PHP-vel biztonságosan (2. rész)

A figyelmes programozás szempontjain túl szó esik a PHP témába vágó beállítási lehetőségeiről és a PHP-fejlesztések ez irányú jövőjéről.

A múlt havi számban megjelent cikk folytatásaként nagyrészt a PHP körüli rendszeradminisztrációs feladatokról fogunk szólni. Vizsgálódásunk tárgya még a biztonsági beállítások mellett a PHP ezen területen várható jövője (részben jelene) is.

A PHP védelmi lehetőségei

A ***php.ini***, a PHP beállításait tartalmazó fájl számos olyan lehetőséget tartalmaz, amelyek használatával PHP-alkalmazásainkat jóval biztonságosabb környezetben tudhatjuk. Ez kiterjed az adatok védelmére, a hibajelzések megfelelő kezelésére. Akár egyes PHP-elemek, -függvénynek letiltása is lehetséges. Érdemes mindjárt azzal kezdenünk, hogy a szószátyár hibakezelőt kicsit gatyába rázzuk.

A hibakezelés beállítása

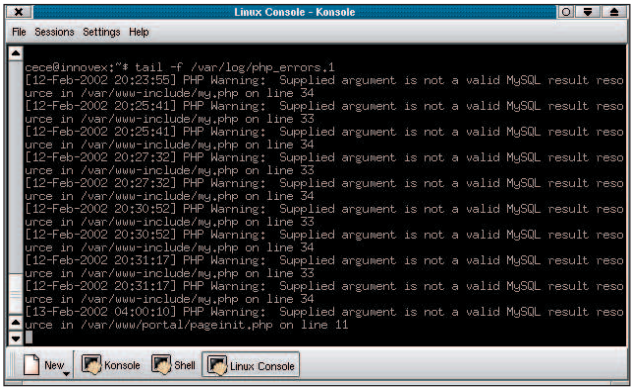
Célszerű mindjárt a PHP alapértelmezett hibakezelési beállításait megváltoztatni. Az `error_reporting` eredeti értéke (`E_ALL & ~E_NOTICE`) tökéletesen megfelelő hibakezelési szint, minden rendben is van vele. A `display_messages` kapcsoló bekapcsolt értéke viszont egy éles kiszolgálón megengedhetetlen. Ezt a kapcsolót `On` vagy `Off` állapotba állításával vezérelhetjük, mégpedig aszerint, hogy a PHP-hibaüzenetek a weblapba ágyazva ott jelenjenek-e meg, ahol a hiba keletkezett. Valójában a PHP hibaüzeneteihez egyedül az üzemeltetőnek (vagy a fejlesztőnek) van köze. Programfejlesztés, hibakeresés során esetleg kényelmesebb lehet bekapcsolva tartani, így a hibamentesítés gyorsabban folyhat. A nagyközönség elé tárt kiszolgálón viszont megéri a következő beállításokat végrehajtani:

```
display_errors = Off
log_errors = On
error_log = <a fÆjl, ahovÆ a hibÆkat  rhatja >
```

E beállítások hatására két dolog történik:

- A hibás működést okozó látogató a hibaüzenetet nem látja. Valaki hibát hozhatott létre véletlen módon, programhibába ütközve, de lehetséges, hogy ez a valaki rendszerünk gyenge pontját puhatolva kényszeríti programunkat hibás működésre.
- A hibák naplózása révén mi viszont minden hibát láthatunk, nem csak a magunk által létrehozottakat. Így a menet közbeni felmerülő működési rendellenességekről is első kézből kaphatunk adatokat.

A `display_errors` kapcsoló mellett a `log_errors` kapcsoló `On` állapota gondoskodik arról, hogy minden egyes hiabüzenet naplófájlba kerüljön. Ez a naplófájl az ***php.ini*** `error_log` beállításával adható meg. Értéke vagy egy fájl teljes elérési útvonalával, vagy pedig a `syslog` szó lehet, ekkor a rendszernaplózó feladata lesz a hibaüzenet naplóbejegyzéseinek lekezelése. Több tartományt kiszolgáló rendszereken a naplózófájlt tartományonként külön-külön is beállíthatjuk, kihasználva, hogy ezt az Apache beállításaiban is módosíthatjuk:




```
<VirtualHost azegyikdomenem.lx>
ServerName azegyikdomenem.lx
ServerAdmin cece@balaton.hu
DocumentRoot /var/www/azegyikdomenem.lx/
php_admin_value error_log
/var/log/php/azegyikdomenem.lx-error.log
... egyØb beÆll tÆsaink ...
</VirtualHost>
```

E kis átállítgatások következtében PHP-s rendszerünk egy csapásra felügyelhetővé válik. Az sem megvetendő előny, hogy innentől nem vagyunk arra utalva, hogy a hibát egyáltalán bejelenti-e valaki, és ha igen, milyen minőségben. Lássuk, mit lehet még tenni!

Beillesztendő PHP-kódok védelme

Az előző cikkben volt pár példa arra nézvést, hogy mit tehetünk PHP-programunk csak `include()`-ra szánt részeinek védelme érdekében. Mindazok a pótmegoldások, amelyekről ott értekeztem, csak akkor jöhetnek szóba, ha a kiszolgáló nem a mi kezünkben van, és nincs e fájlok védelmére beállítva. Valós védekezésképpen ezeket az állományokat a nyilvánosság elé tárt könyvtáron kívül, külön kell tárolni.
A függvényeket és egyéb programrészleteket tartalmazó fájlok könnyebb elérése érdekében hasznos beállítani, hol keresse őket a PHP. Lehetne ugyan teljes elérési útvonalak megadásával is ügyködni, de ez jócskán visszadobná a PHP-ben írt rendszer hordozhatóságát. A ***php.ini***-ben az `include_path` segítségével adható meg egy vagy akár több könyvtár is, ahol a beillesztendő fájlokat sorban keresni fogja. Az egyes lehetséges könyvtármegjelöléseket kettősponttal kell elválasztani, valahogy így:

```
include_path .:/usr/local/php/include/
```

No és természetesen igény szerint akár tartományonként is megadhatjuk ezeket az ***httpd.conf***-ban – az `error_log` példájára.

magic_quotes-huncutságok

A PHP jelenlegi változata több helyen nyögi még a PHP3 hagyatékait. Támogatottságukat ugyanis nem lehet egyik napról a másikra eltüntetni, mivel rengeteg PHP-ben írt oldal mondaná fel a szolgálatot egy rendszerfrissítés után. Nem pont a legfájdalmasabb, de az egyik ilyen maradvány a `magic_quotes_gpc` bekapcsolt állapota a PHP hivatalos alapbeállításaiban. Hatására a PHP a bejövő adatokon önműködően végrehajt bizonyos átalakításokat. Minden egyszeres és kétszeres idézőjelet fordított perjellel (backslash) véd, természetesen magát ezt a jelet is. Ezzel a lépéssel, vagyis hogy minden érkező adatot így alakít, sok kellemetlen beavatkozástól védheti meg az amúgy óvatlan programfejlesztőt. Csakhogy e szolgáltatás miatt sajnos a PHP-programozók java a tapasztalat szerint valóban gyanútlanná vált.
A teljesen kezeletlen és ellenőrizetlen beérkező adatok egy SQL-lekérésbe vagy -parancsba kerülve nagy veszélyforrást jelentenek, hisz a következő kis SQL-adatmódosító parancs könnyedén támadófelület lehet:

```
$qstr =  SELECT * FROM bizalmasadatok
WHERE tulajdonos= $PHP_AUTH_USER
➥AND tipus=$tipus ;
```

Itt a `$tipus` helyére a terv szerint egy típust azonosító, legördülő menüből választható számnak kell kerülnie. Az űrlapot mentve és a legördülő menüt módosítva viszont ez is a `$tipus` változóba kerülhet: `1 OR (tulajdonos= masvalaki )`. Ekkor egy ilyen lekérést kapunk:

```
SELECT * FROM bizalmasadatok
WHERE tulajdonos= $PHP_AUTH_USER
➥AND tipus=1 OR (tulajdonos =  masvalaki )
```

Ezáltal a lekérés minden rekordra teljesül, ahol az adott típussal a saját bizalmas adataink is szerepelnek, és a „masvalaki” belépőkódú ismeretlen adatai is a listához csapódnak. Megszokott (terv szerinti) esetben a másik felhasználó bizalmas adataihoz nem férhetünk hozzá, de egy ilyen kívülről megbütykölt lekéréssel ez mégis megtörténhet. A legjobb lenne, ha figyelnénk, hogy `$tipus` csakis egy 0 és 50 közti egész szám lehet, minden más értéket eldobva. Ehelyett ott van a `magic_quotes_gpc`, amely a fenti gondot annyival oldja meg, hogy a két aposztrófot egy-egy fordított perjellel levédi, így a lekérés a jogtalan adatszerzést meghiúsítva SQL-hibát hoz. Ez a védőbástya igencsak minimális védelmet nyújt. Előfordulhat ugyanis olyan lekérés is, ahol minden szűrőfeltétel számszerű. Talán megfontolandó a számszerű adatokat is idézőjelek közé szorítani, mert ekkor ez és a `magic_quotes_gpc` jól együttműködik. Hangsúlyozom azonban, hogy a biztos védelmet csakis alapos adatellenőrzéssel lehet elérni.
Frissítéskor a telepítő szerencsére az eredeti beállításokat nem bántja, így nem kell attól tartanunk, hogy a `magic_quotes_gpc` beállítás a tudtunk nélkül egyszer csak megváltozik. Érdemes viszont észben tartani, hogy a jövőt illetően a PHP ezen képessége is nemkívánatossá vált. A PHP-terjesztésekben található ***php.ini-optimized*** állomány tartalma is ezt igazolja. Ez már az új elvárásoknak felel meg, ámde a régebbi alkalmazásokkal nem együttműködő beállításokkal rendelkezik. A PHP fejlesztőgárdája a jövőben a `magic_quotes_gpc` alapbeállítását `Off` állásba fogja helyezni Az adatok kezeléséről tehát mindenkinek magának kell gondoskodnia. A dolog oka prózai: ezzel is rá kívánják venni a PHP-ben fejlesztőket, hogy minden egyes bejövő adatot maguk ellenőrizzenek. A túl nagy kényelem segíti a figyelem lankadását, márpedig egy korrektül megírt programnak `magic_quotes_gpc` nélkül is biztonságosan kell üzemelnie.

Safe Mode

A PHP-t is adó tárhelyszolgáltatás egyik alapfeladata, hogy minden egyes a kiszolgálón futó PHP-parancsfájl ugyanazon felhasználóazonosítóval fusson. Ez az azonosító azonos az Apache webkiszolgálóhoz beállítottal. Ez előhív egy igen fájdalmas probléma, amelynek során a szerveren ügyködő előfizetőket nem tudjuk védeni egymástól, hiszen minden PHP egy ***nobody*** vagy ***www-data*** vagy hasonló nagy közös felhasználó neve alatt működik. Ezáltal egy egyszerű program segítségével az egész webkiszolgáló terület átböngészhető. Márpedig ez nyilvánvalóan nem szerencsés, mivel ilyen módon bárki hozzáférhet a kiszolgálón a másik felhasználó legféltettebb PHP- és egyéb állományaihoz. Például ahhoz is, amelyben a kényes adatokat tartalmazó MySQL-adatbázishoz tartozó felhasználó azonosító-jelszó párost tárolja.
A jelenlegi Apache-beállításokban nincs is lehetőség arra, hogy a különböző virtuális kiszolgálók különböző felhasználóazonosító alatt fussanak. Egyedül ***suexec***-kel fordított Apache esetén tudjuk ezt a CGI módban működő PHP-vel ezt mégis megtenni – ami viszont rengeteg egyéb rizikót hoz be a képletbe. Megoldást ebben az Apache 2.x változatai fognak hozni, ahol már minden további nélkül virtuális szerverenként állítgatható lesz az `User` és `Group` Apache fordítói utasítás, és ez a modulként beépülő PHP esetén is hatni fog. Eképpen a kiszolgálón előfizető felhasználók már nem figyelhetnek be mások féltett könyvtáraiba.
Addig sem maradhat tétlen az ember. A fenti problémák orvoslására született meg a ***Safe Mode*** PHP-kiegészítés. Ezt a biztonságosabb üzemeltetést biztosító futási módot a ***php.ini***-ben tudjuk ki-bekapcsolni:

```
safe_mode = On vagy Off
```

Ilyen bekapcsolt állapotban a PHP több oldalról is igyekszik levédeni a kiszolgálón tartózkodó adatokat. Egyrészt korlátozható a fájlrendszerben az a terület, amit a PHP látni képes. Ezt és magát a biztonságos üzemelést biztosító kapcsolót az Apache beállító állományán keresztül akár virtuális kiszolgálónként is kapcsolgathatjuk:

```
<VirtualHost www.vedett.hu>
    .. egyØb beÆll tÆsok ..
    php_admin_value safe_mode 1
    php_admin_value open_basedir
/var/www/html/safephphost/
</VirtualHost>
```

Ha bekapcsoljuk a `safe_mode` állapotot, a PHP a PHP-fájlok tulajdonosi viszonyainak is figyelmet szentel. Innentől a PHP-parancsfájl csak a tulajdonosával azonos birtokviszonyú fájlokkal tud valamit kezdeni. Más esetben a műveletet egy `Warning: SAFE MODE Restriction in effect.` kezdetű hibaüzenettel visszautasítja. Ez a korlátozás a PHP összes fájlokat kezelő vagy bármilyen műveletre felhasználó parancsára érvényes.
Az `open_basedir`-ben megadott könyvtáron kívül ekkor a PHP nem lát, és ugyanez történik a PHP által meghívott parancssori alkalmazásokkal is. Ez a védelem persze korántsem

tökéletes, hisz nem a megfelelő szinten próbálja orvosolni a problémát. Az `open_basedir` a `safe_mode` ki- vagy bekapcsolt állapotától függetlenül működőképes. Az `open_basedir` után több könyvtár is megadható, itt szintén kettőspont az elválasztó karakter. Fontos megjegyeznünk azt is, hogy az `open_basedir` valójában nem könyvtárat ad meg, hanem az elérési útvonal kezdeti karaktereit. Tehát a */var/www/phpinc* a */var/www/phpinc/* könyvtárnak felel meg, de ugyanígy a */var/www/phpinclude* vagy a */var/www/phpincs* könyvtáraknak is megfelelhet. Érdemes tehát az ebből előálló galibákat elkerülendő a megadott könyvtárakat záró perjellel ellátni.
A `safe_mode`-nak is létezik egy, az `open_basedir`-hez hasonló könyvtármegadási lehetősége. A PHP csak a `safe_mode_exec_dir` után megadott könyvtáron belül hagyja, hogy a futtatható állományokat a `system()`, `exec()` vagy hasonló függvényekkel meghívhassuk. A végrehajtó operátor, azaz a balra dőlő aposztróf, amelynek segítségével egyszerű módon lehet parancssori utasításokat futtatni, a `safe_mode` bekapcsolt állapotában egy az egyben le van tiltva.
A `disable_functions` ***php.ini*** fordítói utasítás után a vesszővel elválasztva felsorolt függvények a programok számára nem lesznek elérhetőek. Végső finomításként talán érdemes körülnézni, hisz pár veszélyesnek (és a feladat szempontjából feleslegesnek) ítélt függvény lekapcsolása is meghosszabbíthatja a nyugodtan átaludt órák számát.

Mérföldkő: PHP 4.1.0 és register_globals

A PHP népszerűségét jórészt annak köszönheti, hogy könnyedén és gyorsan készíthető vele az Internetre vagy a belső hálózatra szánt alkalmazás. E könnyedség egyik oka, hogy a PHP alapesetben az űrlapokról, URL-ekből, sütikből érkező adatokat azonnal változóban tálalja. Ez rengeteg terhet vesz le a programozó válláról, de sajnos így könnyedén lehet biztonságosnak egyáltalán nem mondható kódot írni. Emellett további könnyítés az is, hogy változóinkat nem szükséges a programkód elején előre deklarálni. Emiatt viszont kívülről bármilyen nevű és értékű változó bejuttatható a futó PHP-programba.
Egy példa, ami jól mutatja ennek veszélyeit
(forrás: ➲ http://www.php.net):

```
<?php

if (authenticate_user()) {
  $authenticated = true;
}
...

?>
```

A fenti példa feltételezi, hogy a felhasználó azonosítást egy `authenticate_user()` nevű PHP-függvény végzi (a felhasználónév-jelszó páros kiértékelésével). Rendes körülmények között a fenti kódrészlet az `$authenticated` változónak abban az esetben ad „igaz” értéket, ha a belépés sikeres volt. A kód további részében a program ezt a változót használja fel a jogosultság megállapítására. Feltéve, hogy ezt a PHP-ben megírt oldalt ***nagyon_nagy_titkaim.php***-nek nevezik, a következő kézzel módosított URL-lel hívva a jogtalan behatolás meg is történt:

```
nagyon_nagy_titkaim.php?authenticated=1
```

Természetesen egy kis adalékkal a fenti kód is megfelelő lenne, mert a feltétel kiértékelése előtt elég lenne az `$authenticated` változót „hamis” értékkel ellátnunk. Ezt az `else`-ágban is megtehetnénk, így védve ki a csalást. Ezeknél azonban sajnos néha sokkal kevésbé szembetűnőbb logikai hibákat is ejthetünk a programtervezés során, és a programunk esetleg kívülről manipulálhatóvá válhat.
A hosszú távú tapasztalatok – és a tény, hogy a PHP mennyire elterjedt – nyomán született meg az a döntés, amelyet a tavalyi év végén a PHP fejlesztőcsapata hozott meg. A 4.1.0 változattól a ***php.ini***-ben a `register_globals` néven ismert kapcsoló alapértelmezésként kikapcsolt állapotban kerül az újonnan telepített PHP beállításait tartalmazó ***php.ini*** fájlba. E kapcsoló segítségével lehet a PHP fenti tulajdonságát, azaz a kívülről jövő adatokból automatikusan változókat létrehozó képességét ki- és bekapcsolni. Jelenleg a `register_globals` `Off`-ra állítása még csak ajánlás, de a fejlesztők nem titkolt szándéka az, hogy idővel ezt a szolgáltatást teljes egészében kivegyék a PHP-ből. Ezáltal a PHP-ben való fejlesztés kissé nehézkesebbé válhat, de ez a nehézség szükségszerű.
Az ajánlásnak eleget tenni és meglévő programjainkat mindezen körülményeknek megfeleltetni meglehetősen nagy feladat.
A nagy traumát enyhítendő bevezetésre került hét új, az átállást és a jövőbeni programozást valamelyest megkönnyítő asszociatív tömb, amelyek minden futás során automatikusan létrejönnek:

- $_GET - a webcímbe (URL) ágyazott, valamint a GET-eljárással elküldött űrlapadatok elérhetősége. Tartalmában a régóta jól ismert `$HTTP_GET_VARS` tömbnek felel meg.
- $_POST – a `$HTTP_POST_VARS` megfelelő párja. A POST-eljárással küldött űrlapadatok elérésére szolgál.
- $_COOKIE – a „süti” forrásból érkező adatokat tartalmazó `$HTTP_COOKIE_VARS` megfelelő párja.
- $_SERVER – a `$HTTP_SERVER_VARS` rövid nevű társa.
- $_ENV – kitalálhatatlan, de a `$HTTP_ENV_VARS`-nak felel meg.
- $_REQUEST – nem, nem, ilyen eleddig nem volt: ez a `$_GET`, `$_POST` és a `$_COOKIE` tömbök uniója. Itt megtalálható minden olyan adat, ami a kiszolgálón kívülről származik. Más szóval azok, amelyekben nem bízhatunk vakon.
- $_SESSION – a PHP saját munkamenet- (session) kezelését megvalósító moduljának tárolt adatai.

Ennyiben nem merül ki a fejlesztőknek az átállni szándékozókat célzó segítőkészsége. A fenti hét tömb PHP-kódunkban minden további nélkül bárhol elérhető. Függvényből hivatkozva rá szükségtelen mindjárt az elején a `global` paranccsal a függvény felségterületére behúznunk. További könnyítés, hogy a `$_SESSION` tömbbe közvetlenül írt adatok ugyanúgy bekerülnek a körforgásba, mintha egy `session_register()` segítségével tettük volna meg. Ilyen tulajdonsággal a többi tömb (a hétből) nem rendelkezik.

Programkövetés

És mindezeknek a végére hagytam még egy nagyon fontos szempontot. A PHP-t folyamatosan fejlesztik, a nyilvánosságra került hibákat (köztük a biztonsági szempontból érdekeseket) időről időre javítják. Fontos tehát telepített PHP-nkat állandó jelleggel frissíteni.

Heilig (Cece) Szabolcs
(cece@php.net) Veszprémben él. Hobbija, kedvenc időtöltése és munkája is a programozás. Szabadidejében hajlamos kerékpárra pattanni, vagy baráti társaságban szerepjátékokkal foglalatoskodni.