Dom XSS

XSS zafiyetleri üçe ayrılırlar. Bunlar Stored Xss, Reflected Xss ve Dom Xss' tirler. Stored Xss girilen input'un filtrelenmeden hedef web uygulamasının veritabanına kaydolduğu ve oradan çekilerek web uygulaması içerisinde kullanıldığı durumda ortaya çıkar. Reflected Xss girilen input'un filtrelenmeden olduğu gibi web uygulamasının bir noktasına yansıtıldığı durumda ortaya çıkar. Dom Xss ise yine girilen input'un filtrelenmeden olduğu gibi web uygulamasının bir noktasına yansıtıldığı durumda ortaya çıkar.

Görüldüğü üzere Dom Xss zafiyeti Reflected xss zafiyetine benzer. İkisi de girilen input filtrelenmeden web sayfasına yansıtıldığı zaman ortaya çıkarlar. Fakat Dom Xss de girilen input reflected xss'in aksine server'a gitmez. Onun yerine gönderilen input client tarafındaki javascript kodlamalarına gider, işlenir ve tarayıcıya yansır. Böylece eğer hedef web uygulamasında client tarafında filtreleme yapılmadıysa girilen xss kodları ekrana olduğu gibi yansıyacak ve çalışacaktır. O nedenle Dom Xss için filtrelemeler client tarafında yapılmalıdır. Reflected xss'de ise girilen input server'a gidip dönerek ekrana yansıtıldığı için server tarafında filtreleme yapılmalıdır.

Dom Xss zafiyeti görüldüğü üzere istemci taraflı filtreleme eksikliğinden meydana gelmektedir. Dom XSS saldırısının nasıl yapıldığına gelecek olursak Dom Xss saldırısı hedef web uygulamasının url'si sonuna hash (#) karakteri konarak gerçekleştirilir. Url sonundaki # karakteri sayesinde url enter'landığında sunucuya talep gitmeyecektir. Dolayısıyla url'deki # karakteri sonrasına javascript kodu girersek ve eğer hedef web uygulaması mevcut url'yi olduğu gibi ekrana yansıtan bir javascript kodlamasına sahipse bu durumda url'ye koyduğumuz xss kodları ekrana yansıyacaktır ve Dom Xss saldırısı gerçekleşmiş olacaktır.

Uygulama

(+) Dom XSS'in temel bir uygulaması gösterilmiştir. Bu uygulama eski tarayıcılarda işe yaradığından test edilememiştir. Ancak yeni tarayıcılarda başarıyla tatbiki Uygulama 2'de gösterilmiştir.

Dom Xss zafiyeti aşağıdaki gibi bir kodlamaya sahip web sayfalarında meydana gelir:

/var/www/DOM XSS Uygulaması/dom_xss_vulnerability.html Kaynak Kod:

```
<script>
        document.write("<b>Current URL</b> : " + document.baseURI);
</script>
```

Yukarıdaki kodlamamayı yorumlayacak olursak zafiyete sahip web uygulamasının kaynak kodundaki javascript kodu document.baseURI ile adres çubuğundaki mevcut url'yi çekmektedir. document.write() ile de mevcut url'yi ekrana basmaktadır. Görüldüğü üzere çekilen url hiçbir denetlemeye tabi tutulmadan olduğu gibi ekrana basılmaktadır. Şimdi zafiyete sahip bu sayfayı tarayıcıda görüntüleyelim.

Tarayıcı Adres Çubuğu

http://localhost/DOM XSS Uygulaması/dom_xss_vulnerability.html

Output:

Current URL : http://localhost/DOM XSS Uygulaması/dom_xss_vulnerability.html

Görüldüğü üzere url olduğu gibi ekrana basılmıştır. Şimdi url'ye # işareti ile beraber xss kodları ekleyelim.

Tarayıcı Adres Çubuğu

http://localhost/DOM XSS Uygulaması/dom_xss_vulnerability.html#<script>alert('xss')</script>

Output:

Current URL : http://localhost/DOM XSS Uygulaması/dom_xss_vulnerability.html#

Yukarıdaki işlem sonrası tarayıcı ekranına alert popup'ı gelecektir. Böylece hedef web uygulamasının zafiyete sahip olduğunu görmüş olacağız. Hedef web uyglamasında istediğimiz javascript kodunu çalıştırabileceğimizi anladıktan sonra buradan hareketle url sonuna çerez çalan kodları koyabilir, url'yi farklı bir görünümle kurbanlara gönderebilir ve kurbanların linke tıklaması sonrası zafiyete sahip web uygulamasındaki çerezlerini çalmayı umabiliriz.

Not:

Xss saldırıları Chrome tarayıcısında Xss_Auditor tarafından önlendiğinden chrome'da işe yaramayacaktır. Firefox 40 ve üzeri tarayıcılarda ise xss saldırıları yapılabilmesine rağmen Dom Xss saldırıları yapılamadığından yukarıdaki saldırı sonucu alert popup'ı gelmeyecektir. Firefox 40 ve üzerinde dom xss saldırısı yapma demosu bir sonraki başlıkta anlatılacaktır.

Uygulama 2

(+) Birebir denenmiştir ve başarıyla uygulanmıştır.

Dom Xss zafiyeti aşağıdaki gibi bir kodlamaya sahip web sayfalarında meydana gelir:

/var/www/DOM XSS Uygulaması/dom_xss_vulnerability.html Kaynak Kod:

```
<script>
	document.write("<b>Current URL</b> : " + document.baseURI);
</script>
```

Yukarıdaki kodlamamayı yorumlayacak olursak zafiyete sahip web uygulamasının kaynak kodundaki javascript kodu document.baseURI ile adres çubuğundaki mevcut url'yi çekmektedir. document.write() ile de mevcut url'yi ekrana basmaktadır. Görüldüğü üzere çekilen url hiçbir denetlemeye tabi tutulmadan olduğu gibi ekrana basılmaktadır. Şimdi zafiyete sahip bu sayfayı tarayıcıda görüntüleyelim.

Tarayıcı Adres Çubuğu (Firefox 40 ve üzeri için)

http://localhost/DOM XSS Uygulaması/dom_xss_vulnerability.html

Output:

Current URL : http://localhost/DOM XSS Uygulaması/dom_xss_vulnerability.html

Görüldüğü üzere url olduğu gibi ekrana basılmıştır. Şimdi url'ye # işareti ile beraber xss kodları da ekleyelim. Ardından enter ile saldırıyı başlatalım.

Tarayıcı Adres Çubuğu (Firefox 40 ve üzeri)

http://localhost/DOM XSS Uygulaması/dom_xss_vulnerability.html#<script>alert('xss')</script>

Output:

Current URL : http://localhost/DOM XSS Uygulaması/dom_xss_vulnerability.html#%3Cscript%3Ealert(%27xss%27)%3C/script%3E

Görüldüğü üzere url ekrana encode edilmiş halde yansımıştır. Demek ki document.baseURI mevcut url'yi encode'lanmış halde çekti ve decode etmeden document.write()'a yerleştirdi. Bu nedenle script kodumuz encode halde yansıtıldığından tarayıcı tarafından çalıştırılamadı. Sonuç olarak Dom Xss saldırısı başarısız oldu. Firefox 40 ve üzeri tarayıcılarda Dom Xss saldırılarının işe yarayabilmesi

için hedef web sayfaları şu yapıda olmalıdırlar.

/var/www/DOM XSS Uygulaması/dom_xss_vulnerability.html :

```
<script>
        document.write("<b>Current URL</b> : " +
        decodeURIComponent(document.baseURI));
</script>
```

Görüldüğü üzere adres çubuğundan çekilen url decode edilerek document.write()'a yerleştirilmektedir ve o şekilde ekrana basılmaktadır. Şimdi bu javascript kodlamasına sahip web sayfasına yine Dom Xss saldırısı düzenleyelim.

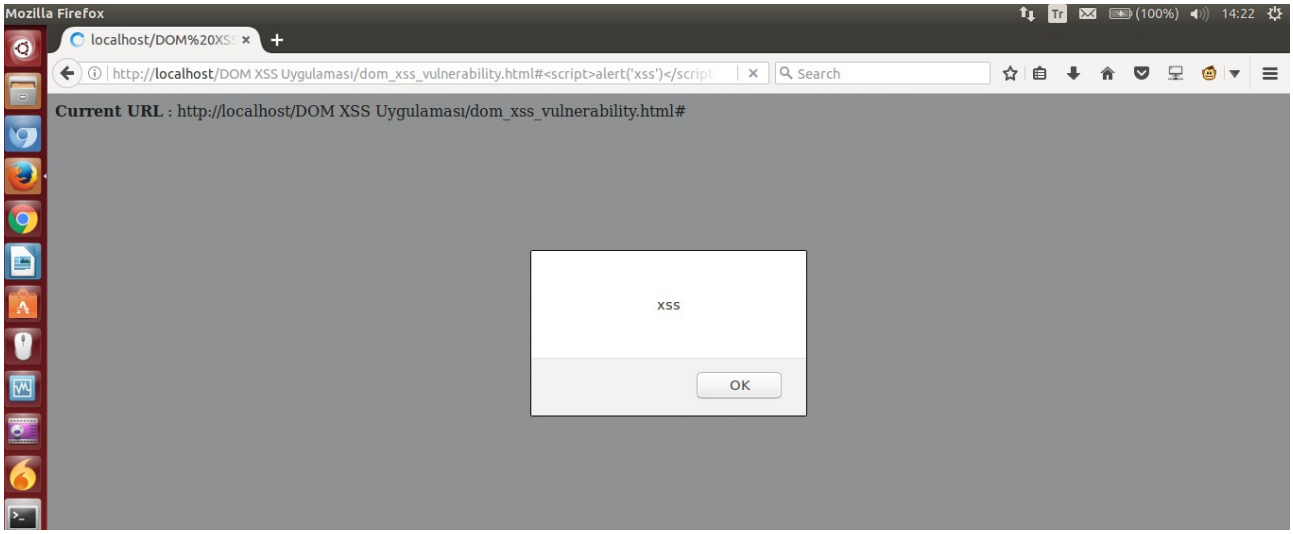
Tarayıcı Adres Çubuğu (Firefox 40 ve üzeri)

http://localhost/DOM XSS Uygulaması/dom_xss_vulnerability.html#<script>alert('xss')</script>

Output:

Current URL: http://localhost/DOM XSS Uygulaması/dom_xss_vulnerability.html#

Yukarıdaki url enter'landığında çıktıdan da görülebileceği gibi javascript kodları ekranda görünmemektir. Demek ki javascript kodları kaynak koda olması gerektiği gibi yerleşmiştir. Yukarıdaki url enter'landığında aşağıdaki gibi popup ekranı bizi karşılayacaktır.



Görüldüğü üzere adres çubuğundaki url girdiğimiz xss kodu ile beraber istemci tarafında çekildi ve ekrana basıldı. Böylece script kodumuz çalıştı ve ekrana popup geldi. Sonuç olarak hedef web uygulamamızın zafiyete sahip olduğunu anlamış olduk. Şimdi yapılabilecek işlem hedef web uygulamasının url'sine çerez çalan xss kodlarını koymak ve url'yi farklı bir görünüşle kurbanlara göndermektir. Kurbanlar url'ye tıkladıklarında hedef web uygulamasındaki çerezleri saldırgana göndereceklerdir ve böylece kurbanların oturumu devralınabilecektir.

Dom Xss Nasıl Önlenir?

İstemci tarafında yer alan javascript kodları mevcut url'yi çektiklerinde olduğu gibi ekrana yansıtmamalıdırlar. Çektikleri Url'yi denetime tabi tutmalıdırlar. Ancak bu denetleme (veya filtreleme) ile uğraşmak yerine aşağıdaki gibi pratik bir yapı da kullanılabilir.

samplePage.html:

```
<b>Current URL:</b> <span id="content"></span>

<script>
        document.getElementById("content").textContent = document.baseURI;
</script>
```

Yukarıda görüldüğü üzere adres çubuğundan çekilen url textContent property'si ile belirlenen tag'lar arasına yerleştirilmektedir. TextContent property'si aldığı değeri diğer property'lerin aksine encode'layarak tag'ların arasına koyduğundan olası Dom Xss saldırısının önüne geçilmiş olacaktır. Sonuç olarak filtreleme yerine javascript'in kendi property'si textContent ile de Dom Xss'e karşı güvenlik sağlanabilmektedir.

Ekstra

document.baseURI gibi başka url çeken property'ler de vardır. Bunlar;

```
document.baseURI        // Mevcut düğümün temsil ettiği sayfanın adres çubuğundaki
                        // adresini döndürür. (örn; node.baseURI)

document.URL            // Yalnızca html sayfalarının adres çubuğundaki adresini
                        // döndürür

document.documentURI    // Tüm sayfaların adres çubuğundaki adresini döndürür.

document.referrer       // Mevcut sayfanın adres çubuğundaki adresini döndürür.

location.href           // Yalın halde iken adres çubuğundaki adresi döndürür (Elle
                        // set edildiğinde ise yönlendirme yapar)
```

(Daha fazlası için bkz. https://www.netsparker.com/blog/web-security/dom-based-cross-site-scripting-vulnerability/)

Tarayıcıdaki bir düğüme veri koymanın birden fazla yolu vardır. Örn;

```
document.write(".....")                    // Kök düğüm content'ine string konur.

(element).innerHTML = "…....."             // Belirlenen düğümün content'ine string
                                           // konur.

(element).src = "…....."                   // Belirlenen düğümün src attribute'una
                                           // string konur.
```

Dolayısıyla url çeken bir property ve düğüme o değeri koyan property kombinasyonlarından herhangi biri kullanıldığında eğer denetleme yapılmamışsa Dom Xss saldırısı gerçekleşebilir.

Not:

document.write("…"), ...innerHTML gibi başka javascript sink'leri de vardır. Listesi için bkz.

https://portswigger.net/web-security/cross-site-scripting/dom-based#which-sinks-can-lead-to-dom-xss-vulnerabilities

Kaynaklar

https://security.stackexchange.com/questions/51994/what-is-the-difference-between-ordinary-xss-and-dom-xss-vulnerabilities

https://www.netsparker.com/blog/web-security/dom-based-cross-site-scripting-vulnerability/

https://www.owasp.org/index.php/DOM_Based_XSS

https://github.com/OWASP/railsgoat/issues/229

https://www.w3schools.com/jsref/prop_doc_baseuri.asp

https://www.w3schools.com/jsref/prop_doc_url.asp

https://www.w3schools.com/jsref/prop_document_documenturi.asp

https://www.w3schools.com/jsref/prop_loc_href.asp

https://portswigger.net/web-security/cross-site-scripting/dom-based#which-sinks-can-lead-to-dom-xss-vulnerabilities