1.1.1 İstemci DOM XSS (Client DOM XSS) (CWE-79)

Açıklık Önem Derecesi: Yüksek

Açıklığın Etkisi: Oturum çalma, oltalama, arayüz tahrifi

Açıklığın Barındıran Dosyalar/Satırlar:

Proje Dosyası/Dosya Adı	Satır Numarası

Açıklığın Açıklaması:

Uygulamalar kullanıcı girdilerini, sunucu taraftaki veri tabanlarını veya sunucu taraftaki diğer kaynakları kullanarak güvensiz veri içeren web sayfalar inşa edebilirler. Bu gibi durumlarda güvensiz veri web sayfanın bir parçası olarak sayfanın HTML'ine doğrudan gömülür. Güvensiz verinin bu şekilde yereldeki web sayfanın HTML DOM yapısını manipüle etmesine DOM XSS (CWE-79) açıklığı adı verilir.

Başarılı bir DOM XSS sömürüsü saldırganlara beklenen / umulan html çıktının değiştirildiği zararlı script'ler ekleme ve web sayfayı yeniden yazma imkanı verir. Yani saldırganlara web sayfalarına ek HTML düğümleri, CSS kodları, keyfi javascript kodları ve üçüncü taraf kodlar için referanslar ekleme imkanı verir. Saldırganlar böylece kullanıcı parolalarını, kredi kart detayları gibi kişisel verileri çalabilirler, kurbanları yanlış bilgilendirme ve yönlendirme yapabilirler veya kurbanları zararlı program çalıştırmaya sevk edebilirler.

Client DOM XSS ve Stored DOM XSS Hakkında:

Client DOM XSS kullanıcı girdisinin istemci tarafta dönüp dolaşıp yine istemci tarafta javascript sink'leri ile DOM yapısına yerleşmesi ile web tarayıcıda tetiklenmesine denir. Stored DOM XSS ise kullanıcı girdisinin sunucu taraflı bir kaynaktan (örn; veri tabanından) istemci tarafa gelmesi ve javascript sink'ler ile DOM yapısına yerleşmesi ile web tarayıcıda tetiklenmesine denir.

Stored XSS ve Stored DOM XSS Hakkında:

Stored XSS ile Stored DOM XSS'in farkı Stored XSS'de kullanıcı girdisi sunucu tarafta bir kaynağa kaydolup istemci tarafa geri yansıdıktan sonra html context'te tetiklenir, Stored DOM XSS ise kullanıcı girdisi yine sunucu tarafta bir kaynağa kaydolup istemci tarafa geri yansıdıktan sonra javascript context'te tetiklenir.

Bu açıklığı somutlaştırmak için örneklere yer verilmiştir:

Javascript - Güvensiz Kod Bloğu (1):

```
// DOM XSS in img Attribute

var url = new URL(window.location.href);
var imgsrc = url.searchParams.get("imageLocation");

// The payload "imageLocation=1 onerror=alert(1)"
// will result in an alert prompt, demonstrating XSS
document.write('<img id="myImage" src=' + imgsrc +' ></img>');
```

Bu güvensiz örnekte URL GET parametresi imageLocation'ın değeri <img etiketinin src özelliğine yerleştirilmektedir. Herhangi bir filtreleme arada olmadığından client dom xss zafiyeti meydana gelecektir.

Javascript - Güvensiz Kod Bloğu (2):

```
// DOM XSS When Using "eval()" to Parse JSON in Javascript

var url = new URL(window.location.href);
var val = url.searchParams.get("val");
var json = `[{"val": "${val}"}]`;

// The payload json=","a":alert(1),"b":" will
// result in an alert prompt, demonstrating XSS
var obj = eval(json);
```

Javascript - Güvenli Kod Bloğu (2):

```
// Replacing "eval()" with "JSON.parse()" to Avoid XSS

var url = new URL(window.location.href);
var val = url.searchParams.get("val");
var json = `[{"val": "${val}"}]`;

// JSON.parse() does not eval JS code
var obj =  JSON.parse(json);
```

Javascript - Güvensiz Kod Bloğu (2)'de URL GET parametresi val'in değeri çekilmektedir ve bu değer json verisine eklenip eval() ile parse edilmektedir. eval() fonksiyonu json verisi içerisindeki javascript kodlarını çalıştırıp çıktılarını kodların yerine koyan bir işleve sahip

olduğundan güvensiz olacaktır. Çünkü bu sayede GET parametresi üzerinden DOM XSS yapılabilecektir. Güvenli örnekte ise bu sorunu gidermek maksadıyla JSON.parse() fonksiyonu kullanılmıştır. Böylece değerlendirmeye alınan json verisindeki olası yer alabilecek herhangi bir javascript kodun çalışması önlenecektir.

Javascript - Güvensiz Kod Bloğu (3):

```
// DOM XSS in iFrame "src" Attribute

var url = new URL(window.location.href);
var iframeLocation = url.searchParams.get("iframeLocation");

// The payload "iframeLocation=javascript:alert(1)" will
// result in an alert prompt, demonstrating XSS. This is
// also vulnerable to open redirection.
document.getElementById("myFrame").src = iframeLocation;
```

Javascript - Güvenli Kod Bloğu (3):

```
// Prepending iFrame "src" Attribute to Prevent Malicious URI Schemes

var url = new URL(window.location.href);
var iframeLocation = url.searchParams.get("iframeLocation");

// Prepending iframeLocation prevents changing
// the URI scheme to "javascript:", mitigating XSS
document.getElementById("myFrame").src = "/example/"+iframeLocation;
```

Javascript - Güvensiz Kod Bloğu (3)'de URL GET parametresi iframeLocation'ın değeri çekilmektedir ve bir iframe düğümünün src özelliğine eklenmektedir. Bu işleyişte herhangi bir filtreleme olmadığından javascript: anahtar kelimesi girdi olarak iframeLocation parametresine girildiğinde dom xss saldırısı düzenlenebilir. Güvenli örnekte ise URL GET parametresi iframeLocation'ın değeri, /example/ dizin yolunun sonrasına eklemlendiğinden herhangi bir javascript: anahtar kelimesi eklendiği senaryoda dom xss saldırısı işe yaramayacaktır.

Kurum uygulamada İstemci DOM XSS (CWE-79) açıklığı tespit edilmiştir.

::::::::::::BULGU::::::::::::::::

Açıklığın Önlemi:

Bu açıklığı gidermek için tavsiye edilen önlemler şu şekildedir:

- Tüm dinamik veriler kaynağına bakılmaksızın çıktıya gömülmeden önce tamamen encode'lanmalıdır.
- Encode'lama bağlam ilişkili olmalıdır. Örneğin;
    - HTML içerik için HTML encode'lama
    - Attribute değerlerine çıktılama için HTML attribute encode'lama
    - Sunucu taraflı javascript için Javascript encode'lama
- Çıktıyı encode'lamak için kullanılan framework'ün temin ettiği encode'lama fonksiyonelliğine sahip bilinen güvenlik kütüphaneleri tercih edilebilir.
- Yalnızca uygulamanın kaynakları izinli sayılacak şekilde beyaz liste kuralı Content Security Policy uygulanmalıdır.
- Ekstra koruma katmanı olması adına tüm girdiler kaynağı neresi olursa olsun doğrulamadan geçirilmelidir (not: Doğrulama encode'lamanın alternatifi bir teknik değildir. Tamamlayıcısıdır). Bu doğrulama belirli yapıdaki girdileri reddetme işlemi yerine sadece belirli yapıya uyanların kabul edildiği şeklinde olmalıdır. Yani whitelist (beyaz liste) kullanılmalıdır. Siyah liste (blacklist) önlemi kullanılmamalıdır. Bir girdiyi doğrulamada şunlar kontrol edilebilir:
    - Veri türü (int, string, float, byte, … v.b. tipte mi geliyor kontrolü)
    - Boyutu (x KB, y MB,…v.b. boyutta mı geliyor kontrolü)
    - Aralığı (Veri ölçülebilir bir sayısal değerse aralık sınırlarını aşıyor mu kontrolü)
    - Formatı (çözümlenebilir şifreli, tek yönlü şifreli, açık metin, … v.b. biçimde mi geliyor kontrolü)
    - Beklenen Değerlerden Biri Olup Olmadığı (Whitelist'te tanımlı desenlerden biri mi geliyor kontrolü)
- HTTP yanıt başlığı Content-Type açıkça karakter encode'lama (charset) tanımlı olmalıdır.
- Derinlikli defans gereği oturum çerezleri üzerinde HttpOnly bayrağı etkinleştirilmelidir. Bu önlem olası başarılı XSS sömürülerinde çerezlerin çalınmasını önleyecektir.

Referanslar:

1. https://cwe.mitre.org/data/definitions/582.html
2. https://vulncat.fortify.com/en/detail?category=Unsafe%20Mobile%20Code&subcategory=Unsafe%20Array%20Declaration#Java%2fJSP