PHP-vel biztonságosan (1. rész)

Mire figyeljünk, ha nem akarjuk, hogy féltett adataink egy rosszul megírt PHP-program révén illetéktelenek kezébe kerüljenek.

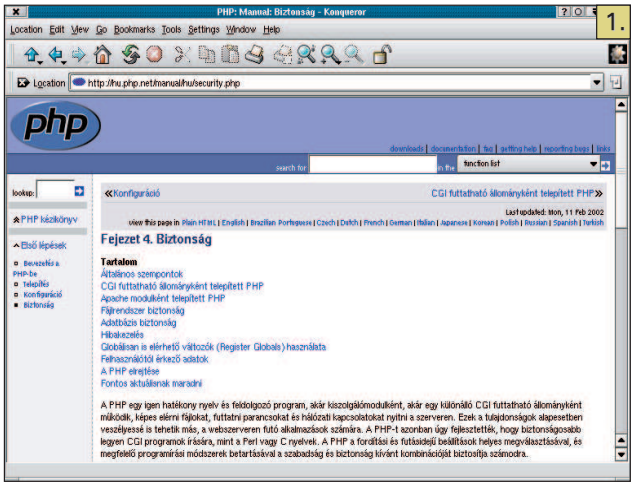
Félreértés ne essék, teljesen tökéletes rendszer nem létezik. Olvasóink e sorok végigolvasása után sem lesznek képesek törhetetlen webes rendszereket írni és fenntartani, de a célom nem is ez, hanem csupán annyi, hogy fellebbentsem a fátylat azokról az elkövethető hibákról, amelyek a leggyakrabban ütik fel fejüket a gombamód szaporodó PHP-ben írt webes alkalmazásokban.
Cikkem egy GNU/Linux-környezetben működő Apache webkiszolgálót és egy 4.x változatszámú PHP-t használó rendszert vesz alapul.

Kinek a feladata?

Egy PHP-projekt körül a biztonsági teendők alapvetően két nagy csoportra bonthatók. A telepített webkiszolgáló és PHP-értelmező beállításait a biztonsági követelményeknek megfelelően finomítani kell. Ezek a rendszergazda hatáskörébe tartozó feladatok.
A rendszergazda mindent megtehet a PHP-rendszer védelme érdekében, de erőfeszítései hiábavalóak, ha maguk a PHP-ben írt programok nem felelnek meg azoknak az alapvető feltételeknek, amelyek egy nagyközönség számára szánt programtól elvárhatók. Ezen elvárások röviden összefoglalva: a programmal a felhasználó ne tehessen olyat, amit mi nem akartunk.

A forráskód elrejtése lenne a megoldás?

Egyes elképzelések szerint a legjobb védekezési módszer a kívülállók sötét tudatlanságban tartása, azaz ha semmit nem árulunk el programunk belső felépítéséről, vagyis eltitkoljuk a forráskódját. Ez nem igazán célravezető megoldás. A trükkös, játékos kedvű idegenek, akik webalkalmazásunk meggyötrésére hivatottak, a sok kis apró adatból nagyon sok információt ki képesek gyűjteni. Inkább arra érdemes törekedni, hogy PHP-s rendszereinket úgy írjuk meg, hogy egyetlen rosszindulatú felhasználó se tudjon károkat okozni, még a teljes forrás ismeretében sem. Ennek már csak egészségügyi szempontból is hatalmas előnyei vannak, hiszen nem kell amiatt rettegnünk, hogy féltett forrásunk illetéktelenek kezébe kerül.
Működő rendszerünk fájljai között azonban olyanok is akadhatnak, melyeket joggal féltünk a külvilágtól. Teszem azt, egy beállításokat tartalmazó ***.inc*** fájlban lehetnek olyan érzékeny adatok, amelyeket nem a nagyközönségnek szántunk, például egy adatbázis-hozzáférés jelszava. Átlagos kiszolgálóbeállítást alapul véve az ilyen ***.inc*** fájlok – ha a nyilvános webdokumentumok könyvtárában helyezkednek el – elérhetővé válhatnak. Hibás lépés az ilyen érzékeny adatokat tartalmazó fájlokat itt tárolni. Az első, amit megtehetünk, hogy a fájlrendszer nyilvánosan el nem érhető pontján tároljuk őket. Ha nem tennénk, a támadási lehetőségeket felmérő kívülállónak elég lenne csupán a fájl nevét megismernie, és máris olyat láthatna, amit nem szabad. Ennek oka, hogy a webkiszolgáló csak olyan fájlkiterjesztésekre ereszti rá a PHP-értelmezőt, amit annak beállításaiban sorolhatunk fel. Általában a következő kiterjesztéseket vizsgálja: a ***.php .php3 .phtml***. Ezek közé a ***.inc*** is



A PHP-leírás ide vonatkozó része már magyarul is elérhető

felvehető, viszont senki sem szeretné, ha a ***.inc***-állományokat önállóan is futtatni lehetne. Ezért kell a webkiszolgáló dokumentum saját könyvtárán kívül helyezni azt, amit nem akarunk, hogy közvetlenül elérjenek, vagy akárcsak megcímezzenek. Ezt rendszerint felhasználóként is megtehetjük. Ha mégsincs rá lehetőség, más megoldás is létezik: ha beillesztendő fájljainkat szintén ***.php*** kiterjesztéssel láthatjuk el. Ekkor viszont arra kell figyelni, hogy ezek a nem önálló futásra tervezett programrészletek önmagukban is meghívhatók, és ezáltal újabb biztonsági rések keletkeztek. Ennek legkönnyebb kivédése, ha beillesztett fájljaink elején leellenőrizzük, nem önállóan lettek-e meghívva. Ezt a következő módon érhetjük el – már feltéve, hogy a védeni kívánt fájl a ***szamlalo.php*** (egy ***szamlalo.inc*** nevű állományt nevezve át) névre hallgat. A beillesztendő fájlt az *1. listán* látható módon kezdjük. Ezáltal a közvetlen lefuttatást meghiúsítottuk, ilyen próbálkozások esetén a beillesztésre hivatott ***szamlalo.php*** weboldalunk nyitólapjára irányít át. A fenti programrészletet eseménynaplózó rendszerrel ki lehet egészíteni , amely rögzíti számunkra az ilyen gyanús eseményeket.

Ne bízzunk a bejövő adatokban sem!

A webkiszolgálón futó programokban a legkönnyebben elkövethető hiba, ha az oldalról oldalra átadott adatokban vakon megbízunk. Egy egyszerű címsorban továbbított adatsort bárki módosíthat, és ezzel rossz esetben a program működése nagyon eltérhet attól, mint amire szántuk.
Minden egyes megkapott adatot érdemes alaposan megvizsgálni, hogy megfelel-e az elvárt formai és tartalmi követelményeknek. Ha nem tesszük, jó esetben is előfordulhat, hogy a nem kívánt tartalmú adatot is elfogadjuk, például a távoli jövőbe mutató születési dátumot – és ez még csak a legkisebb rossz, ami megtörténhet…

1. lista Számláló

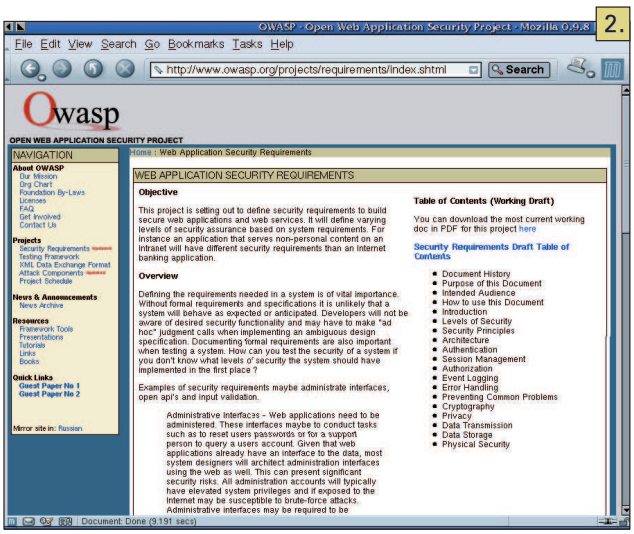
```
<?

if (basename($PHP_SELF) == "szamlalo.php") {

     header("Location: /");
     die();

}

?>
```

2. lista A letölthető anyagok

```
<HTML>
<HEAD>
     <TITLE>Let ltheti anyagok</TITLE>
</HEAD>
<BODY>

<FORM ACTION="letoltes.php" METHOD="POST">

     <SELECT NAME="mit">
       <OPTION VALUE="dokumentacio.pdf"
       >dokumentacio.pdf
         ➥<OPTION VALUE="terkep.pdf">
         ➥terkep.pdf
       <OPTION VALUE="tech_parameterek.pdf">
       ➥tech_parameterek.pdf
     </SELECT><BR>

     <I
NPUT TYPE=submit NAME="indit"
➥VALUE="Let ltØs ind tÆsa">

</FORM>

</BODY>
</HTML>
```

3. lista A letoltes.php

```
<?

// szÆmlÆl  f ggvØny behœzÆsa
include "szamlalo.inc";

$download_dir = "/home/endre/downloads/";
$letoltendo_fajl = $download_dir.$mit;

if  (file_exists($letoltendo_fajl)) {

     header("Content-Type: application/pdf");
     readfile($letoltendo_fajl);

     // ez a f ggvØny n veli a szÆmlÆl t
     szamlalo_novel($mit);

} else {

     echo "HIBA: A fÆjl nem lØtezik!";

}
```



2.

Tegyük fel, weboldalunkon letölthető pdf formátumú dokumentumokat szeretnénk felkínálni a nyájas közönség számára. Mindezt ráadásul úgy, hogy a letöltéseket számon tarthassuk. Kézenfekvő megoldásnak tűnik a 2. és *3. listán* látható PHP-támogatott letöltő programocska megírása.
Senkinek nem ajánlom, hogy ilyen letöltőrendszert használjon! A gond abban lakozik, hogy a *letoltes.php* feltételezi, hogy az őt meghívó űrlapba senki sem fog belekotorászni. Pedig megteheti. Bárki mentheti az űrlapot tartalmazó oldalt a saját gépére, és kedvére módosíthatja. Ráadásul nincs, aki megakadályozza abban, hogy mondjuk a `mit` nevű bejövő adat értékét erre változtassa: ***../../../../etc/passwd*** (lásd *4. lista*). Letöltendő fájlunk ekkor a ***/home/endre/downloads/../../../../etc/passwd*** lesz. Ez érvényes elérési útvonal, és megegyezik a sima ***/etc/passwd*** hivatkozással. Tegyük fel, hogy jelszavainkat itt, és nem a shadow állományban tároljuk, és a baj már meg is történt. Az alattomos gonosztevő a próbálgató módszerrel máris nekikezdhet a kódolt jelszavak visszafejtésének. De mégha a shadow telepítve is van, e baki által a kiszolgáló felhasználólistája akkor is illetéktelenek kezébe került. Ilyen esetekben a védekezésnek több módja is akad:

- Elfogadható megoldás: a `basename()` függvény segítségével a `$mit` változó elejéről a könyvtárhivatkozásokat – csak a fájlnév megtartásával – levághatjuk.
- Kicsit jobb megoldás: használhatjuk az `ereg()` függvényt, hogy kiszűrjük a `../` részleteket, és a rendszergazdát a letöltés megtagadása mellett levélben értesítsük. A PHP-leírás alapján ilyen `ereg`-szűrésre példa a következő: `ereg('^[^./][^/]*$', $mit)` Ha e kifejezés értéke hamisnak bizonyul, az rettentően gyanús.
- Az elegáns megoldás: az űrlapban nem közvetlenül a fájlnevekkel dolgozunk, hanem minden letölthető fájlhoz azonosítószámot rendelve az azonosítót kérjük be. Ezzel a letölthető anyagok halmazát egyértelműen egy adott fájlcsoportra szűkíthetjük.

4. lista A módosított űrlap

```
<SELECT NAME="mit">
  <OPTION VALUE="dokumentacio.pdf">
  ➥dokumentacio.pdf
    <OPTION VALUE="terkep.pdf">
    ➥terkep.pdf
  <OPTION VALUE="tech_parameterek.pdf">
  ➥tech_parameterek.pdf
  <OPTION VALUE="../../../etc/passwd">
  ➥/etc/passwd
</SELECT><BR>
```

5. lista Űrlap némi hibával

```
<FORM ACTION="change_pw.php" METHOD="POST">

    <B>Jelsz vÆltoztatÆs:</B>
    ➥<?echo$username?><BR>
    <INPUT TYPE="hidden" NAME="user"
    ➥VALUE="<?echo$username?>"><BR>

     j jelsz :<BR>
    <INPUT TYPE="password"
NAME="password1"><BR>

    Jelsz  mØgegyszer:<BR>
    <INPUT TYPE="password"
    ➥NAME="password2"><BR>

    <INPUT TYPE="submit" VALUE="OK">

</FORM>
```

Mindemellett hasznos lehet az űrlapot feldolgozó oldalon a `$HTTP_REFERER` Apache környezeti változót is vizsgálni. Ha ez nem a mi webes űrlapunk címe, a feldolgozást meg lehet tagadni.

Biztonságos adatkezelés

Tegyük fel, hogy egy weboldalon bizonyos szolgáltatásokhoz csak felhasználónév és jelszó bekérése után akarjuk megengedni a látogatók hozzáférését, mindezt ráadásul testreszabott jogosultságokkal. Egy ilyen webes rendszer gyenge pontja a felhasználói adatokat módosító almodul lehet. Tegyük fel, egy „geza” néven belépett felhasználó jelszómódosítás kérése esetén az *5. listán* látható űrlapot kapja.
Itt a „user” nevű rejtett űrlapelemben tárolódik, hogy kinek is a jelszavát kell megváltoztatni. Egy ilyen űrlappal bármelyik bejegyzett felhasználó átveheti egy másik belépőkódját, csak a rejtett mezőben át kell írnia a felhasználónevet. Ez nyilvánvalóan tervezési hiba, de sajnos találkozni lehet vele a weben.
Hogy is kellene ehelyett eljárnunk? Ha http-azonosítást használunk: a `$PHP_AUTH_USER` változóban megtalálható, ki is a belépett felhasználó.
Egy másik elfogadható megoldás, ha munkamenet- (session) kezeléssel egyszerű űrlapon megvalósított beléptetéssel oldjuk meg az azonosítást. A munkamenet-kezelés nemcsak hasznos módszer, de a biztonságos adatkezelésben is hasznos

társ. Ekkor ugyanis elég lapról lapra egyetlen, a kapcsolatunkat meghatározó egyedi azonosítót küldözgetni. Egy ilyen azonosító olyan hosszú és bonyolult karaktersorozat lehet, aminek az ellophatósági (vagyis a címsorban átadott vagy rejtett űrlapelemként beépített azonosítót módosítgatva valaki más kapcsolatazonosítóját kapjuk meg) valószínűsége a nullával egyenlő. Ha egy ilyen „elköthetetlen” kapcsolatazonosítás adott, minden hozzá tartozó adatot – például azt, hogy ki lépett be ezen a kapcsolaton – a kiszolgálón tárolhatunk.
Ekkor bármilyen adatmódosító űrlap esetében nem az űrlapon keresztül kell küldeni, hogy kinek az adatait módosítjuk, hanem az a kapcsolatkiszolgálón tárolt adataiból kiderül.
Jelen bevezető cikkben csak a legalapvetőbb hibák és kivédésük módozatai kerültek terítékre. Soron következő írásomban kicsit mélyebbre ások a PHP és a biztonság kapcsolatában, és a rendszergazdai feladatkörbe tartozó tennivalókra is sor kerül.

Heilig (Cece) Szabolcs (cece@mail.uti.hu) Veszprémben él, huszonhat éves fejjel már hatszoros nagybácsi. Több cégnek dolgozik PHP-programozóként, de PHP-távoktatást is végez. Linuxot először 1994-ben látott, kezdő perles szárnypróbálgatásai után 1997-ben szeretett bele a PHP-be. Szabadidejében hajlamos kerékpárra pattanni, vagy baráti társaságban szerepjátékokkal foglalatoskodni.

Kapcsolódó címek

A PHP-leírás vonatkozó része immár magyarul
➲ http://hu.php.net/security

Összefoglaló a webalkalmazások biztonsági követelményeiről
➲ http://www.owasp.org/projects/requirements/index.shtml (2. kép)

Ismertetés a telepített PHP biztonságossá tételéről
➲ http://www.onlamp.com/pub/a/php/2001/03/29/php_admin.html (3. kép)