


Testing for Cross Site Request Forgery (Çapraz Site için Test Sahte Talep)

Summary (Özet)

Cross-Site Request Forgery (CSRF), bir son kullanıcıyı şu anda doğrulandıkları bir web uygulamasında istenmeyen eylemler yürütmeye zorlayan bir saldırdır. Biraz sosyal mühendislik yardımı ile (e-posta veya sohbet yoluyla bir bağlantı göndermek gibi), bir saldırgan bir web uygulamasının kullanıcılarını saldırganın seçtiği eylemleri yürütmeye zorlayabilir. Başarılı bir CSRF sürücüsü, normal bir kullanıcıyı hedef aldığı anda son kullanıcı verilerini ve işleyişini tehlikeye atabilir. Hedeflenen son kullanıcı yönetici hesabı ise, bir CSRF saldırısı tüm web uygulamasını tehlikeye atabilir.

CSRF şuna güveniyor:

1. Çerezler ve HTTP kimlik doğrulama bilgileri gibi oturumla ilgili bilgilerin işlenmesine ilişkin web tarayıcısı davranışı.
2. Bir saldırganın geçerli web uygulama URL'leri, istekleri veya işlevselliği hakkında bilgi sahibi.
3. Uygulama oturumu yönetimi yalnızca tarayıcı tarafından bilinen bilgilere dayanmaktadır.
4. Varlığı bir HTTP[S] kaynağına anında erişmeye neden olan HTML etiketlerinin var olması; örneğin resim etiketi  . .

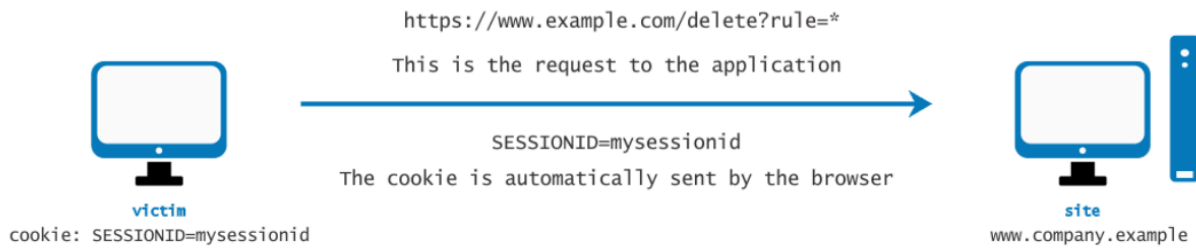
Puan 1, 2 ve 3, kırılabilirliğin mevcut olması için gereklidir, 4. nokta ise fiili sömürüyü kolaylaştırır, ancak kesinlikle gerekli değildir.

1. Tarayıcılar bir kullanıcı oturumunu tanımlamak için kullanılan bilgileri otomatik olarak gönderir. *Sitenin* bir web uygulamasına ev sahipliği yapan bir site olduğunu varsayalım ve kullanıcı *kurbanı* sadece *siteye doğrulanmıştır*. Buna karşılık, *site kurban tarafından* mağdur *tarafından* gönderilen talepleri

kurbanın doğrulanmış oturumuna ait olarak tanımlayan bir çerez gönderir. Tarayıcı çerez setini *sitesiteye* göre aldıktan sonra, *siteye* yönlendirilen başka taleplerle birlikte otomatik olarak gönderir.

2. Uygulama URL'lerde oturumla ilgili bilgileri kullanmazsa, uygulama URL'leri, parametreleri ve meşru değerleri tanımlanabilir. Bu, kod analizi ile veya uygulamaya erişerek ve HTML veya JavaScript'e gömülü form ve URL'leri not alarak gerçekleştirilebilir.
3. "Taracı tarafından bilinen", tarayıcı tarafından depolanan ve daha sonra bu kimlik doğrulamasını isteyen bir uygulama alanına yönelik her talepte bulunan çerezler veya HTTP tabanlı kimlik doğrulama bilgileri (Temel Kimlik Doğrulama ve form tabanlı kimlik doğrulaması gibi) gibi bilgileri ifade eder. Daha sonra tartışılan güvenlik açıkları, bir kullanıcı oturumunu tanımlamak için tamamen bu tür bilgilere dayanan uygulamalar için geçerlidir.

Basitlik uğruna, GET'e erişilebilen URL'leri düşünün (tartışma POST istekleri için de geçerli olsa da). *Kurban* zaten kendilerini doğruladıysa, başka bir talep göndermek, çerezin otomatik olarak gönderilmesine neden olur. Aşağıdaki rakam, kullanıcının bir uygulamaya eriştiğini göstermektedir. www.example.com . .



Şekil 4.6.5-1: Oturum Sürüşü

GET talebi, kullanıcı tarafından birkaç farklı şekilde gönderilebilir:

- Web uygulamasının kullanılması
- URL'yi doğrudan tarayıcıya yazmak
- URL'ye işaret eden harici bir bağlantıyı takip etmek

Bu çağrılar uygulama tarafından ayırt edilemez. Özellikle üçüncüsü oldukça tehlikeli olabilir. Bir bağlantının gerçek özelliklerini gizleyebilecek bir dizi teknik ve güvenlik açığı vardır. Bağlantı bir e-posta mesajına gömülüp, kullanıcının cezbedildiği kötü amaçlı bir web sitesinde görünebilir veya üçüncü taraf (başka bir web sitesi veya HTML e-postası gibi) tarafından barındırılan içerikte görünür ve uygulamanın bir kaynağına işaret edebilir. Kullanıcı bağlantıya tıklarsa, sitedeki web uygulaması tarafından zaten doğrulandıklarından, tarayıcı kimlik doğrulama bilgileri (oturum kimliği çerezi) eşliğinde web uygulamasına bir GET talebi yayınlayacaktır. Bu, kullanıcının beklemediği web uygulamasında gerçekleştirilen geçerli bir işlemle sonuçlanır; örneğin, bir web bankacılığı uygulamasına bir fon aktarımı.



Örneğin bir etiket kullanarak `img` Yukarıdaki 4. noktada belirtildiği gibi, kullanıcının belirli bir bağlantıyı takip etmesi bile gerekli değildir. Saldırganın kullanıcıya aşağıdaki (aşırı basitleştirilmiş) HTML içeren bir sayfaya atıfta bulunan bir URL'yi ziyaret etmeleri için onları teşvik eden bir e-posta gönderdiğini varsayalım.

```
<html>
  <body>
...

...
  </body>
</html>
```

Tarayıcı bu sayfayı görüntülediğinde, belirtilen sıfır-dersiyon (bu nedenle görünmez) görüntüyü görüntülemeye çalışacaktır. `https://www.company.example` Aynı zamanda. Bu, *sitede* barındırılan web uygulamasına otomatik olarak gönderilen bir talep ile sonuçlanır. Resim URL'sinin uygun bir görüntüye atıfta bulunmaması önemli değildir, çünkü varlığı isteği tetikleyecektir. `action` içinde belirtilen `src` Her neyse, saha. Bu, görüntü indirmenin tarayıcıda devre dışı bırakılmaması şartıyla gerçekleşir. Çoğu tarayıcı, resim indirmeleri devre dışı bırakılmamıştır, çünkü çoğu web uygulamasını kullanılabilirliğin ötesinde sakat bırakacaktır.

Buradaki sorun şu:

- Sayfadaki HTML etiketleri otomatik HTTP istek yürütme ile sonuçlanır ( Bunlardan biri olmak da).
- Tarayıcının, referans alınan kaynağı söylemenin bir yolu yok  Meşru bir imaj değildir.
- Görüntü yüklemesi, iddia edilen görüntü kaynağının bulunduğu yerden bağımsız olarak gerçekleşir, yani, form ve görüntünün kendisinin aynı ana bilgisayarda veya hatta aynı alanda bulunması gerekmez.





Web uygulamasıyla ilgisi olmayan HTML içeriğinin uygulamadaki bileşenlere ve tarayıcının otomatik olarak uygulamaya yönelik geçerli bir istek oluşturması gerçeği, bu tür bir saldırıya izin verir.

Saldırganın uygulama işlevselliği ile etkileşime girmesi imkansız hale getirilmedikçe bu davranışı yasaklamanın bir yolu yoktur.

Entegre posta / tarayıcı ortamlarında, görüntü referansını içeren bir e-posta iletisi görüntülemek, talebin web uygulamasına ilgili tarayıcı çerezi ile yürütülmesine neden olacaktır. E-posta mesajları aşağıdaki gibi görünüşte geçerli görüntü URL'lerine başvurabilir:

```

```

Bu örnekte,  Saldırgan tarafından kontrol edilen bir yer. Yönlendirme mekanizması kullanılarak, kötü amaçlı site kullanabilir  Kurbanı yönlendirmek için  ve tetikleyin  . .

Çerezler, bu tür bir güvenlik açığında yer alan tek örnek değildir. Oturum bilgileri tamamen tarayıcı tarafından sağlanan web uygulamaları da savunmasızdır. Bu, yalnızca HTTP kimlik doğrulama mekanizmalarına dayanan uygulamaları içerir, çünkü kimlik doğrulama bilgileri tarayıcı tarafından bilinir ve her talep üzerine otomatik olarak gönderilir. Bu, sadece bir kez gerçekleşen ve oturumla ilgili bilgilerin bir türünü, genellikle bir çerez oluşturan form tabanlı kimlik doğrulamasını içermez.

Diyelim ki kurbanın bir güvenlik duvarı web yönetim konsoluna kayıtlı olduğunu varsayalım. Giriş yapmak için, bir kullanıcının kendilerini doğrulaması gerekir ve oturum bilgileri bir çerezde saklanır.

Güvenlik duvarı web yönetim konsolunun, doğrulanmış bir kullanıcının sayısal kimliği tarafından belirtilen bir kuralı veya kullanıcının belirtmesi durumunda yapılandırmadaki tüm kuralları silmesine izin veren bir işleve sahip olduğunu varsayalım. * (gerçekte tehlikeli bir özellik, ancak daha ilginç bir örnek teşkil eden bir özellik). Sil sayfası daha sonra gösterilir. Diyelim ki bu form – basitlik uğruna – bir alma talebinde bulunur. Kuralı bir numarayı silmek için:

```
https://[target]/fwmgmt/delete?rule=1
```

Tüm kuralları silmek için:

```
https://[target]/fwmgmt/delete?rule=*
```

Bu örnek kasıtlı olarak saftır, ancak CSRF'nin tehlikelerini basit bir şekilde gösterir.



Şekil 4.6.5-2: Oturum Sürüş Güvenlik Duvarı Yönetimi

Yukarıdaki rakamda resmedilen formu kullanarak, değeri girmek * ve Silin düğmesini tıklamak aşağıdaki ALAN talebini gönderecektir:

```
https://www.company.example/fwmgmt/delete?rule=*
```


Bu tüm güvenlik duvarı kurallarını siler.



Şekil 4.6.5-3: Oturum Sürüş Güvenlik Duvarı Yönetimi 2

Kullanıcı ayrıca URL'yi manuel olarak göndererek aynı sonuçları elde etmiş olabilir:

[https://\[target\]/fwmgmt/delete?rule=*](https://[target]/fwmgmt/delete?rule=*)

Veya doğrudan veya bir yönlendirme yoluyla yukarıdaki URL'ye işaret eden bir bağlantıyı takip ederek. Ya da yine gömülü bir HTML sayfasına erişerek  Etiket aynı URL'ye işaret eder.

Tüm bu durumlarda, kullanıcı şu anda güvenlik duvarı yönetim uygulamasına giriş yapılırsa, talep başarılı olur ve güvenlik duvarının yapılandırmasını değiştirir. Hassas uygulamaları hedef alan ve otomatik açık artırma teklifleri, para transferleri, siparişler, kritik yazılım bileşenlerinin yapılandırılmasını değiştirmek vb.

İlginç bir şey, bu güvenlik açıklarının bir güvenlik duvarının arkasında uygulanabileceğidir; yani saldırıya uğrayan bağlantının doğrudan saldırgan tarafından değil, kurban tarafından ulaşılabilmesi yeterlidir. Özellikle, herhangi bir intranet web sunucusu olabilir; örneğin, daha önce bahsedilen güvenlik duvarı yönetim senaryosunda, İnternet'e maruz kalması muhtemel değildir.

Kendiliğinden savunmasız uygulamalar, yani hem saldırı vektörü hem de hedef olarak kullanılan uygulamalar (web posta uygulamaları gibi), işleri daha da kötüleştirir. Kullanıcılar e-posta mesajlarını okuduklarında oturum açıldığından, bu tür savunmasız bir uygulama, saldırganların mesajları silmek veya kurbandan kaynaklanmış gibi görünen mesajlar göndermek gibi eylemler gerçekleştirmelerine izin verebilir.

Test Objectives (Test Hedefleri)

- Kullanıcı tarafından başlatılmayan bir kullanıcı adına isteklerin başlatılmasının mümkün olup olmadığını belirleyin.

How to Test (Nasıl Test Edilir)

Oturum yönetiminin savunmasız olup olmadığını belirlemek için başvuruyu denetleyin. Oturum yönetimi yalnızca istemci tarafındaki değerlere (tarayıcıya sunulan bilgiler) güveniyorsa, uygulama savunmasızdır. "Mcli-side değerleri", çerezler ve HTTP kimlik doğrulama kimlik bilgilerini (Temel Kimlik Doğrulama ve diğer HTTP kimlik doğrulaması; uygulama düzeyinde bir kimlik doğrulama olan form tabanlı kimlik doğrulamayı değil) ifade eder.

HTTP GET istekleri ile erişilebilen kaynaklar kolayca savunmasızdır, ancak POST istekleri JavaScript ile otomatikleştirilebilir ve savunmasız da savunmasızdır; Bu nedenle, POST kullanımı tek başına CSRF güvenlik açıklarının oluşumunu düzeltmek için yeterli değildir.

POST durumunda aşağıdaki örnek kullanılabilir.

1. Aşağıda gösterilene benzer bir HTML sayfası oluşturun
2. HTML'yi kötü amaçlı veya üçüncü taraf bir sitede barındırın
3. Sayfanın bağlantısını mağdurlara gönderin ve tıklamaya teşvik edin.

```
<html>
<body onload='document.CSRF.submit()>

<form action='http://targetWebsite/Authenticate.jsp' method='POST' name='CSF
  <input type='hidden' name='name' value='Hacked'>
  <input type='hidden' name='password' value='Hacked'>
</form>

</body>
</html>
```

Geliştiricilerin JSON'ı sunucu iletişimine tarayıcı için kullandıkları web uygulamaları durumunda, kendi kendine teslim formları olan JSON formatında sorgu parametrelerinin olmadığı gerçeğiyle ilgili

bir sorun ortaya çıkabilir. Bu davayı atlamak için, CSRF'yi sömürmek için gizli girdiler de dahil olmak üzere JSON yükleriyle kendini teslim eden bir form kullanabiliriz. Kodlama türünü değiştirmek zorunda kalacağız (`enctype`) Tona `text/plain` Yükün olduğu gibi teslim edildiğinden emin olmak için. Sömürü kodu aşağıdaki gibi görünecektir:

```
<html>
<body>
<script>history.pushState('', '', '/')</script>
<form action='http://victimsite.com' method='POST' enctype='text/plain'>
  <input type='hidden' name='{"name":"hacked","password":"hacked","padding":"=something"}'>
  <input type='submit' value='Submit request' />
</form>
</body>
</html>
```

POST talebi şöyle olacaktır:

```
POST / HTTP/1.1
Host: victimsite.com
Content-Type: text/plain

{"name":"hacked","password":"hacked","padding":"=something"}
```

Bu veriler POST isteği olarak gönderildiğinde, sunucu isim ve şifre alanlarını memnuniyetle kabul eder ve ihtiyaç duymadığı gibi isim dolgusu olanı görmezden gelecektir.

Remediation (Düzeltilme)

- Önleme önlemleri için OWASP CSRF Önleme Hile Sayfasına bakın.

Tools (Araçlar)

- OWASP ZAP
- CSRF Test Cihazı

- Pinata-csrf-tool

Referances (Referanslar)

- Peter W: "Cross-Site Request Forgeries"
- Thomas Schreiber: "Oturum Sürüş"
- En eski bilinen yazı
- Çapraz Site Talebi Sahte SSS
- Cross Site Talebiyle İlgili En Çok İhmal Edilen Bir Gerçek (CSRF)
- Çoklu POST KSSF
- SANS Pen Test Webcast: Çok POST XSRF üzerinden komple uygulama röle