© Kiskapu Kft. Minden jog fenntartva

Biztonságos PHP a gyakorlatban

Ahhoz, hogy meg tudjuk védeni PHP alkalmazásainkat, az első és legfontosabb lépés a biztonsági veszélyek felismerése és megértése.

Az elmúlt két évben a PHP magfejlesztői rendkívüli munkát végeztek: a PHP felhasználói közösséget ellátták egy olyan hatékony technológiával, amely figyelemre méltóan jól teljesít változatos környezetben. Ahogy a webalkalmazások egyre népszerűbbek lettek, a webfejlesztőknek muszáj volt szembenézni a lehetséges biztonsági rések növekvő számával, melyek komolyan veszélyeztethetik a munkájukat. Sok útmutató, könyv és cikk látott napvilágot, ahogy az új technikákat fejlesztették. Ezek az újonnan felmerülő veszélyek azonban mégsem kapnak annyi figyelmet, amennyit megérdemelnek.

Jelen cikk azoknak a profi és nyílt-forráskódú PHP fejlesztőknek szól, akik magas szintű biztonságot szeretnének nyújtani felhasználóiknak illetve ügyfeleiknek. Az írásnak nem célja, hogy a programozók minden egyes kérdésére választ adjon. Inkább abban próbál segíteni, hogy az alkalmazás lehetséges problémái már a tervezési folyamat során kiderüljenek. Hosszú távon ugyanis ez teszi lehetővé nekünk, PHP fejlesztőknek, hogy az új biztonsági fenyegetésekre megfelelően reagáljunk.

Számos cikk megjelent már a biztonságos PHP fejlesztésről, s valamennyi nagyjából azonos témákat tárgyal. A következőkben ugyan gyorsan átvesszük a fontosabb alapfogalmakat, de feltételezem, hogy a legtöbben már jól ismerik ezket, ezért nem fogok sok időt tölteni velük.

register_globals

A PHP nyelvben használhatjuk a `register_globals` beállítási változót, melynek engedélyezésével az alkalmazás valamennyi változója globálissá válik. Ez annyit tesz, hogy a Webkiszolgálónak eljuttatott POST, GET, süti és session változók ugyanabba a kalapba kerülnek, ami fejlesztői szemmel nézve meglehetősen kényelmes módja a változók kezelésének.

Tervezési szempontból e beállítás engedélyezése valószínűleg alkalmazásunk egészének biztonsági szintjére kedvezőtlen hatással lesz, hiszen a felhasználók közvetlen eléréssel rendelkeznek majd az alkalmazásunkban használt valamennyi változóhoz. A PHP-t ma már alapértelmezés szerint kikapcsolt `register_globals` változóval kapjuk, és azt javaslom, a biztonság kedvéért hagyjuk is így. Ez alól csak az jelenthet kivételt, ha gépünkön régről ott maradt alkalmazások futnak, amelyek megkövetelik e változó engedélyezését.

Cross-site scripting

A népszerű Cross-site scripting, azaz XSS technika segítségével a felhasználó átveheti az irányítást a megjelenés, a tartalom, vagy akár a Web alkalmazás teljes biztonsági rendszere felett. Nem a PHP az egyetlen nyelv, amely sebezhető e módszerrel, hiszen nem nyelvi hibáról van szó, hanem sokkal inkább a webalkalmazás tervezésének bizonyos hiányosságairól.

A Cross-site scripting sok formában létezik. Az egyik népszerű megoldás, amikor HTML vagy JavaScript kódot helyezünk az űrlapok mezőibe, így kényszerítve az alkalmazást valami olyan megjelenítésére, amit egyébként nem tenne. Ez az eljárás ékesen bizonyítja, mennyire fontos, hogy minden bemenetet megszűrjünk érkezzen az a felhasználótól, másik honlaptól, vagy adatbázisból. A `htmlentities()` PHP függvény általában jó megoldás az ilyesfajta támadások elhárítására.

GET változók

A legtöbb Webalkalmazás számára nagyon fontos, hogy olyan URL-t tudjunk nyújtani a felhasználóinknak, melyet később felhasználhatnak ha vissza szeretnének térni oda ahol éppen állnak. Fejlesztőként azonban nem árt ha meg tudjuk állapítani, mely információkhoz tud a felhasználó hozzáférni a lehetséges módok bármelyikével.

A lekérdezés szöveg módosításával a felhasználó módosítani tudja az alkalmazásunkban használt változók tartalmát. Az ilyesfajta próbálkozások megakadályozása már bonyolultabb annál, mintsem hogy egyszerű bemenetszűréssel megoldható legyen, de ez a követendő irány. Az ilyen támadások ellen talán a legjobb módszer, ha alkalmazásunkhoz hibatűrő adatfolyam-kezelést és jól felépített hibakezelő rendszert készítünk.

SQL beszúrás

Ennek a webalkalmazások ellen indított rosszindulatú támadásnak olyan pusztító következményei lehetnek, melyek messze túlszárnyalva az egyéb támadási formák (például a cross-site scripting) által okozott károkat, hiszen ezzel a módszerrel akár a teljes adatbázisunkat elveszíthetjük.

Az SQL beszúrás elve nagyon egyszerű. A legtöbb Webalkalmazás sütikből, POST és GET változókból várja a bemenő paramétereket. Ezeket gyakran használjuk SQL lekérde-

1. lista PHP alatti SQL lekérdezés készítése POST változók alapján

```
<?php
$query = "SELECT id, name FROM `records` LIMIT "
        . $_POST['NUM'];
$result = $db->select($query);
?>
```

2. lista Rosszindulatú űrlap, amit SQL beszúrásos támadáshoz használhatnak fel.

```
<form action="example.com/form.php"
method="POST">
<input type="text" name="NUM"
       value="5; DELETE FROM `records`">
<input type="submit">
</form>
```

3. lista Egyszerű „Ártalmas SQL parancsok" szűrő

```
<?php
function filter_sql($input) {
    $reg = "(delete)|(update)|(union)|(insert)";
    return(eregi_replace($reg, "", $input));
}
?>
```

zések paramétereként, ezzel biztosítva a felhasználónak dinamikus tartalmat. Amennyiben a felhasználónak van valami elképzelése arról, hogy hogyan épülhet fel az adatbázisunk, elvileg képes lehet úgy módosítani a paramétereinket, hogy azzal SQL parancsokat adjon ki a lekérdezésünkön belül.
Nézzünk egy gyors példát. Az alkalmazásunk POST módszerrel vár adatokat egy űrlaptól. A célunk legyen az adatbázis x bejegyzésének megjelenítése, ahol a felhasználó tetszése szerint maga állíthatja be x értékét. Ezért az űrlapunk tartalmaz egy NUM mezőt, amely aztán a parancsfájlunkhoz továbbítja a beírt értéket. Az *1. lista* mutatja be a folyamatot. Ebben az esetben a felhasználó hamisíthat egy HTML űrlapot, amely a táblánkat teljesen kiürítő, gondosan megtervezett adatot küld el.
Ha a felhasználó úgy dönt, hogy a *2. listában* bemutatott űrlaphoz hasonlót használ, az eredmény a következőképpen néz majd ki:

```
SELECT id, name FROM `records` LIMIT 5;
DELETE FROM `records`
```

Nyilvánvalón igen egyszerű kivédeni az ilyen támadásfajtákat, mégis úgy vettem észre, rengeteg alkalmazás egyáltalán nem rendelkezik a megfelelő védelemmel.

Jelen esetben ugyan az SQL beszúrás ellen meglehetősen jó védelmet nyújtana ha az `intval()` függvénnyel egész számmá alakítanánk a NUM értékét, de fontos látnunk, hogy a fejlesztők nem gondolhatnak végig minden egyes változót minden egyes SQL lekérdezésben. Ezért a legjobb, ha alkalmazásunkban automatizáljuk ezt a műveletet. Minthogy a modern web alapú alkalmazások egyre gyakrabban használnak magmodult vagy valamilyen központi kapcsolótábla rendszert, az ilyen megoldások alkalmazás szintű megvalósítása is leegyszerűsödik. Az alkalmazásunkban használható automatizált (streamlined) képességekkel a cikk későbbi részében foglalkozunk. Addig is felsorolunk néhány ötletet, ami segíthet felépíteni a saját megoldásunkat:

1. Az SQL parancsokhoz használjunk szkifeket (szabályos kifejezéseket): ez a módszer akkor az igazi, ha nem várunk szöveges adatokat a felhasználótól, de az SQL kulcsszavak kiszűrésével általában jó eredményt érhetünk el (*3. lista*).
2. Használjunk Ellenőrző kódokat (assertion): Ezzel a megoldással később foglalkozunk majd.
3. Védett szövegek: amennyiben várhatóan nem kapunk bináris adatokat, a bemenet biztonsága érdekében érdemes védeni a szövegeket (escaping). A fenti példánkban ugyan a szövegvédelem nem sokat segített volna, de sok SQL beszúrásos támadás alapul az SQL lekérdezés megszakításán és új lekérdezés beszúrásán. Az ilyen támadásokat hatékonyan megelőzhetjük például a `mysql_escape_string()` függvénnyel .

Titkosítás

Az adatbázis kiszolgálókban és más tárhelyeken gyakran tárolunk sebezhető adatokat. Ilyen esetekben nekünk, fejlesztőknek, rendkívül fontos, hogy legyen valamilyen módszerünk az adatok biztonságos tárolására és igény szerinti könnyű előkereshetőségére.
A PHP tartalmaz egy bővítményt, melynek segítségével kihasználhatjuk az Mcrypt Könyvtár (***mcrypt.sf.net***) képességeit adataink titkosítására illetve későbbi visszakódolására. A Mcrypt PHP bővítmény dokumentációja a ➲ http:\\www.php.net/mcrypt lapon olvasható, amit alkalmazás előtt nem árt gondosan áttanulmányozni.
A bővítmény lenyűgöző mennyiségű algoritmust ismer, kezeli többek között a triple-DES-t, a Blowfisht, a Twofisht és a Two-Wayt. Hacsak nem értünk a titkosításokhoz, a Mcrypt bővítmény alkalmazása nem teljesen magától értetődő; a számtalan blokkalgoritmus és titkosítási módszer elsőre elég zavaró lehet. A *4. listában* megtekinthetjük, milyen módszereket kínál a Mcrypt bővítmény valamint hogyan használhatjuk azokat.

Ellenőrző kódok (assertions)

Ez a funkció lehetővé teszi a PHP fejlesztőnek, hogy az alkalmazást hibakezeléssel lássa el, és megőrizze az adatok integritását. Ez nem kifejezetten biztonsági jellegű funkció, ráadásul a PHP-ben több élvonalbeli nyelvnél is megtalálható (C, Python), miért is hozom fel akkor? Dióhéjban: azért, mert ha ügyfeleinknek vagy felhasználóinknak biztonságos alkalmazásokat szeretnénk nyújtani, az első lépés a hibakezelés.

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

4. lista Tipikus példa a Mcrypt bővítmény használatára

```
<?php
/* készítsünk véletlenszerű kulcsot
   de tartsuk kéznél, hiszen később
   ezt használjuk a visszakódoláshoz
*/
$key = ”AOQKJLCLIGAKJHSD
       <NKLXASLUIHJKHAS
       OIUDSgfuyJKLBLKU”;

$string = $_POST[´password´];

/* Először is meg kell nyitnunk a Mcrypt titkosí-
tó modulját*/
$mod = mcrypt_module_open
(´blowfish´,´´,´ecb´,´´);

/* Ezután létre kell hoznunk egy Initialization
   Vector-t
   a méret és a forrás alapján.
   A forrásunk tetszőleges lehet, illetve hasz
   nálhatunk előre definiált állandókat.
   A vektor méretének megadása a felhasznált
   modultól függ*/

$iv_size = mcrypt_enc_get_iv_size($mod);

/* Az alaphelyzetbe állító vektor a példánkban a
   /dev/random forrásból származó $size karakte
   rein alapszik  */

$iv =
mcrypt_create_iv($iv_size,MCRYPT_DEV_RANDOM);

/* A következő lépésként ellenőrizzük, hogy
   a kulcsunk nem túl nagy-e
   és csonkoljuk ha szükséges*/
$max_key_size = mcrypt_enc_get_key_size($mod);
$key = substr($key,0,$max_key_size);

/* Alaphelyzetbe állítjuk a titkosító algoritmust
   a mcrypt_generic_init segítségével*/
mcrypt_generic_init ($mod,$key,$iv);

/* Az adatainkat ez után már titkosíthatjuk a
   mcrypt_generic függvénnyel. A függvény
   visszatérési értéke a titkosított adat lesz*/
$encrypted = mcrypt_generic($mod,$string);

/* A Mcrypt használatának befejezése után
   fel kell szabadítanunk a folyamatban
   felhasznált vermeket*/
mcrypt_generic_deinit ($mod);

/* Végül, le kell zárnunk a felhasznált titkosító
   modult*/
mcrypt_module_close ($mod);

/* Most nézzük, hogyan kódoljuk vissza az adatot:
*/
$padded = // lásd a következő sort
mcrypt_decrypt(´blowfish´,$key,$encrypted,´ecb´,$
iv);

/* Most a visszakódolt szövegünk nullával tagolt,
   ezért el kell távolítanunk a felesleges \0-kat
*/
$plain = str_replace(”\0”,””,$padded);
echo ”Titkosított szöveg: $encrypted<br>”;
echo ”Visszakódolt szöveg: $plain<br>”;
?>
```

A Assertions PHP alatt két függvény, az `assert_options()` és az `assert()`, segítségével valósítható meg. Az elsőt alkalmazásunk alaphelyzetbe állításakor, illetve beállítás állományában hívjuk meg, a másodikat bárhol meghívhatjuk a kódban, ahol a bemenet helyességét kell kikényszerítenünk. Az *5. lista* azt mutatja be miképpen készíthetünk hibakezelő rendszert, ami egyszerű jelentést készít, ha az ellenőrzés sikertelen.
A „PHPUnit Project” egy teljes körű elemtesztelő készlet, amely ingyenesen hozzáférhető a PHP fejlesztők számára, és pontosan arra alkalmas amit mi most összeraktunk.
A honlapot a ➲ http:\\www.phpunit.sf.net címen találjuk.

Adatfolyam

Amennyiben több különféle webprojekten is dolgoztunk, az új fejlesztéseinkhez valószínűleg van már valamiféle közös váz, amit használni kezdtünk, netán kifejlesztettük saját változatunkat. Az adatkezelés központosítására sokféle módszer létezik és a projektünket leíró követelmények függvényében bizonyos modellek nyilván jobban megfelelnek mint mások. A következő néhány bekezdésben bemutatok egy egyszerű tervezési sablont amely a fejlesztőknek üzleti célú projektekhez is elegendő rugalmasságot és méretezhetőséget biztosít.
Első lépésként ki kell dolgoznunk egy olyan módszert, amellyel minden bemenő adatunkat központosíthatjuk, és egységes szűrőfelületen küldhetjük keresztül. Ezáltal a további módosításokat egyszerűen moduláris jelleggel végezhetjük majd. Példánkban következő fájlstruktúrát használjuk:

- */**index.php***: az egyetlen állomány a gyökérben.
- */**lib***: könyvtárakm amelyeket .htaccess véd.
- */**lib**/**config.inc.php***: beállítás állomány.
- */**tpl***: sablonok, szintén a .htaccess védelme alatt.
- */**doc***: projektek és API dokumentációk.
- */**images***: képek.
- */**classes***: osztályok, .htaccess védelemmel.

Amint az a *2. ábrán* látható, alkalmazásunk magját az ***index.php*** állomány alkotja, melynek közvetlen elérése van

5. lista Hibajelentés Assertion segítségével

```
<?php

/* A teljes alkalmazásunkban ki-be
   kapcsolhatjuk az
   assertion képességet az ASSERT_ACTIVE változó
   1 vagy 0
   értékre állításával.
*/
assert_options(ASSERT_ACTIVE,1);

/* Azt szeretnénk, ha az alkalmazásunk kilépne
   ha az assertion sikertelen. (Legalábbis ebben
   a példában)
*/
assert_options(ASSERT_BAIL,1);

/* Példánkban magunk végezzük a hibajelentést
   ezért kikapcsoljuk az alapértelmezett
   figyelmeztetéseket.
*/
assert_options(ASSERT_WARNING,0);

/* Az sikertelen assertion ellenőrzéskor
   meghívandó
   saját függvényünk neve „display_error” lesz.
*/
assert_options(ASSERT_CALLBACK, ”display_error”);

$email = strtolower($_POST[´email´]);
$parts = array();

// elkészítjük a szabályos kifejezésünket
$regex = ”^([.\´a-z0-9]+)@([.\´a-z0-9]+)$”;

/* Helyes formátum ellenőrzése, és az email
   cím ellenőrzése egy időben történik.
   Figyeljük meg a különleges formátumot.
   Minden idézőjelek közt található, a hibát
   pedig
   megjegyzésbe tettük. Ezt a hibát később,
   szabályos
   kifejezéssel fogjuk kigyűjteni.
*/
assert(”ereg(\$regex, \$email, \$parts); /*
      hibás e-mail cím: $email */”);

/* Ez a rész nem hajtódik végre, ha az
   assertation
   sikertelen így biztonságosan továbbléphetünk
*/
$username = $parts[1];
echo ”Üdv otthon, ” . $username;

// Most következik az ASSERT_CALLBACK függvényünk
function display_error($file, $line, $error) {

    // Ez a rész gyűjti ki a hibaüzenetet
    $regex = ”(.*)/\* (.*)\*/”;
    $parts = array();
    ereg($regex, $error, $parts);
    $msg = $parts[2];

    // csinos kimenetet készíthetünk
    echo ”
    <table bgcolor=\”#bbbbee\”>
    <tr><td colspan=´2´ align=´center´>
    <b>Hibajelentést</b>
    </td></tr>
    <tr><td>Állomány:</td><td>$file</td></tr>
    <tr><td>Sor:</td><td>$line</td></tr>
    <tr><td>Üzenet:</td><td>$msg</td></tr>
    ”;

}

?>
```

a `template`, `class` vagy `configuration` állományok bármelyikéhez; a felhasználó azonban soha nem érheti el ezeket az állományokat.
Menjünk végig lépésről lépésre a 2. ábrán látható vázlaton. Vegyük példaként azt a folyamatot, ahogy a felhasználó bejelentkezik a rendszerbe.

1. A felhasználó paraméter nélkül kéri le az ***index.php*** állományt. Az index létrehoz egy vermet, majd továbbadja a kapcsolótáblának, amely meghívja az alapértelmezett modult. A modul sablon segítségével megjeleníti az alkalmazás alapértelmezett bejelentkező képernyőjét.
2. A felhasználó kitölti, majd elküldi az azonosítási űrlapot. Az űrlap kimenetét valami ilyesmi címre irányítja: `?module=account&action=login`. A kapcsolótábla meghívja a login függvényt, amely nem más mint a user osztály elérési felülete. A függvény létrehoz egy példányt a user osztályból. Ez az objektum szolgál csatolófelületként a felhasználónk és az adatbázis között, valamint ez végzi el a lekérdezést.
3. Az adatbázis visszaküldi az adatokat az objektumnak, majd az objektumból a modulhoz kerül, amely beállítja a megfelelő session változókat, meghívja a megfelelő sablont, és segítségével módosítja a vermet. Végül a válaszüzenetet elküldi az index-nek.

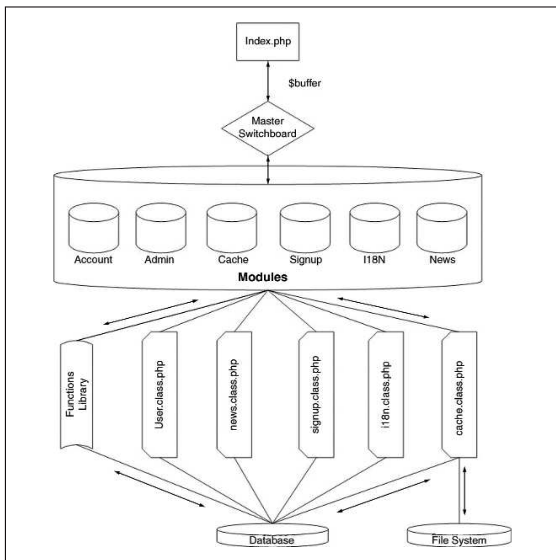
A bemutatott modell adatfolyam-kezelése első látásra talán kicsit bonyolultnak tűnhet, valójában azonban meglehetősen egyszerű. A felhasználói bemenet gyorsan a megfelelő modulhoz kerül, a hibakezelést a kapcsolótábla szintjén végezzük. Az egyéb bemenetek – mint például az adatbázis-adatok és fájlrendszer-elérések – szűrését a nekik megfelelő osztályok végzik. Minden osztály egy

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva

```
Error Report
File:    /var/www/localhost/htdocs/assert1.php
Line:    38
Message: Invalid email address: test@
```

1. ábra Az 5. lista által készített mintaüzenet



2. ábra Alkalmazás mag

különleges vázosztályt bővít amely biztosítja a megfelelő bemenetszűrési képességeket, így egyik osztálynak sem kell ilyesmi miatt aggódnia.
A modell igen hatékony, hiszen méretezhető és hibatűrő szerkezetre épül, de ne feledjük, hogy más, érdekes modellek is léteznek. Vethetünk például egy pillantást a Phrame Project weblapjára (➲ http:\\www.phrame.sf.net) , amely az MVC rendszeren (➲ http:\\www.ootips.org/mvc-pattern.html) alapuló Model2 megközelítést alkalmazza.

Biztonsági mód

A PHP biztonsági módja (Safe mode) olyasmi amit mindenképpen érdemes megtanulni minden PHP fejlesztőnek és rendszergazdának. A biztonsági mód olyan beállítások gyűjteménye, melyek segítségével a rendszergazda befolyásolni tudja a PHP-értelmező működését és biztonsági alapelveket határozhat meg. Rendszergazdai szemmel nézve ez annyit jelent, hogy meg kell tanulnunk, hogyan kell helyesen beállítanunk a rendszert úgy, hogy közben ne tegyük lehetetlenné fejlesztőinknek az alkalmazásaik elindítását a kiszolgálónkon. Fejlesztői szemmel nézve azt kell megtanulnunk, milyen hatással lehet egy adott képesség az alkalmazásunkra, ha történetesen be van kapcsolva.
A `safe_mode` beállítások bekapcsolása indokolt lehet, ha PHP alkalmazásokat futtató megosztott kiszolgálókat üzemeltetünk, és a kiszolgálókat használó PHP fejlesztőkben nem bízhatunk meg. A `safe_mode` beindítása ***php.ini*** állományunkban hatékonyan meggátol minden fájlrendszer elérést, kivéve, ha a fájl tulajdonosának UID azonosítója nem azonos a futó parancsfájl UID értékével. A PHP `safe_mode` alatt is lehetőséget ad e funkció átállítására, ehhez a `safe_mode_gid` kapcsolót kell engedélyeznünk. Ilyenkor, a PHP az állományok GID értékét vizsgálja az UID érték helyett.
Jó gyakorlat, ha felhasználóinknak nem engedélyezzük a rendszer bármely bináris állományának végrehajtását; ilyenkor jöhet szóba a `safe_mode_exec_dir`. Ennek a felbecsülhetetlen értékű funkciónak a segítségével utasíthatjuk a PHP-t, hogy csak akkor hajtson végére bármilyen programot az `exec()` vagy bármely más függvényen keresztül, ha a program megtalálható a `safe_mode_exec_dir` által megjelölt könyvtárban. (Ilyen például az */**usr**/**local**/**php**/**bin***).
Miután megismertük a PHP `safe_mode` engedélyezésekor érvénybe lépő korlátozásokat, elkezdhetünk olyan programokat fejleszteni, amelyek nem robbannak le, ha ilyen beállítással rendelkező gépen kell futniuk. Sok ISP használja a `safe_mode` beállításait. A legfontosabb szempontok a következők:

- A fájlműveleteket, legyenek azok írási vagy olvasási műveletek, próbáljuk meg az alkalmazásunkhoz adott állományokra korlátozni.
- Ne használjuk ki, hogy valamilyen külső program már telepítve van vagy végrehajtható, ha a projektünk nem csak a saját kiszolgálóinkon fog futni.

A rendszergazdák ezen felül más hatékony eszközöket is használhatnak rendszer teljes körű biztonságának növelésére. Ide tartozik a `disable_functions` amely megakadályozza, hogy bizonyos függvényeket meghívjunk, valamint az olyan függvények mint az `open_basedir`, amely minden fájlműveletet egy megadott könyvtárra korlátoz.

A PHP dokumentáló csapat által készített irodalom részletesen foglalkozik a témával. Ezen túl készült olyan írás is, amely a `safe_mode`, a hozzá kapcsolódó függvények valamint azok következményeinek részletei taglalja.

Linux Journal 2004. április, 120. szám

Xavier Spriet az utóbbi négy év során PHP alatt fejlesztett programokat. A eliquidMEDIA International vezető fejlesztője. Xavier xavier@wuug.org címen érhető el.

FORRÁSOK

Mcrypt bővítmény: php.net/mcrypt
Mcrypt Project: mcrypt.sf.net
Az MVC minta: ootips.org/mvc-pattern.html
PHP Dokumentáció: php.net/manual/en
PHP Biztonság: php.net/manual/en/security.index.php
A PHPUnit Projekt: phpunit.sf.net
A Phrame Projekt: phrame.sf.net
Safe Mode: php.net/manual/en/features.safe-mode.php