# Cross Site Request Forgery (CSRF) Saldırılarını Önleme

## (İleri Düzey)

harezmi bilişim çözümleri

# Önleme Yolları

- **Synchronizer Token Pattern**
  - Sunucu her web requesti için benzersiz ve gizli bir token üretir
  - Bu token'ı kullanıcıya gönderir
  - Kullanıcı bir sonraki request'i bu token ile birlikte gerçekleştirir
  - Token sunucu tarafında her seferinde kontrol edilir
  - Yoğun AJAX kullanan web sitelerinde implement edilmesi zor olabilir

# Önleme Yolları

- **Cookie-To-Header Pattern**
  - JavaScript'i yoğun olarak kullanan web uygulamalarında tercih edilir
  - Temeli JS'in same origin politikasına dayanır
  - Web sitesi login sırasında bir cookie içerisinde oturum boyunca sabit bir token gönderir
  - JS request'leri bu token ile gerçekleştirilir
  - Farklı bir sitenin JS'leri same origin politikasından dolayı bu cookie'ye erişemez ve token'ı elde edemez

# Önleme Yolları

- **İlave Kimliklendirme ve CAPTCHA**
  - Sunucu kritik bazı işlemlerden önce kullanıcıdan tekrar kimliklendirme yapmasını isteyebilir
  - Ya da sunucu kritik bu işlemlerin gerçekleştirilmesinde CAPTCHA kullanabilir
  - Örneğin, para transferi veya şifre değiştirme işlemleri öncesi

# JSON ve AJAX İsteklerinde CSRF Token

- JSON kullanılan durumlarda CSRF token'ın **http request parametresi** olarak gönderilmesi mümkün değildir
- Bunun yerine **HTTP header**'da gönderilebilir
- Öncelikle CSRF token'ı **HTTP meta tag**'leri ile sayfanın içerisine yerleştirilir
- Ardından AJAX request'inde de **meta tag'lerinden** CSRF token alınarak HTTP header'ları ile sunucuya iletilebilir

# JSON ve AJAX İsteklerinde CSRF Token

```
<html>
  <head>
    <meta name="_csrf" content="${_csrf.token}"/>
    <!-- default header name is X-CSRF-TOKEN -->
    <meta name="_csrf_header" content="${_csrf.headerName}"/>
    <!-- ... -->
  </head>
  <!-- ... →
</html>
```

```
$(function () {
  var token = $("meta[name='_csrf']").attr("content");
  var header = $("meta[name='_csrf_header']").attr("content");
  $(document).ajaxSend(function(e, xhr, options) {
    xhr.setRequestHeader(header, token);
  });
});
```

# Dikkat Edilmesi Gereken Noktalar: Logout

- Eğer mutlaka **link** kullanılması gerekiyor ise **javascript** ile bu gerçekleştirilebilir
- Javascript'in **disable** edildiği tarayıcılarda bu sorun olabilir

# Dikkat Edilmesi Gereken Noktalar: Timeout

- CsrfToken HttpSession'da tutulduğu için timeout söz konusu olduğu vakit **InvalidCsrfTokenException** fırlatılacaktır
- Default durumda da bu **access denied hatası**na yol açacaktır
- Bazen bu durum **kullanıcılar için yanıltıcı** olabilir
- **AccessDeniedHandler** üzerinden bu özelleştirilebilir

# Dikkat Edilmesi Gereken Noktalar: Multipart

- Multipart file upload işlemlerinin CSRF protection'a takılmaması için **MultiPartFilter**'ın Spring Security Filter'dan önce tanımlanması gerekir
- Böyle bir durumda herhangi bir kullanıcı sunucuda **temp dosya** oluşturabilir
- Ancak bu çoğu sunucu için problem değildir
- Sadece yetkili kullanıcılar uygulama tarafından process edilen dosya submit edebileceklerdir

# Dikkat Edilmesi Gereken Noktalar: Multipart

- Eğer temp dosya oluşturulması kabul edilemez bir durum ise CSRF token'ın **action url'e eklenmesi** ile bu problem aşılabilir
- Ancak bu **CSRF token'ın leak etmesi**ne yol açabilir
- Çünkü token bilgisi **Request URI**'da yer alacaktır

```
<form action="./upload?${_csrf.parameterName}=${_csrf.token}"
method="post" enctype="multipart/form-data">
</form>
```

# Dikkat Edilmesi Gereken Noktalar:HiddenHttpMethodFilter

- **HiddenHttpMethodFilter** ile tarayıcı üzerinden **PUT** ve **DELETE** metotlarında web request'i yapılması sağlanmaktadır
- Aslında **POST** metodunda çalışan bir filter'dır
- Ancak **CSRF** ile birlikte bazen **probleme neden olabilmekte**dir
- Bu filter'ın da Spring Security Filter'dan **önce** tanımlanması sağlıklı olacaktır

# CSRF ve XSS

- **Cross Site Scripting** (XSS) ise farklı bir açıktır
- XSS, diğer kullanıcılar tarafından erişilebilen web sayfaları üzerinde saldırganların **client side scriptler inject etmesi**dir
- Saldırılar hedef kullanıcıların makinasında istenmeyen **scriptleri çalıştırarak** gerçekleştirilir
- XSS açıkları **CSRF önlemlerinin bypass edilmesi**ne neden olabilir

# İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com