Web Yazılım Geliştirme

Ders 10 - CORS, CSRF ve XSS

Erciyes Üniversitesi
Bilgisayar Mühendisliği Bölümü

Eğitmen: Ömür ŞAHİN

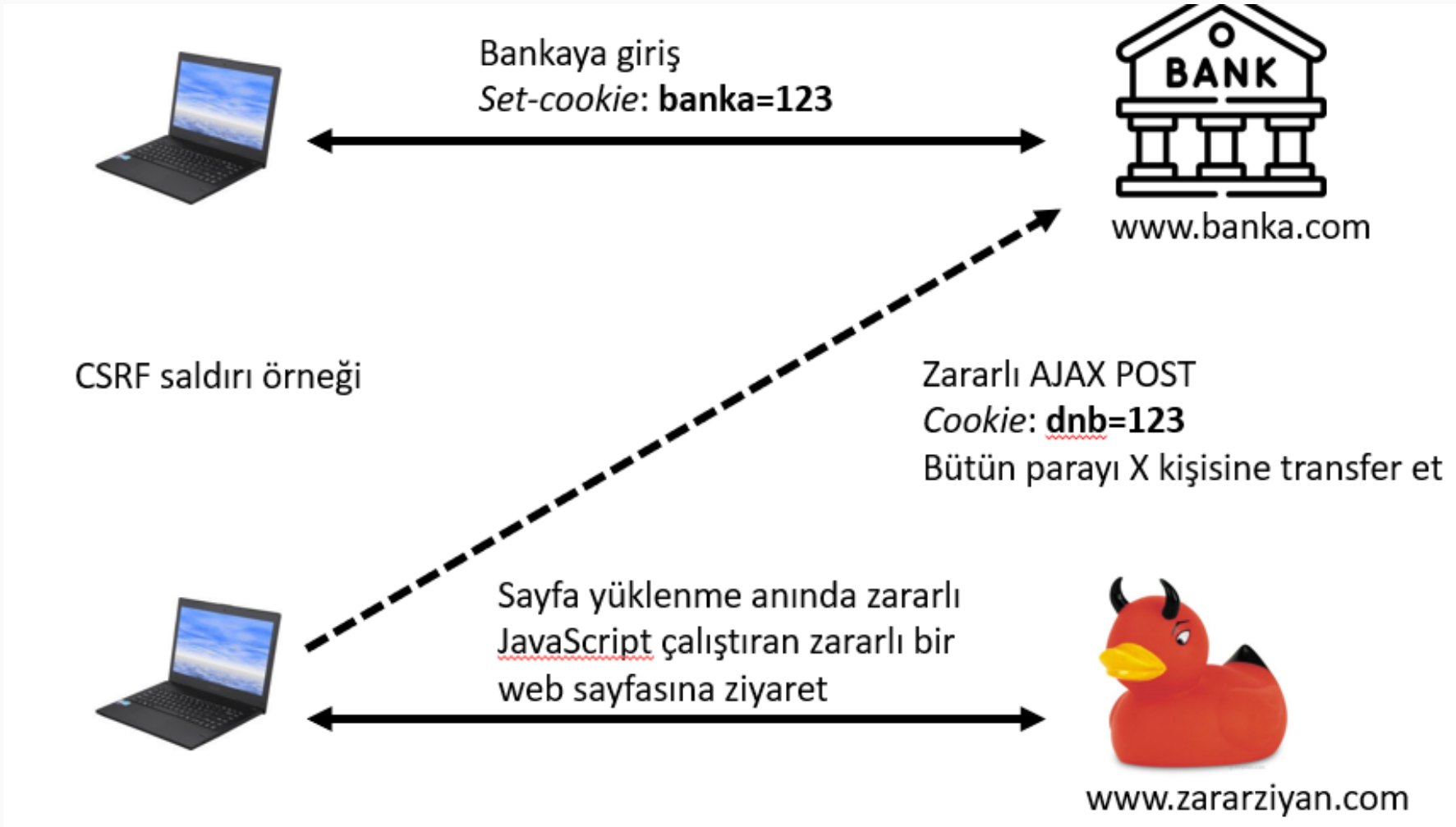
Hedefler

- Cross-Origin Sharing (CORS)'u anlamak
- Cross-Site Request Forgery (CSRF)'nin risklerini anlamak
- Kullanıcı girdilerini düzenleme, temizleme
- XSS ataklarını anlama

1-CORS ve CSRF

HTTP ve Cookie

- Tarayıcı üzerinden foo.com adresine istekte bulunduğumuzda session'ı da içeren bütün cookie'ler ilgili domain ilişkilendirilerek kaydedilir ve bütün isteklerde bu cookie'ler kullanılır.
- Bütün HTTP çağrılarına bu uygulanır.
  - HTML içerisindeki a ve form taglarına
  - AJAX ile gerçekleştirilen XMLHttpRequest ve fetch çağrılarına
- Ancak buradaki problem nedir?
  - Cross-Site Request Forgery (CSRF) saldırısı


Bankaya giriş
Set-cookie: **banka=123**
BANK
www.banka.com
CSRF saldırı örneği
Zararlı AJAX POST
Cookie: **dnb=123**
Bütün parayı X kişisine transfer et
Sayfa yüklenme anında zararlı JavaScript çalıştıran zararlı bir web sayfasına ziyaret
www.zararziyan.com

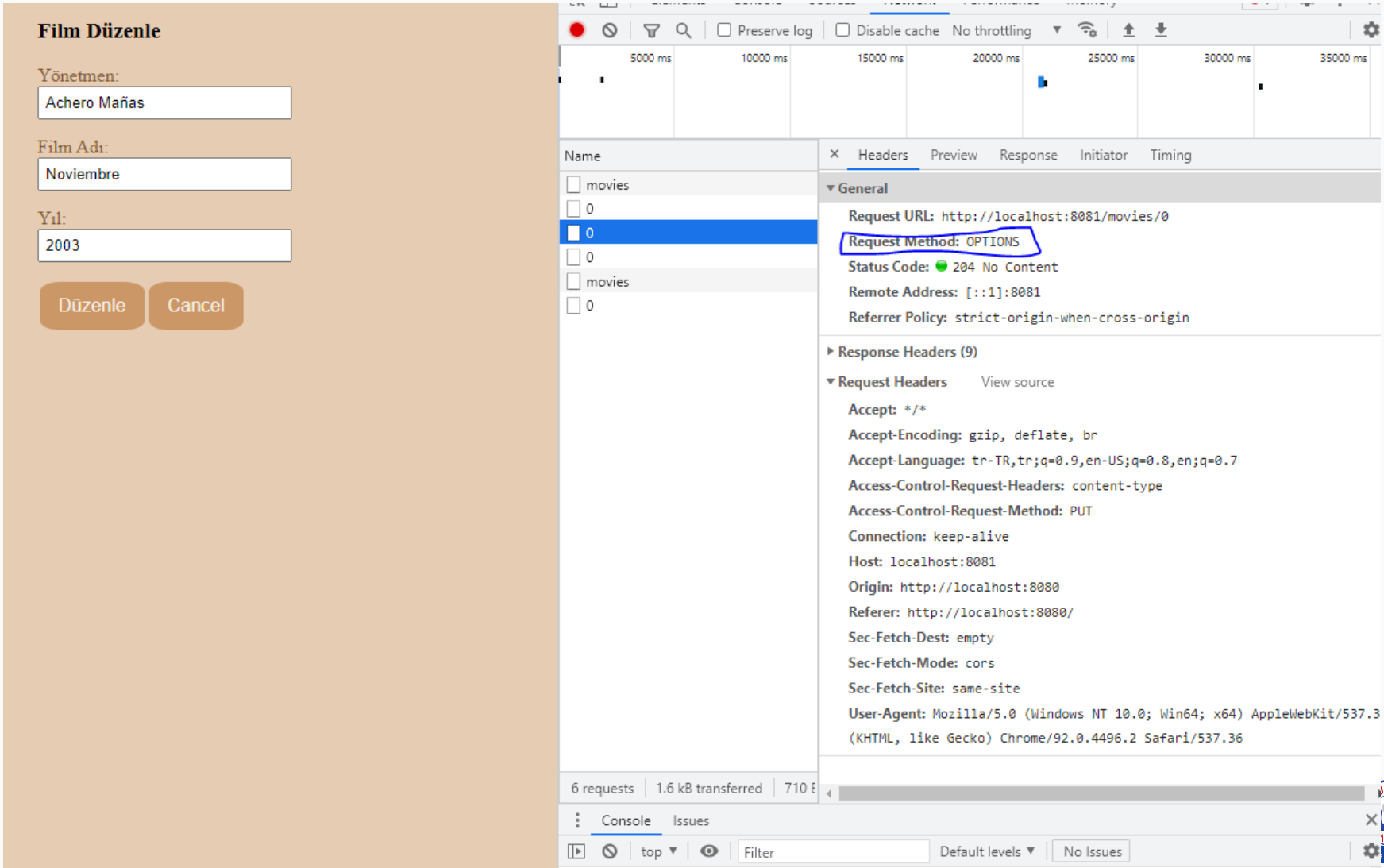
Cross-Origin Resource Sharing (CORS)

- Varsayılan olarak, X sitesinden indirilen JS bir başka Y sitesine çağrıda bulunamaz.
  - Tarayıcı JS'in indirildiği domain'e (ip:port) istek atılmasına izin verir.
  - Örnek: saldirgan.com.tr adresinden indirilen bir JS dosyası yalnızca saldirgan.com.tr adresine AJAX çağrıları atabilir.
- Böyle bir HTTP çağrısı yapılmaya çalışıldığında tarayıcı ilk başta OPTIONS metodu ile çağrı yaparak ön kontrol yapacaktır.
  - Bu orijinal çağrının Y sunucusuna yapılıp yapılamayacağını soracaktır.
  - Y sunucusu ise tarayıcıya bu isteği yapıp yapamayacağını söyleyecektir.
  - Eğer OK cevabı gelirse orijinal çağrı gerçekleştirilir.
  - Böylelikle 2 çağrı yapılmış olur.

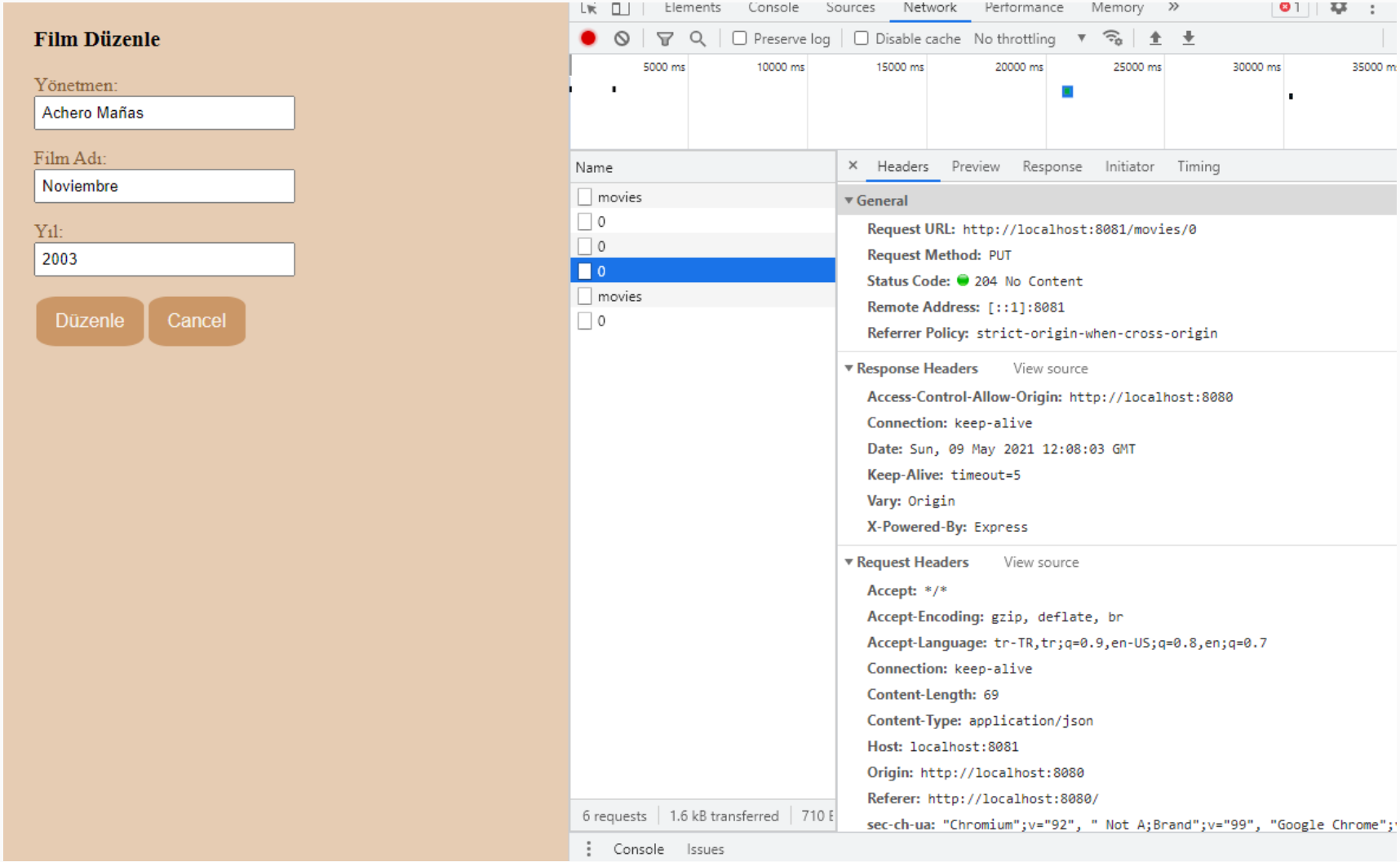
Frontend ve Backend ayrıldığında

- Ders 8'deki movie uygulamasını hatırlayalım. Frontend localhost:8080 üzerinden backend ise localhost:8081 üzerinde çalışmaktaydı.
- CORS ile ilgili problemi çözmek zorunda kalmıştık.
- Örnek: Bir kaydı güncellemek istersek neler gerçekleşir?

Tarayıcı ilk olarak OPTIONS kullanarak esas eylemin yapılıp yapılamayacağını kontrol eder. Eğer yapılabilir ise PUT işlemi gerçekleşir.



Eğer OPTIONS isteği sonucu OK gelirse PUT gerçekleşir.



Ön kontrolsüz İstekler

- Tarayıcılar bütün HTTP isteklerinde ön kontrol yapmazlar.
- Aşağıdaki content-type ile GET, HEAD, POST metotları:
  - application/x-www-form-urlencoded
  - multipart/form-data
  - text/plain
- Bu konular doğru şekilde ele alınmaz ise güvenlik zafiyeti oluştururlar.

CORS ve GET

- GET isteğinde OPTIONS ile ön kontrol yapılmasa da güvenli kalmaya devam edebilmektedir.
- Sunucu yalnız OPTIONS değil GET de dahil olmak üzere bütün isteklere Access-Control-Allow-Origin cevabı verebilmektedir.
- Eğer origin eşleşmiyor ise tarayıcı durum kodu da dahil olmak üzere cevabın tüm içeriğini siler.
- Böylelikle GET isteği yapılır ancak JS yanıtı okuyamaz.

GET ve Yan etki

- GET isteklerinde OPTIONS ile ön hazırlık yapılmaz.
- Eğer CORS Origin ile eşleşmez ise JS'in gelen cevabı okumasına izin verilmez.
- Ancak GET isteği yine de yapılır!
- Eğer sunucuda yan etkiler varsa CORS korumasından bağımsız olarak eylemler gerçekleştirilecektir.
  - Örnek: Kaynak oluşturma/silme işlemleri, "GET /api/data?action=delete"
- Bu yüzden GET isteklerinin sunucuda bir yan etki oluşturmadığından emin olmak oldukça ÖNEMLİDİR.

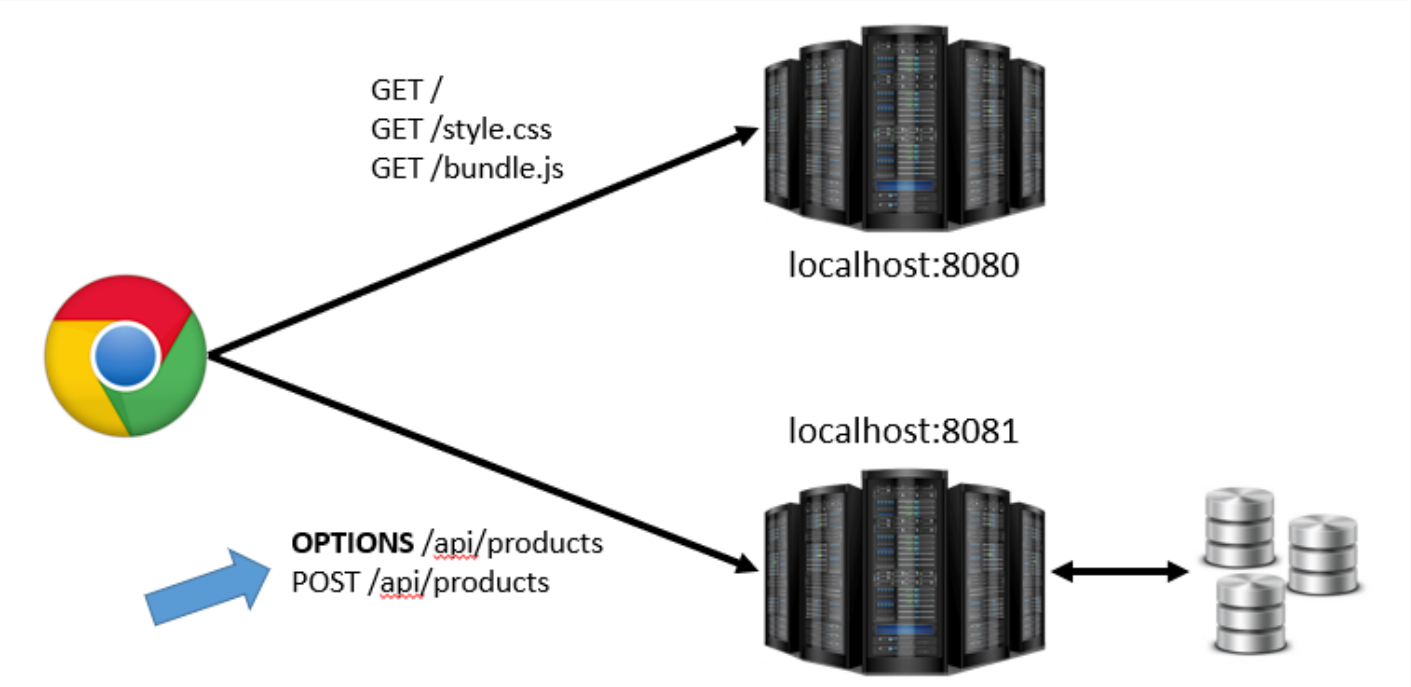
Ön kontrolsüz POST

- Bu aşağıdaki content-type'lar ile gerçekleşir.
  - application/x-www-form-urlencoded
  - multipart/form-data
  - text/plain
- SPA'larda, JSON API'lara bağlı iseniz genellikle sorun olmamaktadır.
- HTML <form> taglarında ön kontrol yapılmadığı için Server-Side-Render geleneksel web uygulamalarında bu problem yaşanmaktadır.
- Çözüm: CSRF Token'lar. Ancak bu ders kapsamında incelemeyeceğiz.
  - SameSite cookieleri de yardımcı olabilmektedir.

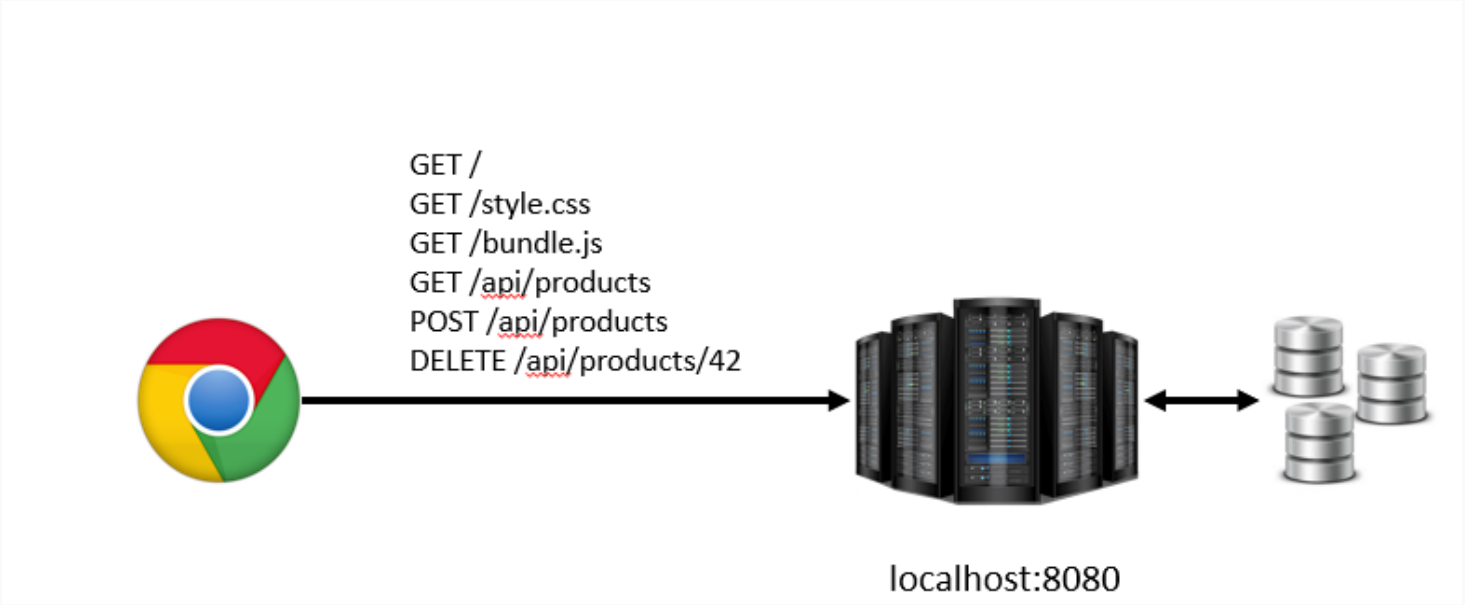
Performans

- Ön kontrol işlemi HTTP istek sayısını ikiye katlamaktadır.
- Önbellekleme bazı istek sayısını düşürebilmektedir ancak problem devam etmektedir.
- Not: JSON verisini content-type: text/plain göndermek gibi akıllıca bir çözüm denememelisiniz. Hızınızı artırır ancak CSRF ataklarına açık hale getirecektir.

Frontend ve backend birbirinden ayrı ise backend responseları için CORS headerının aktif edilmesi gerekir. Bu durumda da performans düşecektir.

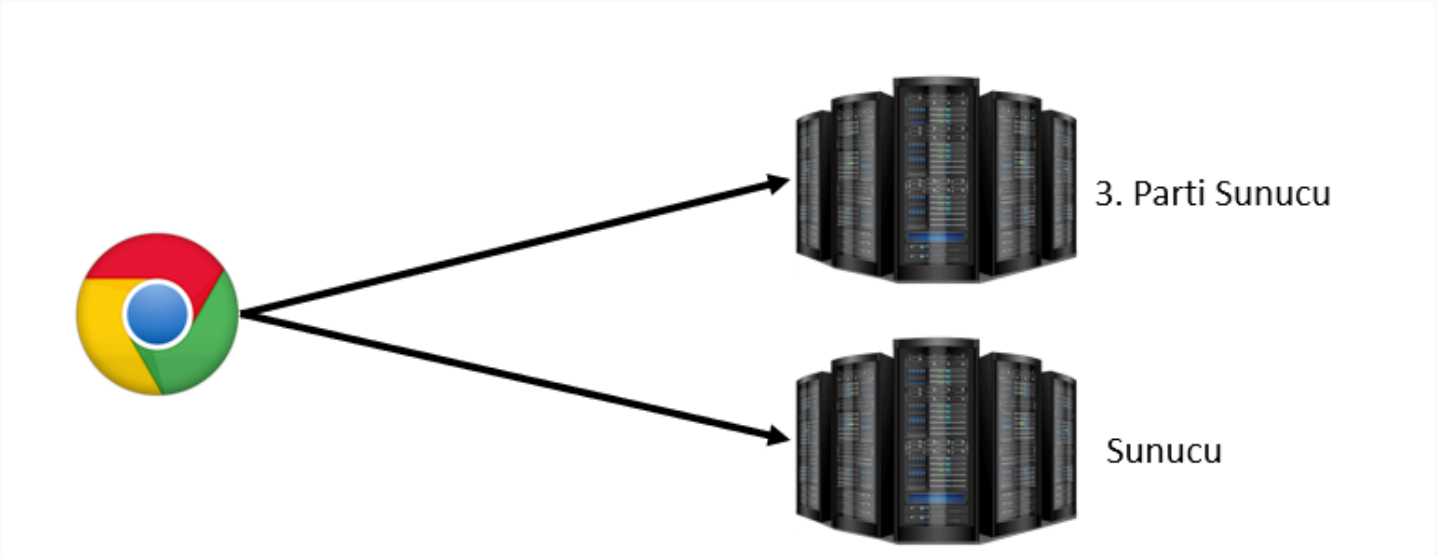


Bütün veri aynı Origin'den geliyor ise CORS aktif edilmez ve CSRF ile ilgili bir problem yaşanmayacaktır.



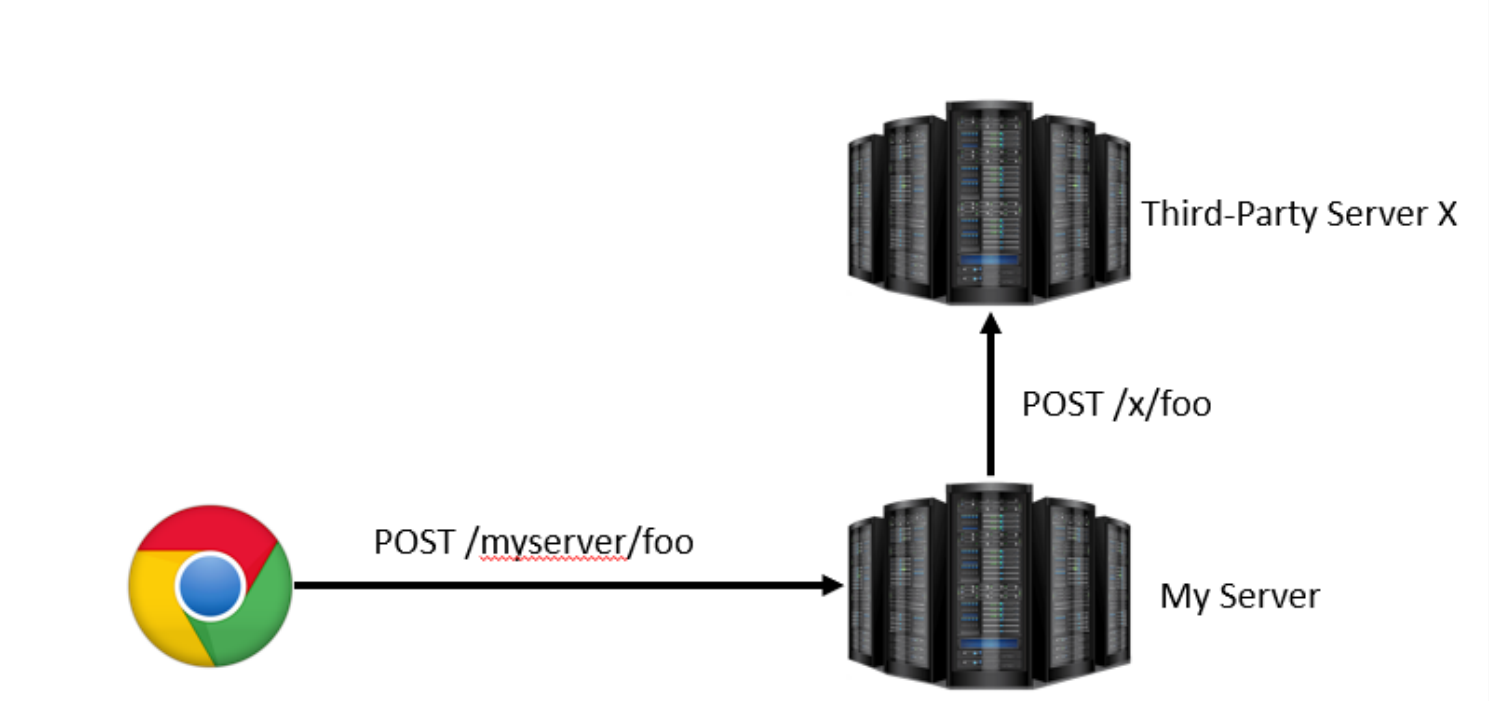
3. Parti Uygulamalar

- JS ile iletişim kurulacak ise bu uzak sunucularda CORS aktif edilmelidir.
- Ancak bu ayar değiştirilemez veya ön kontrol isteklerinden kaçınmak istenirse ne olacak?



Proxy Çağrılar

- 3. parti uygulamaları doğrudan JS ile çağırmak yerine kendi sunucunuz tarafından çağırabilirsiniz.
- CORS yalnızca tarayıcılara uygulanır. Sunucu uygulamalarına uygulanmaz.



CORS'u devre dışı bırakma

- CORS'u anlamayan insanlar **"Access-Control-Allow-Origin:*** yaparak CORS'u devre dışı bırakma eğiliminde olabilirler.
  - "*" burada bütün origin'ler geçerlidir demek.
- Bu hassas verilerin olmadığı "sadece okunabilir" (read-only) servisler için problem yaratmayabilir.
- Ancak aksi durumlarda saldırıya açık hale gelmenize sebep olacaktır.

SameSite Cookie

- Cookie'ler için, Secure ve HttpOnly'nin yanında bir diğer seçenek de SameSite'dır.
- 2016 yılında Chrome tarafından tanıtılmıştır.
  - Ardından pek çok diğer tarayıcı desteklemeye başlamıştır.
- CSRF ataklarını önlemek için eklenmiştir.
  - Şu ana kadar söz ettiğimiz pek çok sorunu engellemektedir.

- **None:** Yalnızca secure olarak işaretlenmiş Cookie'leri CSR içinde gönderir.
  - HTTPS ile
- **Lax:** CSR istekleri engeller ancak GET'in a yönlendirmelerine izin verir.
- **Strict:** Aynı origin hariç bütün istekleri engeller.

Lax kullanımı

- Neden Strict ile her şey bloklanıp güvenli kalınmıyor?
- Bir kişinin kendi web sayfasından sizin web sayfanıza <a> linki verdiğini varsayın.
- Bu türde bir a bağlantısına tıklayan kullanıcının oturum açmışsa oturum açmış olarak devam etmesini ve login sayfasına yönlenmemesini istiyoruz.
- Bu yüzden Lax güvenlik ve kullanılabilirlik arasında iyi bir denge sunmaktadır.
    - Ancak sunucunun GET istekleriyle ilgili bir yan etki barındırmaması gerekmektedir.

2020 Yılında Büyük Değişiklik

- Eğer SameSite eksik olursa Chrome bunu Lax olarak varsayar.
  - Diğer büyük tarayıcılar da aynı şeyi yaptılar/yapıyorlar.
- Bu oldukça büyük bir değişikliktir.
  - Açıkça belirtilmediği sürece CSR varsayılan olarak engellenir.
  - Bu da web'i daha güvenli hale getirmiştir.
- Problem 1: Bunu desteklemeyen tarayıcıların hala desteklenmeye ihtiyacı vardır.
- Problem 2: Bu, tümü aynı kimlik doğrulama cookie'sini kullanan cross-origin isteklerine dayalı web sitelerini bozabilir

Son söz

- Güvenlik oldukça karmaşık bir konudur.
- Ne yazık ki pek çok üniversite eğitimi ve araştırmaları bu konuyu kapsamıyor. Kapsayanlar da oldukça yüzeyseller.
- Sonuç: Ne hakkında konuştuklarına dair hiçbir fikri olmayan kişiler tarafından yazılmış çok sayıda çevrimiçi kaynak
- Öneri: bu soruna karşı dikkatli olun ve güvenlik hakkında okurken körü körüne güvenmeyin (bu sunum da dahil olmak üzere)

2-Kullanıcı Girdilerini Düzenleme, Temizleme

HTML Form Verisi

- HTML formlarında veri nasıl gönderilir?
- HTTP POST isteklerinde payload yapısı nasıldır?
- JSON? {"username":"foo","password":123}
- XML?

x-www-form-urlencoded

- HTML form girdileri gibi text tabanlı veriler için kullanılır.
  - Dosya yükleme gibi binary veriler için multipart/form-data kullanılabilir.
- HTML özelliklerinin bir parçası olan eski bir formattır.
- Her bir eleman <name>=<value> olarak gösterilir ve & işareti ile ayrılır.
- Örnek: username=foo&password=123

Peki değer ”=” veya ”&” içeriyorsa?

- Örnek: password: ”123&bar=7”
- Sonuç: username=foo&password=123&bar=7
- bar=7 değeri 3. bir parametre olarak algılanacaktır.

Çözüm: Özel Kodlama

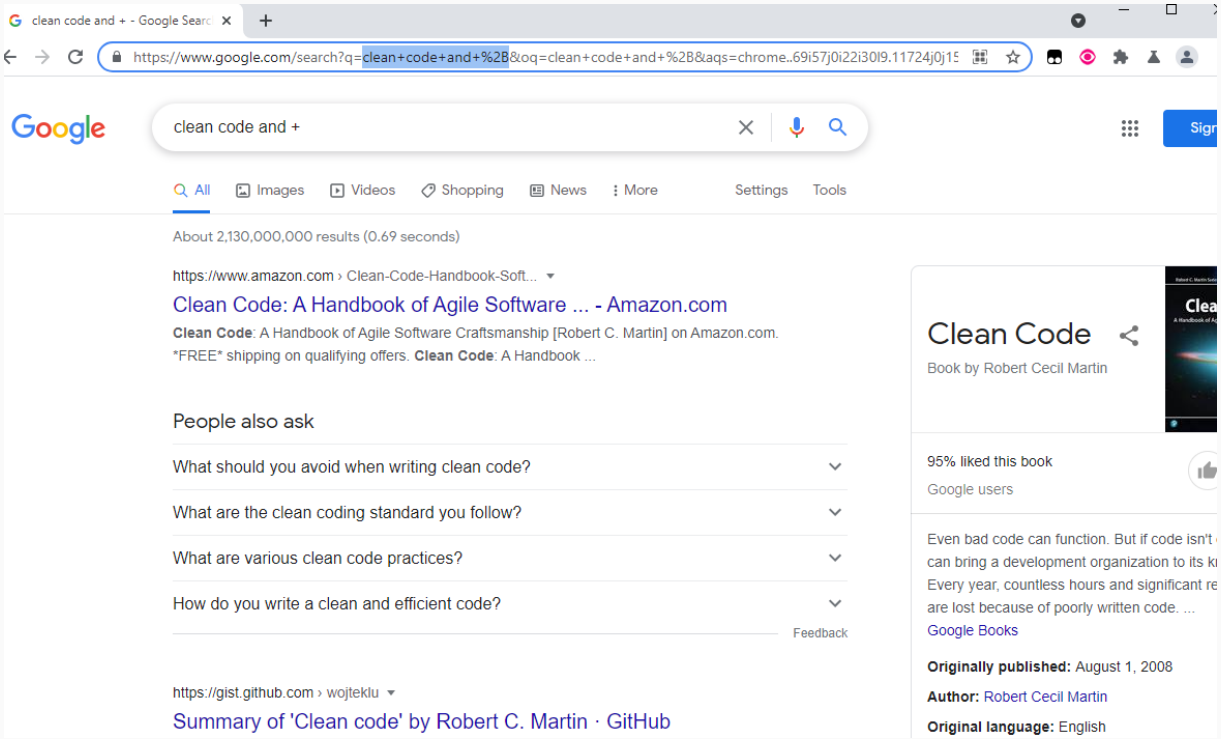
- ”“, ”-“, ”.“, ”_“, 0-9, a-z, A-Z değerleri aynı kalır.
- Space (” “) değeri ”+” olur.
- Geri kalanlar %HH'a döner. H burada hexadecimal sayıdır ve ilgili harfin karakter koduna denk gelir (varsayılan olarak UTF-8).
- ”123&bar=7” değeri ”123%26bar%3D7” olur.
- $\%26 = (2 * 16) + 6 = 38$, & karakterinin ASCI karşılığı.
- $\%3D = (3 * 16) + 13 = 61 =$ karakterinin ASCI karşılığı

Peki % değerler içerisinde bulunursa kodlama bozulur mu?

- Eğer password="%3D" gibi bir değer olursa,
- "%253D" olarak kodlanacaktı. 37 ise % değerinin ASCI kodudur ve bir problem çıkmayacaktır.
  - %25=(2*16)+5=37

URL ve Query Parametreleri

- URL içerisindeki query parametreleri <key>=<value> olarak verilmektedir ve & ile ayrılmaktadır.
- Eğer içerisinde & veya = gibi özel karakterler bulunsa ne olurdu?
- Bunlarda da %HH gibi escape karakterleri kullanılır.
  - Tek bir fark: boş karakter "+" ile değiştirilirken, + karakteri %2B (+ karakterinin ASCI karşılığı: 43) ile değiştirilir.

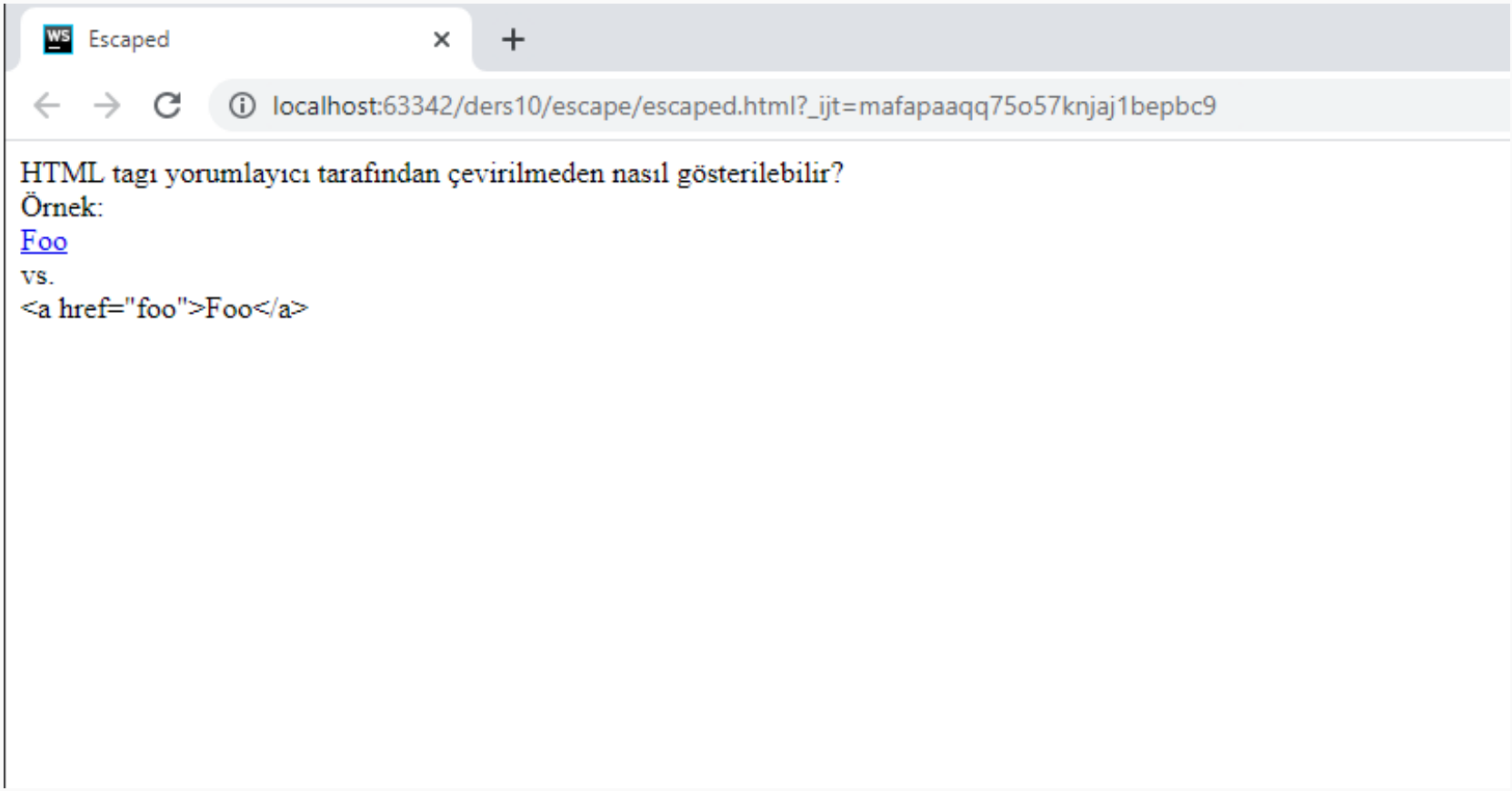


- Google'da "clean code and +" ifadesini aradığımızda,
- Sunucu şu query parametresi ile arama yapacaktır: clean+code+and+%2B
- Boşluklar + ile + karakteri ise %2B ile değiştirildi.

Text Dönüştürme

- Metinler (text) HTML, XML, JSON, x-www-form-urlencoded gibi farklı formatlarda temsil edilmektedir.
- Bu tür formatlarda da özel karakterler bulunmaktadır.
- Input text'ler bu özel karakterleri barındırmamalıdır.
- Bu yüzden bir dönüşüme (escaped) ihtiyaç bulunmaktadır.
  - &, %26'ya ve =, %3D'ye gibi.

Peki HTML'de?


Escaped
localhost:63342/ders10/escape/escaped.html?_ijt=mafapaaqq75o57knjaj1bepbc9
HTML tagı yorumlayıcı tarafından çevirilmeden nasıl gösterilebilir?
Örnek:
Foo
vs.
Foo

HTML/XML Escape

- **&** ardından ad veya kod, ";" karakteri ile sonlandırma.
- **"** $\rightarrow$ "
- **&** $\rightarrow$ &
- **'** $\rightarrow$ '
- **<** $\rightarrow$ $<$
- **>** $\rightarrow$ $>$
- Bunlar en yaygın kullanılanları

3-XSS

Kullanıcı İçeriği

- HTML sayfalarındaki görüntülenen ve kullanıcı tarafından yazılan girdilerdir (HTML form gibi)
  - Chat, Forumlar
  - Arama ekranları
  - vs.
- Ayrıca URL içerisindeki sorgu parametreleri bir saldırgan tarafından oluşturulmuş ise bir kullanıcı girdisi olarak adlandırılır.
  - Örnek: x değeri HTML tarafından görüntüleniyor ise www.foo.com?x=10
  - Hatırlatma: Saldırganlar sosyal mühendislik teknikleri ile kullanıcıların çeşitli linklere tıklamasını isteyebilir.
- Bir sisteme girdi olarak verilen kullanıcı içeriğiyle ilgili en önemli kural nedir?

ASLA KULLANICI GİRDİLERİNE GÜVENME

ASLA KULLANICI GİRDİLERİNE GÜVENME ASLA KULLANICI GİRDİLERİNE GÜVENME ASLA KULLANICI GİRDİLERİNE GÜVENME ASLA KULLANICI GİRDİLERİNE GÜVENME ASLA KULLANICI GİRDİLERİNE GÜVENME ASLA KULLANICI GİRDİLERİNE GÜVENME

- ASLA KULLANICI GİRDİLERİNE GÜVENME
- ASLA KULLANICI GİRDİLERİNE GÜVENME
- ASLA KULLANICI GİRDİLERİNE GÜVENME
- ASLA KULLANICI GİRDİLERİNE GÜVENME
- ASLA KULLANICI GİRDİLERİNE GÜVENME

Chat XSS

Ad:

Veli50

Mesaj:

Gönder

Ali49: Merhaba :)

Veli50: Merhaba :) Bu sisteme XSS atak yapabilirim, göstereyim mi? :)

Ali49: O ne ola ki?

Veli50: Bak (:

Chat XSS

Ad:

Veli50

Mesaj:

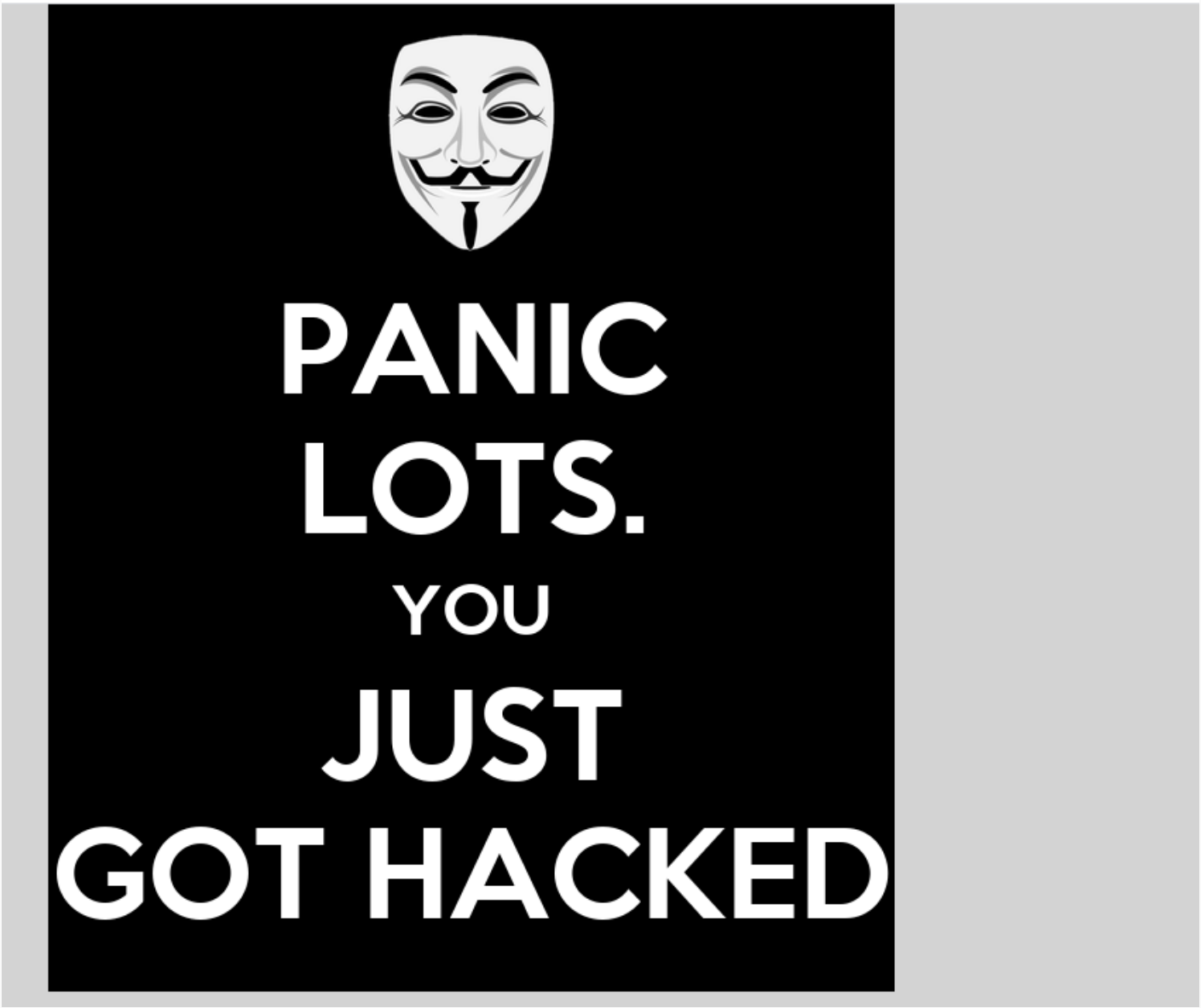
Gönder

Ali49: Merhaba :)

Veli50: Merhaba :) Bu sisteme XSS atak yapabilirim, göstereyim mi? :)

Ali49: O ne ola ki?

Veli50: Bak (:


PANIC
LOTS.
YOU
JUST
GOT HACKED

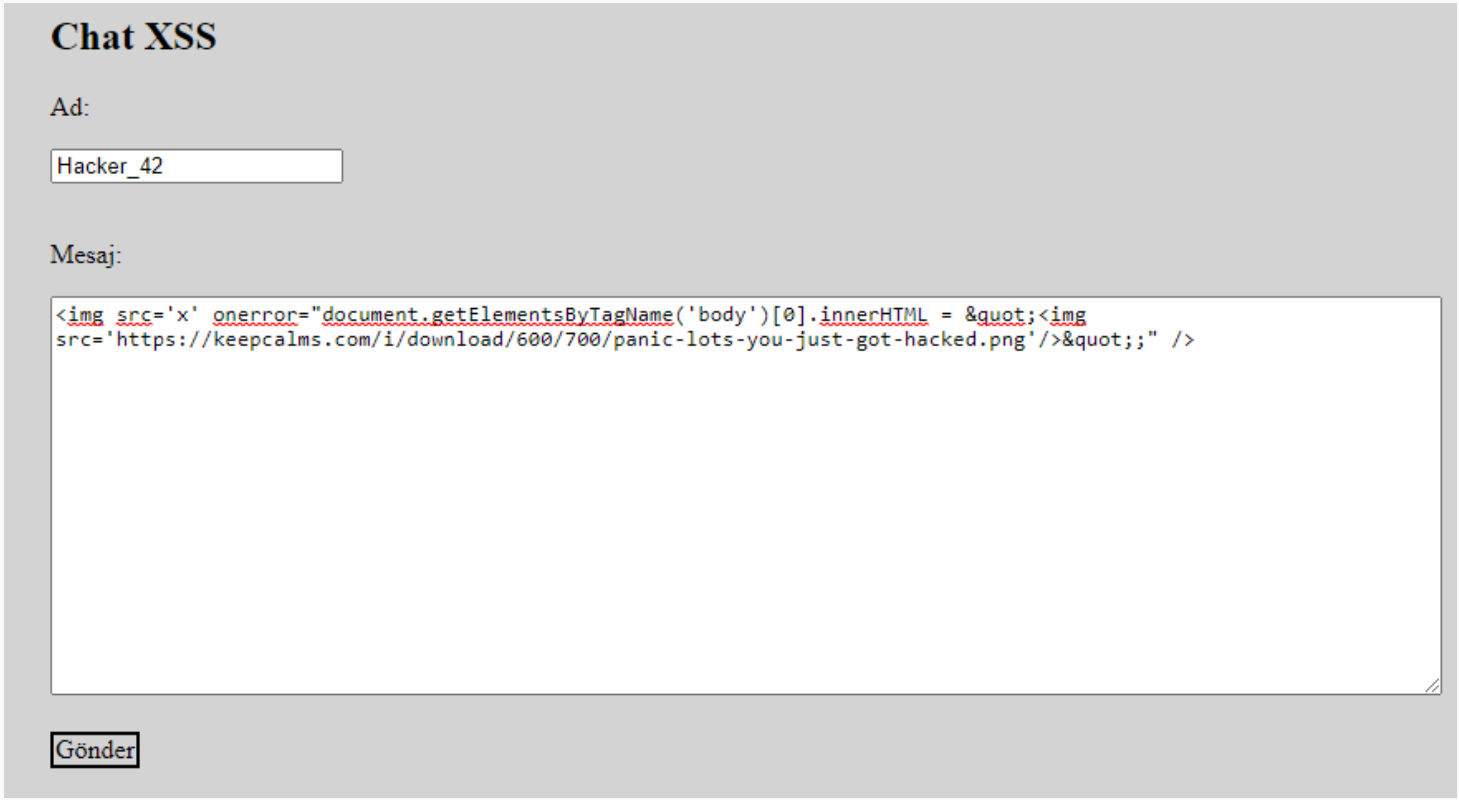
Peki problem nedir?

```
let msgDiv = "<div>";
for(let i=0; i<messages.length; i++){
    const m = messages[i];
    //UYARI: XSS'e sebep olabilir!!!
    msgDiv += "<p>" + m.author + ": " + m.text + "</p>";
}

msgDiv += "</div>";

document.getElementById( elementId: "msgDiv").innerHTML = msgDiv;
}
```

Böyle bir mesaj gönderildiğinde...



String Birleştirme

- msgDiv += "<p>" + author+":"+text+"< /p>;
- Kullanıcıdan veri geldiğinde ASLA string ifadeleri HTML oluşturmak için birleştirmemelisiniz.
- Eğer veride escape işlemi yapılmazsa HTML tagı gibi algılanabilir.
- Bununla birlikte JS kodları bile çalıştırılabilir ve bir sayfada yapılmak istenen her şey gerçekleştirilebilir.
- Örnek: text="<script>...< /script>"

Cross-site Scripting (XSS)

- Zararlı bir JS kodunun web sayfasına sızdırıldığı atak tipidir.
- Web'te güvenlik problemine sebep olan en yaygın türlerden biridir.
- Kullanıcı girdilerinin temizlenmemesinden faydalanılarak gerçekleştirilir.
- XSS mevcut sayfaya JS kodu eklediği için oldukça zararlıdır ve CORS bu noktada işe yaramamaktadır.

Tarayıcı Güvenliği

- Pek çok tarayıcı dinamik olarak eklenmiş <script> tagını çalıştırmamaktadır.
  - Örnek: innerHTML ile HTML içeriğinin değiştirilmesi gibi.
- Ancak bu da işe yarayan bir önlem değildir. Hemen çalıştırılan HTML tagları ile JS kodu rahatlıkla çalıştırılabilmektedir.
- <img src='bulunmayanBirURL' onerror='...JS kodu...'>

Çözüm?

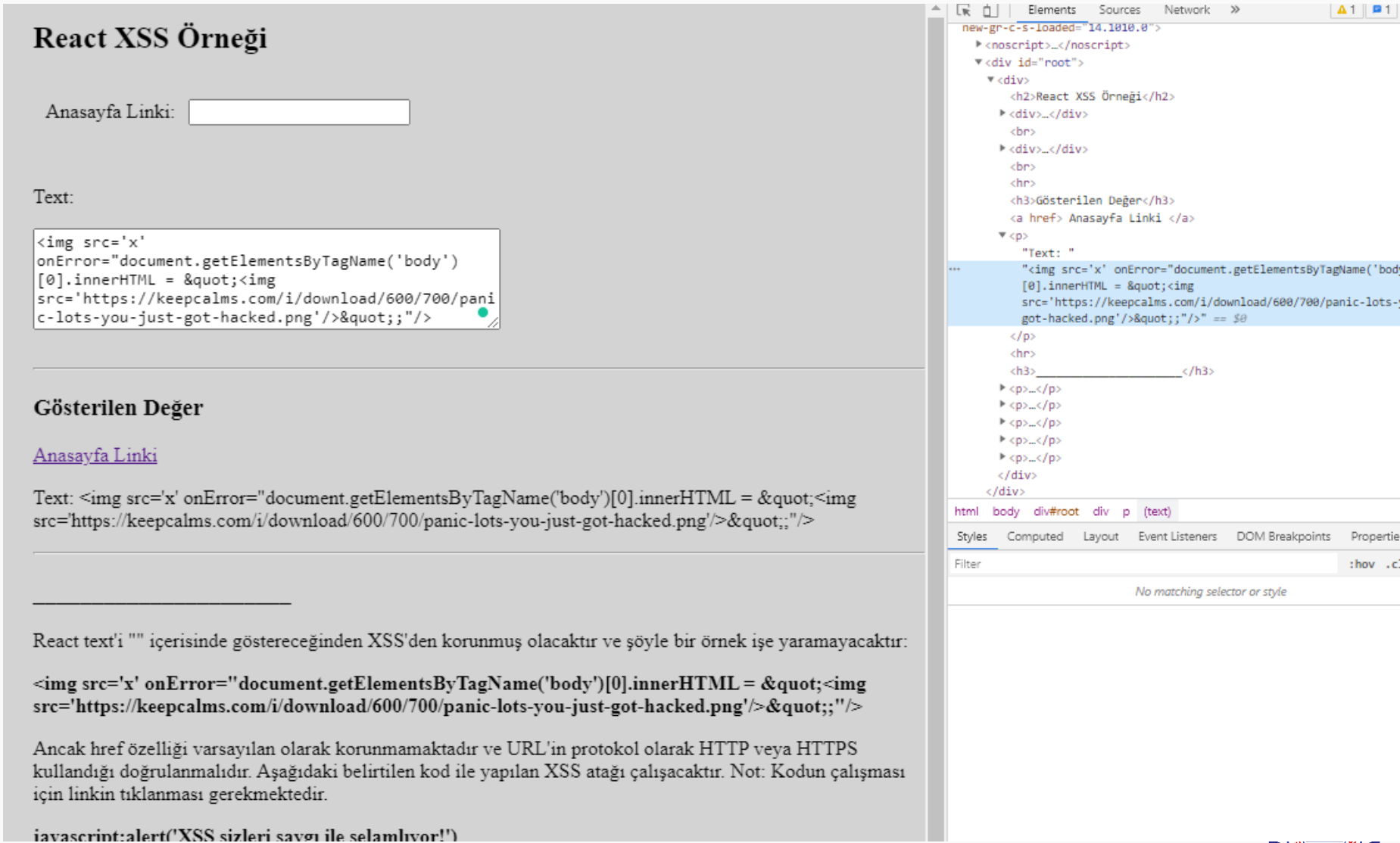
- Kullanıcı girişlerini kullanmadan önce temizlenmesi gerekmektedir.
- Bu eylem hem istemci (JS) hem de sunucu (Java, PHP, C# vs.) tarafında gerçekleştirilmelidir.
- Pek çok uç nokta bulunmaktadır bu yüzden mevcut kütüphaneleri kullanmalısınız.
  - Bu kütüphaneler programlama dili ve framework'e bağlıdır.
  - Kendi temizleme fonksiyonunuzu yazmamalısınız!

4-XSS ve React

React ile Temizleme

- XSS o kadar büyük problemdir ki pek çok kütüphane/framework HTML manipülasyonu yaparken girdileri de temizler.
- Örnek: JSX'i ele aldığımızda <p>Text:{this.state.userInput}< /p>
- ve userInput içeriği de <a> olursa,
- React <a> olarak çevrir.
- Böylelikle herhangi < veya > değerleri HTML tagı olarak algılanamaz.

Böyle bir mesaj gönderildiğinde...



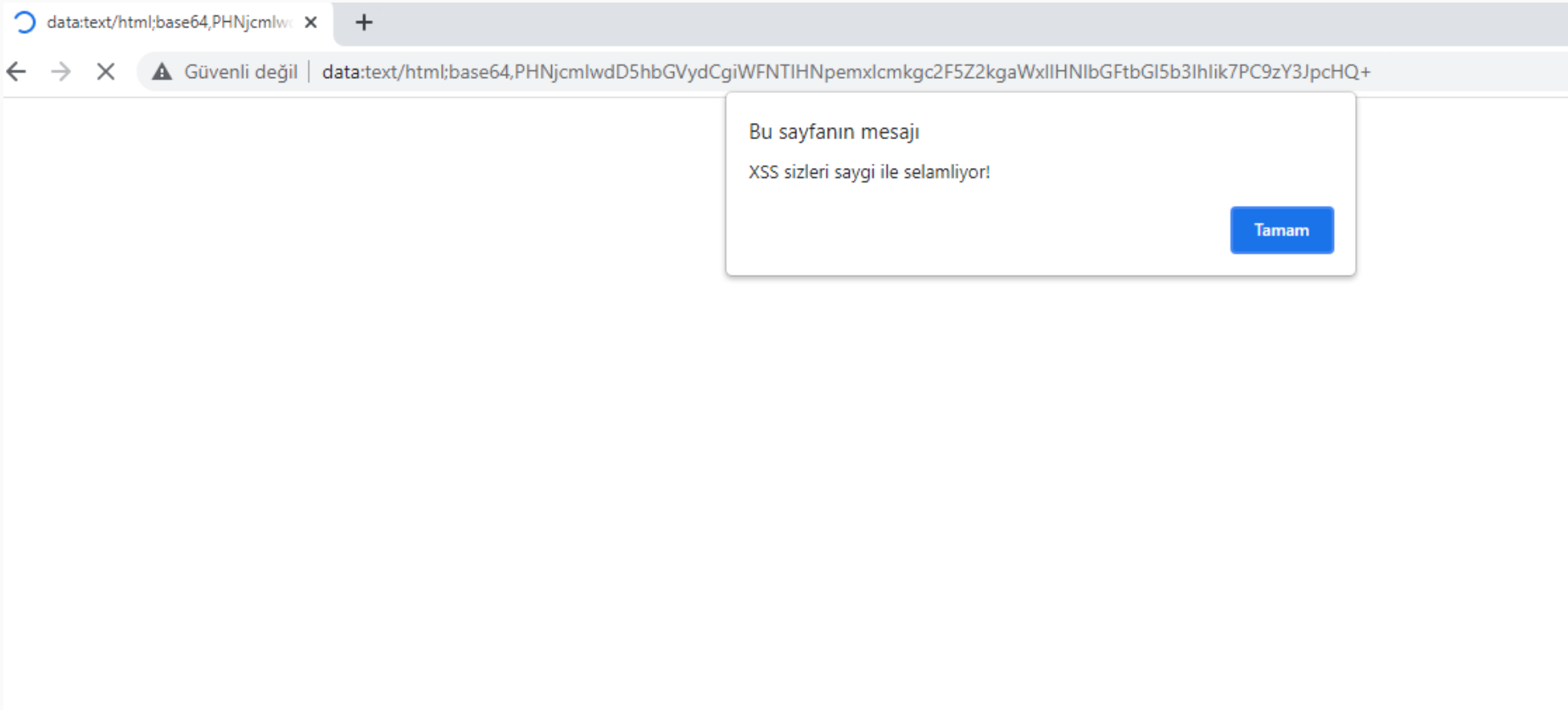
React kullanarak tamamen güvende misiniz?

HAYIR

Temizleme

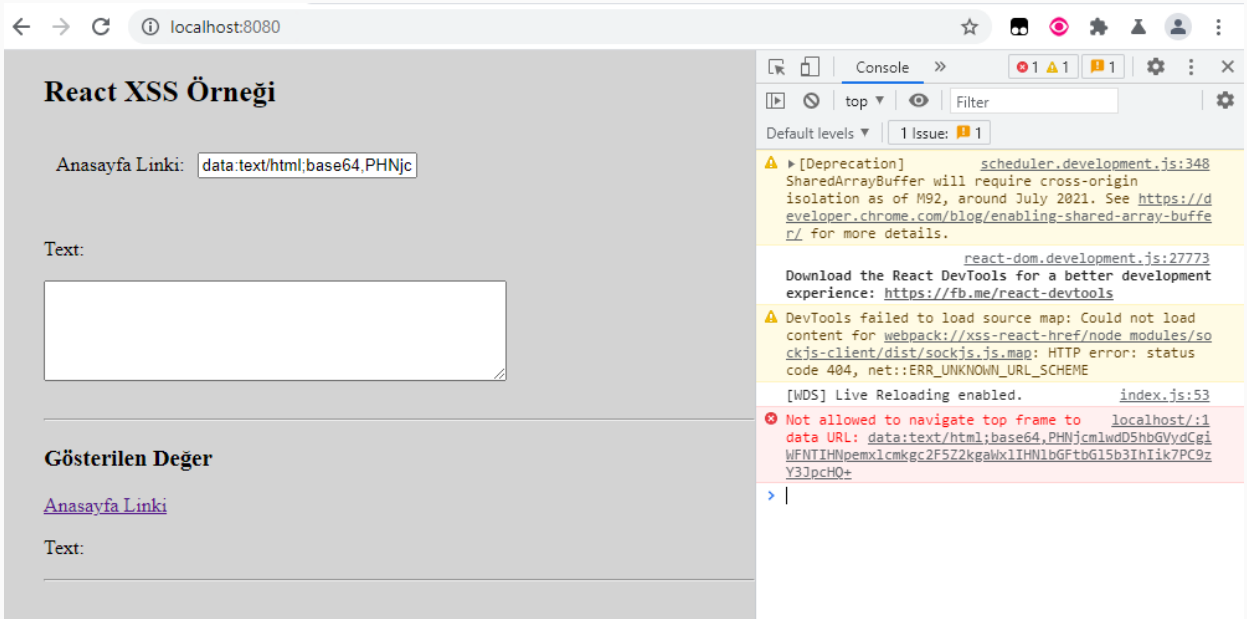
- URL'ler ele alındığında, kullanıcı girdileri manuel olarak temizlenmelidir.
  - Örnek: "javascript:" protokolüne izin verilmemelidir.
  - 2021 notu: React'ın gelecek versiyonlarında bu kaldırılacaktır.
- Genel bir kural olarak, kendi temizleme işlevlerinizi yazmamalı, bunun yerine mevcut kütüphaneleri kullanmalısınız.
  - Ancak siz yazacaksanız da whitelist kullanın. Örnek: Sadece "javascript:"'i reddedip geri kalan her şeye izin vermek yerine "http:" veya "https:" protokollerine izin verilirken geri kalan her şey engellenmelidir.
- Çünkü şöyle bir girdi de gelebilir ve bunu engellemek gerekir: Sadece
  - data:text/html;base64,PHNjcmlwdD5hbGVydCgiWFNTIHNpemxlcmk

Böyle bir mesaj gönderildiğinde...



Böyle bir mesaj gönderildiğinde...

- "Top frame" adı verilen bu özellik pek çok tarayıcıda güvenlik sebebiyle 2017 yılında kaldırıldı.
- Ancak kendi temizleme fonksiyonunuzu neden yazmamanız gerektiğini gösteren oldukça güzel bir örnektir. Çok fazla tuhaf uç örnekler bulunmaktadır.
- Eğlenceli örnekler için:
- https://owasp.org/www-community/xss-filter-evasion-cheatsheet



Kullanıcılar vs Geliştiriciler

- Bir kullanıcı olarak: **HER ZAMAN TARAYICININ EN GÜNCEL SÜRÜMÜNÜ KULLANIN**
  - Sizi pek çok bilinen saldırıdan koruyacaktır.
- Geliştirici olarak: Pek çok müşteriniz eski tarayıcıları kullanıyor olabilir.
  - Bu yüzden kullanıcılarınızı korumak için mevcut tarayıcılar engelliyor olsa da uygulamalarınıza pek çok ekstra güvenlik katmanı eklemeniz gerekir.

- 2021: Internet Explorer'ın hala %0.71 pazar payı bulunmaktadır.
  - Türkiye'de %0.56
  - Çok eski yıllardan bir masala göre 2015 yılında **Edge** ile değiştirilecekti.
- 2019: Edge Chromium ile tekrardan yazıldı.
- 2021 yılında Edge:
  - Dünyada: %3.39
  - Türkiye'de: %1.37
- Ayrıntılı bilgi: https://gs.statcounter.com