PHP Kod Güvenliği ve Yanlış Bilinenler

Canberk Bolat, Aralık 2010, WGT E-Dergi 7. Sayı

PHP uygulamaları ve kullanılabilecek güvenlik yöntemlerine dair birçok şey yazılıp çiziliyor fakat yanlış bilinenler ve bu yanlış kullanımlar her şeye rağmen güvenlik zafiyetlerine sebebiyet veriyor. Bu yazıda bilinen güvenlik fonksiyonlarının yanlış kullanımlarına ve exploit edilmelerine değineceğim. İlk olarak XSS zafiyetine karşı alınan önlemlerle ilgili bir şeyler anlatmaya çalışayım.

XSS (Cross-site Scripting)

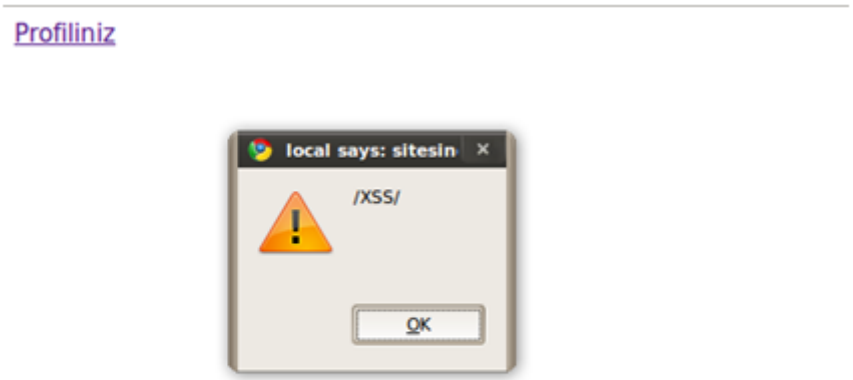
XSS zafiyetlerine karşı bilinen ve incelediğim kaynak kodlarında en çok gördüğüm şey PHP fonksiyonlarından **htmlspecialchars**. Htmlspecialchars fonksiyonu özel HTML karakterlerini fonksiyona belirtilen veri içerisinde arar ve bulduklarını kodlayarak HTML olarak yorumlanmalarını engeller. Fakat htmlspecialchars sizi her zaman korumaz. Örnek bir hatalı kullanım için aşağıdaki PHP kod alıntısına bakabilirsiniz;

```
…
$user = htmlspecialchars($_GET["user"]);
echo "<a href=xss.php?user=$user title=\"$user\">Profiliniz</a>";
…
```

Görüldüğü üzere htmlspecialchars kullanılmış fakat bu uygulamayı koruduğu söylenemez. Çünkü siz burada aslında sadece kendinizi avutmuş olursunuz bu kod ile. Neden mi? user parametresine canberk%20onmouseover=alert(/XSS/) değerini girelim ve sonucu görelim.

```
<a href=xss.php?user=canberk onmouseover=alert(/XSS/) title
```

Resimde görüldüğü üzere browser yolladığımız değeri yorumladı ve aşağıdaki resimde göreceğiniz üzere başarıyla onmouseover olayı gerçekleştiğinde alert fonksiyonunu çağırdı.



Peki neden böyle oldu, htmlspecialchars neden korumadı beni derseniz cevabı çok basit, biz gönderdiğimiz değerde herhangi bir html karakteri yollamadık ki denetime takılsın. Eğer tırnak ile aralansaydı link bu XSS'ten bahsedemeyecektik. Maalesef bu hataya düşen birçok uygulama ile karşılaşabiliyoruz.

Yapılan bir diğer hata ise platforma güvenerek kod yazmak. Nasıl olsa platform bana bir şeyler sağlıyor fikriyatı ile magic_quotes_gpc'nin verdiği güven ile kod yazmakta uygulamanın çuvallamasına sebep olabiliyor. Bu örnekte magic_quote_gpc'nin desteğini arkasına almış bir uygulama ile karşı karşıyayız.

```
...
echo "<a href=\"xss.php?user=$user\" title=\"$user\">Profiliniz</a>";
...
```

Tamam her şey çok güzel gözüküyor, çift tırnak kullanmışız, magic_quotes_gpc'de açık, daha ne isteriz? Tabikide saldırganın canberk"%20onmouseover=alert(/XSS/)%09 değerini girmesini isteriz.

```
<a href="xss.php?user=canberk\" onmouseover=alert(/XSS-2/)       " title='
```

Yollanan değerde tek çift tırnak var o da bir öncekinin kapanmasını sağladı ve magic_quotes_gpc'ye rağmen browser çift tırnak içindeki veriyi xss.php?user=canberk\ olarak algıladı ve yine bize izin verdi. Ferruh Mavituna'nın 5.sayıda yazdığı "Web Uygulamalarında Güvenli Platform Seçimi" yazısını hatırlamakta fayda var. [1]

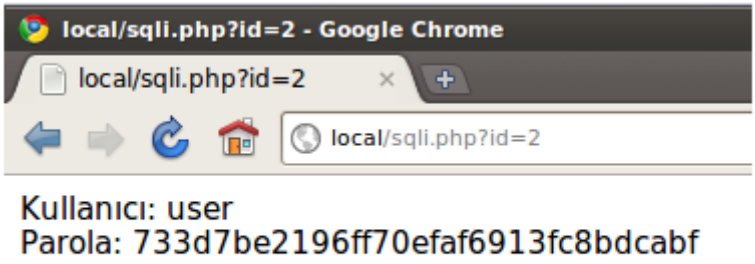
SQL Injection

SQL Injection zafiyetlerine maruz kalan web uygulamalarında karşılaştığım en büyük hatalardan biriside mysql_real_escape_string (mres) fonksiyonuna karşı duyulan inanılmaz güven. İlk XSS örneğinde olduğu gibi sorguya gelen veriyi çift tırnak ile çevrelemeden sadece mres'e güvenerek yoluna devam eden birçok web uygulaması var ve ne zaman bu tip bug'ları avlasam geliştiricisine detaylıca açıklıyorum ve "ama mres fonksiyonundan geçirdim..." gibisinden ortak cevaplar alıyorum. Aşağıdaki örneği bir inceleyelim.

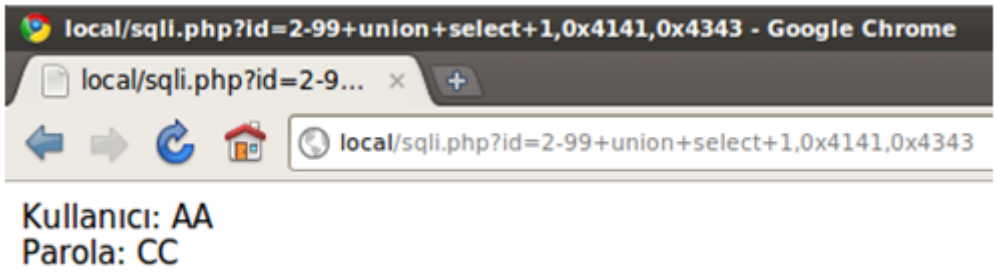
```
...
$id = mysql_real_escape_string($_GET["id"]);
$query = mysql_query("SELECT * FROM tblUsers WHERE `userId`= $id ORDER BY
userName");
...
```

Bu kodun güvenli olduğunu iddia eden birçok geliştirici bulabilirsiniz, çünkü inandıkları(!), öğrendikleri(!) kaynaklar bazı şeyleri üstü kapalı işlediği için hiçbir zaman hatayı neden yaptığını anlayamayacaktır.

Normal şartlarda uygulama aşağıdaki gibi bir cevap dönmektedir.



Biz bu açığı msre'ye rağmen exploit edebiliriz. Aynen aşağıda yaptığım gibi.



Kullandığım hiçbir karakter msre'ye takılmadı ve başarıyla exploit edilebildi. Bu hataya düşen birçok uygulama bulmak mümkün.

Bu örnektede aldığımız veriyi tırnak işaretleri ile çevrelemek en mantıklı çözüm olacaktır. Yalnız kullandığımı tırnak işaretinede dikkat etmek şart. Mesela msre ` (left quote) karakteri için herhangi bir kaçırma işlemi uygulamıyor. Aldığınız parametreyi veri değilde alan olarak kullanacağınız zaman left-quote karakterini kullanabilirsiniz fakat bu da güvenlik sorunu doğuracaktır. Aşağıdaki örneğe bir bakın;

```
...
$table = mysql_real_escape_string($_GET["table"]);
$query = mysql_query("SELECT * FROM `$table` WHERE `userId`=1");
...
```

Veri çekilecek olan tablo adını kullanıcıdan almak(!) her ne kadar güvenli gibi dursada msre left-quote'lara izin verdiği için güvenli olmayacaktır. İşin kötü yanı ise sorgunuzda kullanıcı tarafından sağlanan tablo adı, kolon adı gibi verileri tırnak işaretleri ile kullanamıyor olmanızdır. Bahsettiğim durumu exploit etmek için,

```
tblUsers`+where+`userId`=1+and+2=9+union+select+1,version(),0x4343--%09x
```

değeri yollamak bizi sonuca götürecektir.

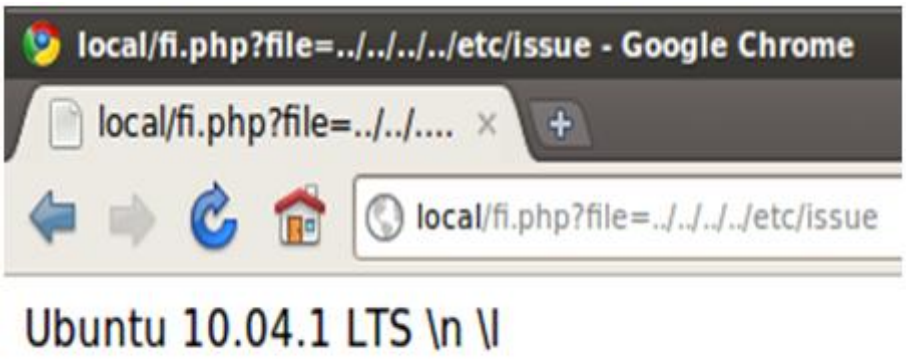
File Inclusion

Her ne kadar biraz geç gelen yama (6 yıl sonra!) NULL byte ile file inclusion yapmayı tarihe gömmüş olsada, bir gerçek var ki insanlar hala güncelleme yapmadan eski PHP versiyonları kullanıyorlar ve geliştiriciler hala bir şekilde FI zafiyetleri bırakıyorlar. FI zafiyeti bırakmak istemeyen geliştiricilerin en çok düştüğü hatalardan biriside kullanıcıdan aldığı değeri file_exists fonksiyonu ile denetleyerek include etmesi. Yine güzel bir örnek hazırladım.

```
...
$file = $_GET["file"];

if(file_exists("includes/$file")) {
    include "includes/".$file;
}
...
```

file_exists fonksiyonuna gelecek olası ../../../../etc/issue değeride gerçekten dosya sisteminde varolduğu için fonksiyonu true değeri dönecektir.



En çok karşılaşılan 3 zafiyeti ve bu zafiyetlere karşı ezbere alınan önlemlerin nasıl atlatılacağına kısaca değindim, daha yaratıcı bir çok zafiyet gerçek uygulamalarda mevcut ve birilerinin onları avlamasını beklemekte. Yapılan yanlışlarla ilgili olarakta en temel sorun web güvenliğini anlamadan önlem almaya çalışmaktır. Güvenli kodlamalar.