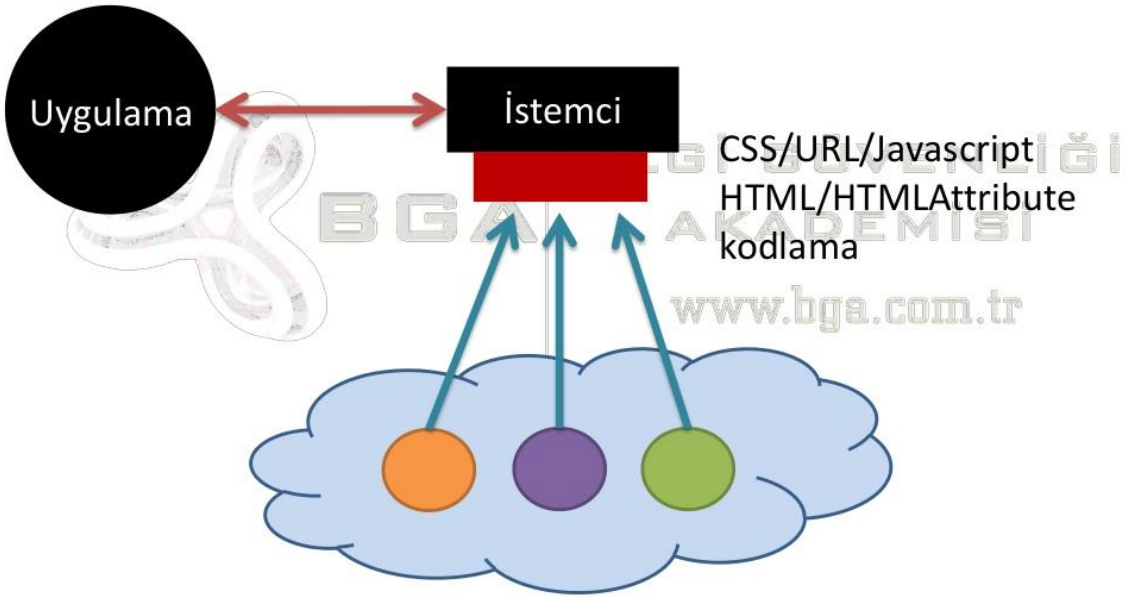
- Javascript Kodlama kullanılması

```
var x = document.createElement(“input”);
x.setAttribute(“name”, “cname”);
x.setAttribute(“value”, “<%= JSEncode(param) %>”);
var form1 = document.forms[0];
form1.appendChild(x);
```

İstemci Taraflı Kodlama

- Browser'da, güvensiz mashup kaynaklarını çağıran javascript uygulamalarının alması gereken önlemler
- Sunucu taraflı XSS koruma yöntemlerinin istemci tarafındaki versiyonu
- JQEncoder

İstemci Taraflı Kodlama



HTML ve HTML Attribute Kodlama

HTML Kodlama:

```
$('#item1').html(data[0]);
```

```
$('#item1').html($.encoder.encodeForHTML(data[0]));
```

HTML Attribute Kodlama:

```
$('#item2').html('<div width="' + data[1] + '">');
```

```
ec = $.encoder.encodeForHTMLAttribute('width', data[1], false);
$('#item2').html('<div ' + ec+ '>);
```

Javascript ve URL Kodlama

Javascript Kodlama:

```
$('#item4').html('<div onclick="s=\'' + data[3] + '\'">');
```

```
ec = $.encoder.encodeForJavascript(data[3]);
$('#item4').html('<div onclick="s=\'' + ec + '\'">');
```


URL Kodlama:

```
$('#item5').html('<a href="?a=' + data[4] + '">link</a>');
```

```
ec = $.encoder.encodeForURL(data[4]);
$('#item5').html('<a href="?a=' + ec + '">link</a>');
```

CSS Kodlama

CSS Kodlama:

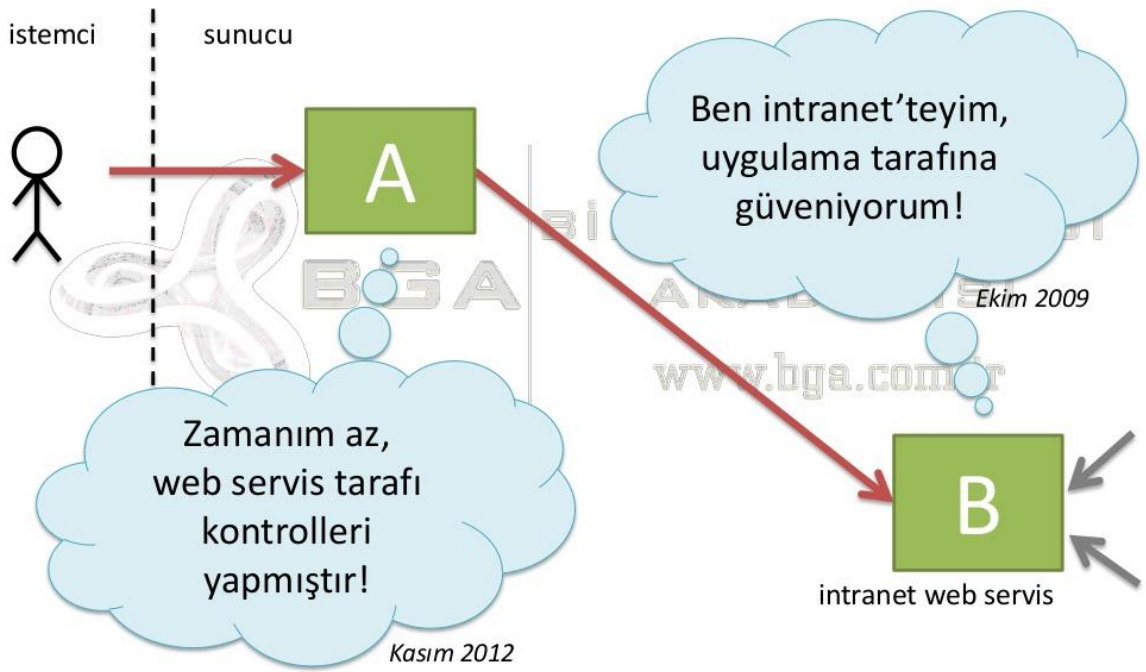
```
$('#item3').html('<div style="color:' + data[2] + ';">');
```

```
ec = $.encoder.encodeForCSS('background-color', data[2], false);
try{
  $('#item3').html('<div style="' + ec + '">');
}
catch(e){
   alert('encodeForCSS throws exception: ' + e);
}
```

SOA Güven Karmaşası

- Servis tabanlı mimaride karşılaşılan en büyük dizayn problemlerinde biri güven karmaşasıdır.
- Servis veren veya tüketen bileşenler güvenlik kontrollerini birbirlerinin sorumluluğu olarak görmektedirler.
- Bu "yanlış sorumluluk transferi" ciddi güvenlik problemlerine yol açmaktadır.

SOA Güven Karmaşası



Kontrol Stratejisi

- SOA'nın mantığına uygun olarak her servis aldığı parametre değerleri ile yapabileceği maximum kontrolü gerçekleştirmelidir.


```
function paraOde(int id, String tamisim, String tckno, int miktar,
                 DateTime odemeTarihi, String ccNo){
// tckno ile tamisim örtüşüyor mu?
// odemeTarihi uygun mu?
// ccNo ile tamisim örtüşüyor mu?
// miktar doğru mu?
// ...
}
```