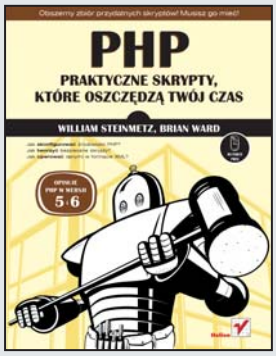
PHP. Praktyczne skrypty, które oszczędzą Twój czas



Autor: William Steinmetz, Brian Ward
Tłumaczenie: Mikołaj Szczepaniak
ISBN: 978-83-246-1851-4
Tytuł oryginału: Wicked Cool PHP: Real-World Scripts That Make Difficult Things Possible
Format: 80x235, stron: 248

Obszerny zbiór przydatnych skryptów! Musisz go mieć!

- Jak skonfigurować środowisko PHP?
- Jak tworzyć bezpieczne skrypty PHP?
- Jakie skrypty musisz znać?

PHP jest łatwym w użyciu językiem skryptowym. Łatwym, a przez to niezwykle popularnym. Jednak, czy ktoś mówił, że w łatwym języku nie można napotkać na skomplikowane problemy? Niestety odpowiedz brzmi – nie. Jednak nie załamuj rąk! Dzięki swej popularności, w sieci istnieje niezliczona liczba stron zawierających informacje, które mogą być przydatne w rozwiązywaniu Twoich problemów.

Ale po co szukać perełek wśród masy kiepskich rozwiązań? Czyż nie lepiej sięgnąć po książkę, która zawierałaby rozwiązania najczęstszych zadań? Oczywiście, że tak. Właśnie taką książkę trzymasz przed sobą! Książka ta zawiera blisko 80 skryptów pozwalających na rozwiązanie najróżniejszych zadań, począwszy od formatowania dat i ciągów znaków, skończywszy na skryptach wykorzystujących pocztę elektroniczną. Dzięki tej książce dowiesz się między innymi, jak tworzyć bezpieczne skrypty oraz pracować z formularzami. Dodatkowo poznasz sposoby konfiguracji samego PHP. Dzięki książce „Praktyczne skrypty, które oszczędzą Twój czas" twoje skrypty będą lepsze, a Ty zyskasz więcej wolnego czasu!

- Zestaw skryptów, które musi znać każdy programista
- Tworzenie szablonów Smarty
- Konfigurowanie środowiska PHP
- Przegląd wszystkich ustawień PHP
- Zastosowanie pakietu SafeHTML
- Zapobieganie atakom XSS
- Zapewnienie bezpieczeństwa w skryptach PHP
- Generowanie losowych haseł
- Praca z formularzami
- Weryfikacja danych z formularza
- Weryfikacja kart kredytowych
- Operacje wykonywane na datach
- Praca z tekstem oraz kodem HTML
- Wykorzystanie plików w codziennej pracy
- Monitorowanie sesji użytkownika
- Mechanizm logowania do aplikacji
- Tworzenie obrazów CAPTCHA
- Operowanie danymi w formacie XML
- Operowanie grafiką

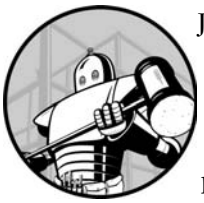
Nie trać czasu na poszukiwanie dobrych rozwiązań! Miej je pod ręką!

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

2

Konfigurowanie PHP



JAK KAŻDY PAKIET OPROGRAMOWANIA, TAK I PHP OFERUJE WIELE OPCJI KONFIGURACYJNYCH, KTÓRE W TEN CZY INNY SPOSÓB WPŁYWAJĄ NA SPOSÓB JEGO FUNKCJONOWANIA. Większość tych opcji nie ma istotnego znaczenia, ale o kilku najważniejszych opcjach każdy programista PHP z pewnością powinien wiedzieć.

Co więcej, istnieje wiele dodatków do PHP (określanych mianem bibliotek) uzupełniających tę technologię o nowe możliwości. Na przykład rozszerzenie cURL umożliwia serwerowi wysyłanie danych formularzy do innych serwerów i przetwarzanie odsyłanych odpowiedzi. Inne przydatne rozszerzenie, Mcrypt, umożliwia nam łatwe i skuteczne szyfrowanie danych celem bezpiecznego składowania poufnych informacji.

W tym rozdziale zajmiemy się ustawieniami konfiguracyjnymi najczęściej wykorzystywanymi przez programistów PHP oraz sytuacjami, w których stosowanie tych ustawień jest uzasadnione.

Ustawienia konfiguracyjne i plik php.ini

Większość początkujących programistów traktuje ustawienia domyślne PHP, jakby byli nieśmiałymi lokatorami wprowadzającymi się do ekskluzywnego apartamentu — obawiają się cokolwiek zmienić w obawie o wpłaconą kaucję. W postrzeganiu PHP jako domu nie ma niczego złego. Będziemy tam jakiś czas mieszkać, dlaczego więc nie poprzestawiać mebli czy nie zburzyć paru ścian?

UWAGA *W zależności od konfiguracji samego komputera, na którym pracuje Twój serwer WWW, możesz nie mieć możliwości samodzielnego modyfikowania ustawień. Dobrzy operatorzy serwerów nie mają jednak nic przeciwko wprowadzaniu niezbędnych zmian w Twoim imieniu, a najlepsze firmy tego typu oferują nawet możliwość modyfikacji ustawień za pośrednictwem specjalnych plików konfiguracyjnych.*

Ustawienia środowiska PHP są składowane w pliku nazwanym *php.ini*, który można przeglądać i modyfikować w dowolnym edytorze tekstu. Ustawienia, które podzielono pomiędzy sekcje, mają następującą postać:

```
max_execution_time = 30     ; Maksymalny czas wykonywania
max_input_time = 60         ; Maksymalny czas analizy składniowej danych wejściowych
memory_limit = 8M           ; Maksymalna ilość pamięci zajmowanej przez skrypt
```

Parametry konfiguracyjne można ustawiać za pomocą znaku równości (=). Średnik (;) oznacza, że mamy do czynienia z komentarzem; okazuje się jednak, że istnieją wyjątki od tej reguły umożliwiające stosowanie średników w niektórych parametrach. Gdybyśmy chcieli trwale zmienić jakieś ustawienie, powinniśmy sporządzić kopię zapasową pliku *php.ini*, zmodyfikować oryginalny plik konfiguracyjny i ponownie uruchomić serwer Apache. Gdybyśmy chcieli zmieniać ustawienia na poziomie skryptu, powinniśmy użyć funkcji `ini_set()`.

Lokalizowanie pliku php.ini

W niektórych przypadkach wskazanie miejsca składowania pliku *php.ini* w systemie, w którym pracujemy (szczególnie jeśli korzystamy z wielu instalacji środowiska PHP), bywa trudne. Poniżej opisano kilka sposobów lokalizowania tego pliku:

- Użytkownicy systemów UNIX powinni zajrzeć do katalogu */usr/lib* lub */usr/local/lib*. Plik *php.ini* powinien się znajdować w podkatalogu *lib* w miejscu, w którym zainstalowano PHP.
- Użytkownicy systemu Windows powinni zwrócić uwagę na katalog *C:\php*.
- Można też wywołać funkcję `phpinfo()` w kodzie skryptu PHP (więcej informacji na ten temat w kolejnym podrozdziale). Lokalizacja pliku *php.ini* zostanie wyświetlona w początkowej części danych wynikowych obok etykiety *Configuration File (php.ini) Location*.
- W wielu systemach UNIX można uzyskać listę wszystkich plików pasujących do wzorca *php.ini* za pomocą polecenia `locate php.ini`.

UWAGA *Wiele ustawień nie jest definiowanych w domyślnym pliku* php.ini*; środowisko PHP stosuje dla niezdefiniowanych ustawień własne wartości domyślne. Listę ustawień domyślnych PHP można znaleźć na stronie internetowej http://www.php.net/*
↪manual/en/ini.php.

Skrypt nr 8: Odkrywanie wszystkich ustawień PHP

PHP oferuje bogatą funkcjonalność, jednak nie zawsze wszystkie te funkcje są włączone lub wbudowane w stosowanej instalacji. Do sprawdzenia, jakie elementy wchodzą w skład danej instalacji środowiska PHP, można wykorzystać bardzo prosty skrypt. Możliwość uzyskiwania tego rodzaju danych jest o tyle niebezpieczna, że szeroki zakres prezentowanych informacji stanowi swoisty podręcznik dla potencjalnych atakujących. Funkcja `phpinfo()` zdaje się mówić: „Tutaj. To są moje słabe punkty. Wprost nie mogę się doczekać włamania do mojego systemu". W tej sytuacji należy pamiętać o konieczności usunięcia tego skryptu zaraz po uzyskaniu interesujących nas informacji:

```
<?php

phpinfo();

?>
```

Funkcja `phpinfo()` wyświetla wszystko, co środowisko PHP „wie" o swojej konfiguracji. Naprawdę wszystko. Zwracane informacje nie ograniczają się tylko do stanu poszczególnych ustawień konfiguracyjnych PHP, położenia pliku *php.ini* czy wersji samego środowiska PHP — obejmują także wersję serwera WWW, skompilowane rozszerzenia oraz dane interfejsu API serwera. Warto zwrócić szczególną uwagę na opcje konfiguracyjne, aby mieć pewność, że wszystkie niezbędne funkcje zostały prawidłowo zainstalowane i włączone.

Aby uruchomić ten skrypt, odwiedź odpowiednią stronę za pomocą swojej przeglądarki internetowej. Nie zapomnij usunąć tego skryptu po uzyskaniu potrzebnych informacji.

Skrypt nr 9: Odczytywanie poszczególnych ustawień

Czasem, kiedy wiemy, czego szukamy, stosowanie funkcji `phpinfo()` jest przesadne i niepotrzebne. Możemy na przykład być zainteresowani tylko sprawdzeniem, czy mechanizm „magicznych cudzysłowów" jest włączony, lub określeniem ścieżki dołączania. Co więcej, funkcja `phpinfo()` w żaden sposób nam nie pomoże, jeśli pisany skrypt zachowuje się inaczej w razie włączenia jakiegoś ustawienia i inaczej w sytuacji, gdy to ustawienie jest wyłączone.

Aby uzyskać wartość określonego ustawienia konfiguracyjnego, należy użyć funkcji `ini_get()`:

```
<?php
echo "Wartość opcji register_globals: " . ini_get('register_globals');
?>
```

Wystarczy przekazać na wejściu funkcji `ini_get()` prawidłową nazwę parametru konfiguracji, a otrzymamy aktualne ustawienie tego parametru na bieżącym serwerze. Opcja jest zwracana w formie zwykłej wartości, zatem można ją wyświetlić, przypisać do zmiennej itd. Korzystając z tej funkcji, musimy jednak mieć na uwadze dwa aspekty.

Po pierwsze: wartości logiczne, np. `"false"`, z reguły są zwracane w formie łańcuchów pustych, zatem jeśli spróbujemy wyświetlić ustawienie `"off"` parametru `register_globals`, być może otrzymamy następujący komunikat:

```
Wartość opcji register_globals:
```

Po drugie: wartości numeryczne często są reprezentowane w formie skróconej. Jeśli na przykład parametrowi `upload_max_filesize` przypisano wartość 8192 bajtów, zostanie zwrócona wartość 8 kB. Podobnie, jeśli maksymalny rozmiar wysyłanego pliku ustalono na poziomie 2 MB, dla parametru `upload_max_filesize` otrzymamy wartość 2 MB, a nie 2 097 152 bajty.

Taki sposób reprezentowania numerycznych ustawień konfiguracyjnych może stanowić poważny problem, jeśli chcemy na tych liczbach wykonywać operacje arytmetyczne. Oficjalna dokumentacja PHP wspomina o funkcji konwertującej wartości skrócone (kilo- i mega-) na prawdziwe wartości:

```
function return_bytes($val) {
   $val = trim($val);
   $last = $val{strlen($val)-1};
   switch(strtoupper($last)) {
      case 'K':
         return (int) $val * 1024;
         break;
      case 'M':
         return (int) $val * 1048576;
         break;
      default:
         return $val;
   }
}
```

Skrypt nr 10: Raportowanie o błędach

Pracując nad kodem, często zapominamy nazw stosowanych zmiennych lub korzystamy z przestarzałych, niezalecanych konstrukcji. W niektórych przypadkach język PHP okazuje się na tyle przyjazny użytkownikowi (przynajmniej jak na standardy programowania), że sam naprawia wiele drobnych błędów w kodzie.

PHP umożliwia nam między innymi pisanie programów bez konieczności deklarowania wszystkich niezbędnych zmiennych na początku kodu, co jest bardzo wygodne, przynajmniej do momentu gdy zamiast nazwy `$string` omyłkowo użyjemy nazwy `$stirng` reprezentującej wartość pustą. Można też przekazywać zmienne na wejściu funkcji w zupełnie niewłaściwy sposób, a mimo to skrypt PHP będzie działał prawidłowo, ponieważ w większości przypadków będzie przyjmował pewne założenia wobec zamiarów programisty. Tego rodzaju mechanizmy są oczywiście bardzo pożądane, dopóki PHP prawidłowo odgaduje nasze intencje — w przeciwnym razie poszukiwanie tajemniczego błędu może nam zająć mnóstwo czasu.

Aby wyłączyć mechanizmy automatycznego usuwania problemów, można włączyć tryb raportowania o błędach, co spowoduje, że PHP będzie wyświetlał na ekranie niezliczone komunikaty w reakcji na każdy wykryty błąd (niezależnie od jego faktycznej wagi). Można te komunikaty wykorzystać do eliminowania potencjalnych luk w zabezpieczeniach i wykrywania błędnych zmiennych przed skierowaniem programu do środowiska końcowego. Włączenie trybu raportowania o błędach wymaga umieszczenia następującego kodu na początku tworzonego skryptu:

```
<?php
error_reporting(E_ALL);
// Tutaj należy umieścić dalszą część skryptu.
?>
```

Włączenie trybu raportowania o błędach powoduje, że PHP wyświetla komunikaty jeszcze przed przetworzeniem dalszej części danego programu. (Takie rozwiązanie uniemożliwia ustawianie ciasteczek w razie wystąpienia błędu, zatem nie powinniśmy nawet próbować zmieniać wartości ciasteczek po włączeniu tego trybu).

Typowe komunikaty o błędach

Warto dobrze opanować i zrozumieć trzy najczęściej generowane komunikaty o błędach.

```
Notice: Undefined variable: var in script.php on line n
```

Komunikat w tej formie oznacza, że korzystamy ze zmiennej, której wcześniej nie zdefiniowano w danym skrypcie. Taka sytuacja może mieć miejsce w kilku przypadkach:

- Być może popełniliśmy błąd w pisowni nazwy zmiennej.
- Być może użyliśmy wyrażenia warunkowego zawierającego definicję zmiennej, np.:

```
if ($fred == "Jestem Fred") {
    $he_is_fred = "yes";
}
```

- Być może próbujemy konkatenować zmienną bez jej uprzedniego zadeklarowania.

Inny popularny problem występuje dużo częściej w sytuacji, gdy w swoim programie próbujemy korzystać ze starszego kodu PHP:

```
Notice: Use of undefined constant k - assumed 'k' in script.php on line n
```

Komunikat ostrzeżenia w tej formie zwykle oznacza, że programista podjął próbę przekazania łańcucha na wejściu funkcji bez otaczających go cudzysłowów. Innymi słowy, użyto na przykład wywołania `strtolower(`*`łańcuch`*`)` zamiast metody `strtolower("`*`łańcuch`*`")`.

I wreszcie istnieje popularny komunikat o błędzie generowany w sytuacji, gdy uzyskujemy dostęp do tablicy:

```
Notice: Undefined index: i in script.php on line n
```

W praktyce komunikat w tej formie oznacza, że podjęto próbę odczytania elementu `$`*`tablica`*`[`*`i`*`]`, mimo że tablica `$`*`tablica`* nie definiuje elementu pod tym indeksem. Z tego rodzaju błędami mamy do czynienia w sytuacji, gdy uzyskujemy wartość z formularza za pośrednictwem zmiennej `$_POST` lub `$_GET`, chociaż żadna z tych zmiennych nie zawiera tak nazwanej wartości. Najczęściej podobne błędy wynikają z tego, że użytkownik nie zaznaczył odpowiedniego pola wyboru lub przycisku opcji — w takim przypadku zmienna reprezentująca ten element formularza w ogóle nie jest przekazywana w ramach żądania `GET` (jako część adresu URL).

Tryb raportowania o błędach należy wyłączyć z chwilą wdrażania skryptu na docelowej witrynie, aby użytkownicy nie mogli się zapoznawać z popełnionymi przez nas błędami i aby wyeliminować wpływ tego trybu na ciasteczka (w szczególności problemy ze śledzeniem sesji).

Skrypt nr 11: Ukrywanie wszystkich komunikatów o błędach

W pewnych sytuacjach dysponujemy prawidłowo działającym skryptem, a mimo to środowisko PHP wciąż sugeruje potencjalne usterki. Innym razem nie chcemy, by oczekiwane problemy powodowały, że nasi użytkownicy będą narażeni na odrażający widok komunikatów o błędach (odkrywających informacje, co szczególnie cenią sobie hakerzy).

Na szczęście istnieje możliwość powstrzymania PHP przed wyświetlaniem wszystkich komunikatów o błędach. Wystarczy w pliku *php.ini* umieścić następujący wiersz:

```
display_errors = Off
```

Przytoczone rozwiązanie warto stosować w środowisku docelowym aplikacji internetowej, aby w przyszłości nie obawiać się szerokiej dostępności komunikatów diagnostycznych PHP odnośnie do naszego kodu. Gdybyśmy chcieli zapoznać się z tymi komunikatami celem wyeliminowania ewentualnych problemów, powinniśmy użyć następującego ustawienia konfiguracyjnego wymuszającego kierowanie tych komunikatów do dziennika zdarzeń serwera Apache:

```
log_errors = On
```

W razie potrzeby można nawet wysyłać komunikaty diagnostyczne do dziennika systemowego lub wskazanego pliku — parametrowi `error_log` należy wówczas przypisać odpowiednio wartość `syslog` lub nazwę pliku.

Pozostaje jeszcze kwestia naszego środowiska wytwarzania, gdzie z reguły chcemy uzyskiwać możliwie wiele komunikatów diagnostycznych. Po przypisaniu parametrowi `display_errors` wartości `On` można dodatkowo (w pliku *php.ini*) ustawić strukturę bitową w parametrze `error_reporting` (więcej szczegółów na ten temat można znaleźć w przykładowym pliku *php.ini* instalowanym wraz ze środowiskiem PHP). Jeśli jednak chcemy „uciszyć" jakiś skrypt, który nieustannie zasypuje nas tymi samymi komunikatami, możemy użyć w jego kodzie następującego wywołania funkcji:

```
error_reporting(0);
```

Skrypt nr 12: Wydłużanie czasu wykonywania skryptu

Pracowałem kiedyś w firmie, która postawiła sobie za cel zmianę mechanizmu obsługi koszyków z zakupami. Do moich zadań należało napisanie skryptu odpowiedzialnego za konwersję 250 MB danych o produktach ze starego na nowy format. Skrypt działał co prawda znakomicie, jednak ilość przetwarzanych danych powodowała, że środowisko PHP stale przerywało jego wykonywanie po upływie 30 sekund, a więc na długo przed osiągnięciem zamierzonego celu.

Właśnie wówczas odkryłem drobne rozszerzenie, które umożliwiło mojemu skryptowi wykonanie zleconego zadania. Poniższy wiersz dodany na początku skryptu powoduje, że będzie on miał maksymalnie 240 sekund na przetworzenie danych:

```
ini_set(max_execution_time, "240");
```

Parametr konfiguracyjny `max_execution_time` określa maksymalny czas wykonywania skryptu przed jego automatycznym zakończeniem. Nie należy jednak tego parametru nadużywać. Jeśli dany skrypt działa kilka minut, to albo usprawiedliwia nas ogromna ilość przetwarzanych informacji (najpewniej zaczerpniętych z bazy danych), albo nasz skrypt jest bardzo nieefektywny, albo korzystamy z niewłaściwego języka programowania.

Co może pójść nie po naszej myśli?

Jeśli nasz serwer pracuje w trybie awaryjnym, ustawianie wartości parametru `max_execution_time` w czasie wykonywania jest niemożliwe.

Warto też dokładnie sprawdzić kod skryptu. Być może omyłkowo zawarliśmy tam nieskończoną pętlę lub pętlę wykonywaną w innej pętli i niepodejmującą żadnych sensownych działań.

Skrypt nr 13: Uniemożliwianie użytkownikom wysyłania wielkich plików

Gdybyśmy chcieli uniemożliwić użytkownikom naszej aplikacji wysyłanie na serwer 70-gigabajtowych MPEG-ów z najnowszym filmem „Gwiezdne wojny", powinniśmy określić maksymalny rozmiar plików kopiowanych na serwer. (Szczegółowe omówienie samych technik przetwarzania wysyłanych plików można znaleźć w podrozdziale „Skrypt nr 54: Wysyłanie obrazów do katalogu" w rozdziale 7.).

```
upload_max_filesize = 500K
```

Maksymalny rozmiar plików wysyłanych na serwer można określić na jeden z trzech sposobów:

- w formie wartości całkowitoliczbowej (wyrażającej łączną liczbę bajtów);
- w formie liczby z przyrostkiem `M` reprezentującej megabajty (`2M` to 2 megabajty);
- w formie liczby z przyrostkiem `K` reprezentującej kilobajty (`8K` to 8 kilobajtów).

Niezależnie od użytego formatu nasi użytkownicy nie będą mogli wysłać na serwer pliku, którego rozmiar będzie przekraczał tak zdefiniowany próg. Domyślnym rozmiarem maksymalnym są 2 MB.

Skrypt nr 14: Wyłączanie rejestrowanych zmiennych globalnych

Język PHP oferuje przestarzałą, niezalecaną funkcję, która nieznacznie ułatwia dostęp do parametrów żądań `GET` i `POST` protokołu HTTP. Jeśli na przykład żądanie `POST` zawiera parametr nazwany `mojparam`, PHP może automatycznie umieścić jego wartość w zmiennej nazwanej `$mojparam`. Działanie tego mechanizmu stwarza poważne ryzyko dla bezpieczeństwa aplikacji, ponieważ umożliwia użytkownikom ustawianie dowolnych zmiennych globalnych — jeśli zapomnimy zainicjalizować odpowiednie zmienne, użytkownik zyska możliwość wpływania na istotne elementy naszego skryptu.

Wspomniany mechanizm można wyłączyć, ustawiając w zmiennej `register_globals` wartość `Off` w pliku *php.ini* serwera:

```
register_globals = Off
```

Opisana funkcja na szczęście została wyłączona w wersjach 4.2 i nowszych języka PHP. Waga problemu jest jednak na tyle duża, że warto to dwukrotnie sprawdzić.

Skrypt nr 15: Włączanie „magicznych cudzysłowów”

„Magiczne cudzysłowy” (ang. *magic quotes*) to wygodne narzędzie stosowane przez administratorów serwerów do ochrony przed atakami polegającymi na wstrzykiwaniu kodu SQL-a (patrz podrozdział „Skrypt nr 19: Wstrzykiwanie kodu języka SQL” w rozdziale 3.). Działanie tego mechanizmu polega na po-

przedzaniu wszystkich apostrofów, cudzysłowów i lewych ukośników dodatkowym znakiem lewego ukośnika (tzw. znakiem ucieczki) we wszystkich danych zapisywanych w zmiennych skryptu PHP i pochodzących z formularzy HTML. W ten sposób na przykład łańcuch `"Ferrett's Book"` zostanie przekształcony w łańcuch `\"Ferrett\'s Book\"`.

Mechanizm „magicznych cudzysłowów" nie jest rozwiązaniem idealnym, jeśli korzystamy z bazy danych MySQL — w takim przypadku należy stosować raczej wyspecjalizowaną funkcję `mysql_real_escape_string()` — jednak generalnie „magiczne cudzysłowy" zdają egzamin. Można ten mechanizm włączyć w pliku *php.ini* za pomocą następującego wyrażenia:

```
magic_quotes_gpc = 1
```

Co może pójść nie po naszej myśli?

Jeśli nie włączymy mechanizmu „magicznych cudzysłowów", będziemy musieli korzystać z funkcji `mysql_real_escape_string()`, aby zagwarantować stosowanie sekwencji ucieczki w wykorzystywanych danych. Jeśli jednak użyjemy tej funkcji dla danych w sytuacji, gdy mechanizm „magicznych cudzysłowów" będzie włączony, ryzykowne znaki zostaną poprzedzone podwójnymi lewymi ukośnikami (zamiast `\"Ferrett\'s Book\"` otrzymamy `\\"Ferrett\\'s Book\\"`). Jak widać, konsekwencja popłaca — chwila nieuwagi może spowodować, że tabele naszej bazy danych będą zawierały niemal wyłącznie lewe ukośniki.

Skrypt nr 16: Ograniczanie dostępu PHP do plików

Jeśli obawiasz się wrogiego skryptu PHP uzyskującego dostęp do plików systemowych (np. do pliku haseł), możesz użyć ustawienia `open_basedir` do ograniczenia zbioru katalogów dostępnych z poziomu kodu PHP. Po ustawieniu tej opcji skrypt PHP nie będzie mógł otwierać ani modyfikować żadnych plików spoza wskazanego katalogu. Poniżej przedstawiono wiersz pliku *php.ini* ograniczający dostęp tylko do katalogu */home/www*:

```
open_basedir = /home/www
```

Istnieje możliwość zapewniania skryptom dostępu do wielu katalogów — w systemie UNIX należy je oddzielać dwukropkami (`:`); w systemie Windows kolejne katalogi oddzielamy średnikami (`;`).

UWAGA *PHP domyślnie zapewnia dostęp zarówno do wskazanego katalogu, jak i wszystkich jego podkatalogów. Gdybyśmy chcieli ograniczyć ten dostęp tylko do plików w określonym katalogu, na końcu użytej ścieżki powinniśmy użyć ukośnika (np. /home/www/).*

Co może pójść nie po naszej myśli?

Jeśli użytkownicy muszą wysyłać pliki na serwer, to do czasu ich przetworzenia przez skrypt otrzymane pliki są składowane w katalogu tymczasowym. Ponieważ katalog tymczasowy z reguły dzieli spora odległość od pozostałych plików PHP, koniecznie musimy pamiętać o jego uwzględnieniu na liście reprezentowanej przez parametr `open_basedir`.

Skrypt nr 17: Wyłączanie obsługi określonych funkcji

Przypuśćmy, że uznaliśmy funkcję `exec()`, która umożliwia bezpośrednie wykonywanie poleceń na serwerze z poziomu skryptów PHP, za zbyt niebezpieczną. Okazuje się, że istnieje możliwość wyłączania obsługi poszczególnych funkcji PHP (właśnie z myślą o wyeliminowaniu luk w zabezpieczeniach) z zachowaniem możliwości stosowania wszystkich pozostałych funkcji. Poniżej przedstawiono przykład wiersza pliku *php.ini* wyłączającego obsługę kilku szczególnie ryzykownych funkcji:

```
disable_functions = system, exec, passthru, shell_exec, proc_open
```

Skrypt nr 18: Dodawanie rozszerzeń do PHP

Naprawdę poważni programiści prędzej czy później odkrywają pewne ograniczenia języka PHP. Mimo ogromnej liczby wbudowanych funkcji i mechanizmów sam język PHP nie oferuje rdzennych rozwiązań w zakresie szyfrowania, grafiki, dostępu do innych stron internetowych czy przetwarzania danych w formacie XML.

Te i inne cele można jednak osiągać dzięki niezliczonym rozszerzeniom wykorzystującym biblioteki tworzone przez niezależnych programistów i podmioty. Kilka najbardziej przydatnych rozszerzeń języka PHP opisano poniżej:

cURL

cURL umożliwia naszemu serwerowi PHP uzyskiwanie dostępu do innych witryn internetowych, w tym wysyłanie i odbieranie informacji za pośrednictwem swoistego protokołu zbudowanego na bazie adresów URL. (Najczęściej korzystamy z protokołu HTTP, który umożliwia nam komunikację z innymi stronami internetowymi, oraz protokołu FTP umożliwiającego nam wysyłanie i pobieranie plików). W praktyce oznacza to, że nasz serwer może być traktowany przez inne witryny jak przeglądarka internetowa, a pobierane strony WWW możemy umieszczać w dowolnych zmiennych w ramach swoich skryptów.

cURL jest niezwykle ważnym narzędziem dla programistów pracujących nad poważnymi sklepami internetowymi, ponieważ umożliwia nam akceptowanie płatności kartami kredytowymi i wyceny towarów dla poszczególnych klientów w czasie rzeczywistym. Za pomocą rozszerzenia cURL można nawiązywać połączenia i wysyłać dane o transakcjach na serwer innej firmy. W odpowiedzi otrzymujemy wówczas informacje o akceptacji bądź odrzuceniu żądania dokonania płatności.

Mcrypt

Musiałeś kiedyś coś zaszyfrować? Wszystkie poufne informacje umieszczane w ciasteczkach i sesjach powinny być szyfrowane. Co więcej, jeśli gdziekolwiek zapisujemy coś naprawdę wartościowego, jak numery kart kredytowych czy dane osobowe, *z pewnością* powinniśmy się upewnić, że odczyt tych informacji nie będzie możliwy przez zwykły zrzut zawartości bazy danych. Na szczęście biblioteka Mcrypt umożliwia nam naprawdę skuteczne szyfrowanie danych bez choćby szczątkowej znajomości technik szyfrowania! (Sposoby korzystania z tego rozszerzenia zostaną szczegółowo omówione w podrozdziale „Skrypt nr 23: Szyfrowanie danych za pomocą rozszerzenia Mcrypt" w rozdziale 3.).

GD

Gdybyśmy chcieli tworzyć obrazy graficzne na żądanie lub po prostu uzyskiwać szczegółowe informacje o obrazach już istniejących, powinniśmy się zapoznać z możliwościami biblioteki GD. Biblioteka GD umożliwia nam pracę na plikach JPEG i GIF — możemy je tworzyć z myślą o graficznej prezentacji rozmaitych danych (np. w formie wykresów) albo modyfikować (np. tworząc miniaturki istniejących obrazów).

MySQL

Podstawowa wersja środowiska PHP w ogóle „nie wie", jak uzyskiwać dostęp do baz danych. Ponieważ jednak system MySQL i język PHP są jak Zan i Jayna z popularnej kreskówki, większość serwerów WWW przystosowanych do obsługi PHP oferuje też domyślnie instalowane biblioteki systemu MySQL, zatem większość programistów korzysta z funkcji `mysql_connect()`, nie wiedząc, że jest ona częścią rozszerzenia.

Zbiór rozszerzeń PHP jest oczywiście dużo bogatszy i obejmuje takie biblioteki jak SOAP (zapewniająca dostęp do usług internetowych), PDF czy Verisign Payment Pro. Na pierwszy rzut oka może się wydawać, że najlepszym rozwiązaniem jest dołączanie do środowiska PHP wszystkich rozszerzeń, które tylko udało nam się odnaleźć, jednak warto mieć na uwadze, że każde z nich może wydłużyć czas inicjalizacji i stwarzać dodatkowe luki w zabezpieczeniach. Co więcej, mniej popularne rozszerzenia z reguły nie są na bieżąco aktualizowane ani rozwijane.

Dodawanie rozszerzeń języka PHP

Skoro wiemy już, co można zyskać, instalując rozszerzenia, przyjrzyjmy się samem procesowi ich instalowania. W pierwszej kolejności należy sprawdzić, czy przypadkiem już nie dysponujemy tym, czego szukamy.

Sprawdzanie, czy interesujące nas rozszerzenia nie zostały już załadowane

Wiele serwerów WWW domyślnie instaluje najbardziej przydatne i najpopularniejsze rozszerzenia, zatem przed podjęciem prób odszukania i instalacji interesującej nas biblioteki powinniśmy się upewnić, czy nie została już zainstalowana i załadowana.

Najprostszym sposobem sprawdzenia ewentualnej obecności rozszerzeń jest wywołanie funkcji `phpinfo()` (opisanej w podrozdziale „Skrypt nr 8: Odkrywanie wszystkich ustawień PHP" we wcześniejszej części tego rozdziału). Listę zwróconą przez tę funkcję należy dokładnie przejrzeć w poszukiwaniu naszych bibliotek. Jeśli na przykład środowisko PHP obejmuje zainstalowane rozszerzenie MySQL, dane wynikowe funkcji `phpinfo()` będą zawierały wiersz podobny do poniższego:

```
mysql

MySQL Support => enabled
...
```

Jeśli uznasz, że takie rozwiązanie nie jest dla Ciebie, jeśli wyda Ci się zbyt wolne, możesz skorzystać z innych możliwości.

Każde rozszerzenie dodaje do PHP nowe funkcje — na przykład cURL uzupełnia funkcjonalność PHP o takie funkcje jak `cURL_init()` czy `cURL_setopt()`, Mcrypt dodaje funkcje `mcrypt_encrypt()` oraz `mcrypt_decrypt()` itd. Przypuśćmy jednak, że nie zainstalowano rozszerzenia Mcrypt. W takim przypadku PHP nie ma pojęcia o funkcji `mcrypt_decrypt()` i traktuje ją jako funkcję niezdefiniowaną.

Można to wykorzystać, stosując funkcję `function_exists()` języka PHP. Poniżej przedstawiono przykładowy skrypt wykrywający rozszerzenie MySQL:

```
<?php
if (function_exists(mysql_connect)) {
   print 'Wykryto rozszerzenie MySQL';
} else {
   print 'Nie wykryto rozszerzenia MySQL';
}
?>
```

Ładowanie rozszerzeń przy pomocy administratorów zdalnych serwerów

Jeśli korzystamy z serwera WWW będącego własnością innej firmy (tak robi większość programistów), musimy się zdać na łaskę administratora tego serwera. Ponieważ z natury rzeczy nie dysponujemy hasłem administratora, nie

możemy instalować niezbędnych bibliotek samodzielnie. W takim przypadku musimy o to poprosić administratora wynajmowanego serwera. Kierując odpowiednie zlecenie, powinniśmy się upewnić, że administrator dysponuje precyzyjnymi informacjami; w przeciwnym razie może się okazać, że została zainstalowana niewłaściwa wersja lub wręcz niewłaściwe rozszerzenie.

Niektóre firmy zrealizują naszą prośbę bez najmniejszych problemów. Inne będą oczekiwały dodatkowych opłat za obciążanie swoich serwerów dodatkowymi rozszerzeniami. Jeszcze inne odpowiedzą: „Nasza oferta nie obejmuje obsługi dodatkowych rozszerzeń. Ograniczamy się tylko do standardowego PHP".

Jeśli z jakiegoś powodu nie możesz zainstalować potrzebnych rozszerzeń, możesz albo spróbować poradzić sobie bez nich, albo zmienić firmę obsługującą serwery.

UWAGA *Nawet jeśli korzystamy z własnego serwera, ale nie potrafimy prawidłowo zainstalować niezbędnych rozszerzeń, warto zwrócić się z prośbą o instalację nowych bibliotek do pracowników wsparcia technicznego. W takim przypadku w razie niepowodzenia procesu instalacji technicy będą w stanie naprawić usterkę (przynajmniej teoretycznie).*

Instalacja rozszerzeń za pomocą internetowego panelu sterowania

Dzierżawione serwery często oferują specjalne panele sterowania, za pośrednictwem których możemy realizować typowe zadania administracyjne (w tym zadanie ponownego uruchomienia usługi Apache lub restartu całego serwera) w oknie przeglądarki internetowej.

Niektóre panele sterowania oferują nawet możliwość automatycznego kompilowania serwera Apache i środowiska PHP wskutek zaznaczenia pól wyboru lub wyboru z list rozwijanych opcji reprezentujących dodawane rozszerzenia. Na przykład WHM (popularny, choć dość trudny w obsłudze panel sterowania) udostępnia opcję *Update Apache*, która powoduje ponowną instalację serwera Apache i środowiska PHP wraz z wybranymi żądaniami.

Jeśli Twój serwer nie udostępnia preinstalowanego panelu sterowania, z reguły za niewielką opłatą można taki panel zainstalować już po wdrożeniu oprogramowania serwera.

Ręczna instalacja rozszerzeń

Ponowna kompilacja PHP jest w systemach UNIX traktowana jako ponowna instalacja tego środowiska wraz z niezbędnymi rozszerzeniami. Dla programistów, którzy nie mają doświadczenia w roli administratorów systemów UNIX, ponowna kompilacja środowiska PHP często jest poważnym wyzwaniem.

Najlepszym rozwiązaniem jest przystąpienie do eksperymentów z lokalnym serwerem Apache z dala od docelowego środowiska pracy aplikacji internetowej. Ponieważ zmiany wprowadzane w konfiguracji pracującego serwera mogą

doprowadzić do poważnych problemów, warto uprzednio sprawdzić, czy w razie kłopotów możemy liczyć na pomoc techniczną i czy zdajemy sobie sprawę z tego, co może się wydarzyć. Jeśli nie jesteśmy przygotowani do tego rodzaju zadań, powinniśmy się zwrócić o pomoc do kogoś bardziej kompetentnego.

Instalacja biblioteki w środowisku PHP jest procesem dwuetapowym — w pierwszej kolejności musimy zainstalować same biblioteki rozszerzeń; drugim krokiem jest takie skonfigurowanie środowiska PHP, aby rozpoznawało te rozszerzenia.

Instalowanie bibliotek

Szczegółowe kroki składające się na proces instalacji rozszerzenia w dużej mierze zależą od dodawanej biblioteki. Można oczywiście sformułować ogólne zasady rządzące tym procesem, jednak przed przystąpieniem do instalacji zawsze należy się zapoznać z podręcznikami dostępnymi na stronie biblioteki oraz wszystkimi plikami *README*. Czytelnicy zainteresowani szczegółowym wyjaśnieniem pracy systemu Linux, w tym technik kompilowania oprogramowania, powinni sięgnąć po książkę Briana Warda zatytułowaną *How Linux Works* (No Starch Press, 2004)[1].

W poniższych punktach opisano ogólne kroki składające się na proces instalacji bibliotek:

1. **Zalogowanie na serwerze jako administrator lub użytkownik z prawem instalacji nowych programów.**

2. **Pobranie archiwum biblioteki i umieszczenie go w katalogu głównym serwera.** Wpisanie w wyszukiwarce *Google* nazwy biblioteki i słowa *PHP* (np. *mcrypt php*) z reguły pozwoli błyskawicznie odnaleźć stronę domową interesującego nas rozszerzenia, gdzie będą dostępne odpowiednie pliki źródłowe. Pliki źródłowe najczęściej są archiwizowane i kompresowane za pomocą narzędzi Gzip i tar (z myślą o oszczędzaniu przestrzeni), zatem pobrany plik najprawdopodobniej będzie nosił nazwę *nazwaplikubiblioteki.tar.gz*.

3. **Wypakowanie zawartości pobranego archiwum.** Archiwum tar jest w istocie zbiorem plików i katalogów. Całe to archiwum jest następnie kompresowane za pomocą pakietu Gzip, stąd rozszerzenie *.gz* jest dopisywane na samym końcu. Oznacza to, że plik *.tar.gz* można traktować tak samo jak plik *.zip*, z tą różnicą, że plik *.tar.gz* powstaje w dwóch etapach i z wykorzystaniem dwóch różnych programów.

   Okazuje się jednak, że nie musimy wprost uruchamiać obu tych narzędzi, ponieważ program tar w wersji GNU „wie", jak korzystać z narzędzia dekompresującego. Wypakowanie zawartości archiwum tar wymaga wydania polecenia **`tar zxvf nazwaplikubiblioteki.tar.gz`** w wierszu poleceń. W wyniku tego polecania otrzymamy listę wypakowanych wszystkich plików i katalogów.

[1] Polskie wydanie: *Jak działa Linux*, Helion, 2005 — *przyp. tłum.*

Większość archiwów tworzy drzewo poniżej katalogu najwyższego poziomu, zatem właśnie takiej struktury powinniśmy oczekiwać.

4. **Przejście do katalogu biblioteki za pomocą polecenia `cd nazwakatalogu`.** Jeśli nie pamiętamy lub w ogóle przeoczyliśmy nazwę z poprzedniego kroku, z reguły możemy przyjąć, że nazwa tego katalogu będzie odpowiadała nazwie samej biblioteki — np. **`cd cURL`**. (Warto pamiętać, że wielkość znaków w nazwach katalogów jest istotna, zatem *cURL* to nie to samo co *CURL*).

5. **Uruchomienie polecenia `configure`, aby sprawdzić, czy wszystkie składniki niezbędne do zakończenia instalacji na danym komputerze zostały rozpakowane.** Z uwagi na różnorodność dostępnych systemów UNIX instalacja pakietów w tych systemach wymaga pewnej wiedzy i doświadczenia. Na szczęście polecenie `configure` w większości przypadków potrafi wykonać całą tę „brudną robotę" za nas, automatycznie analizując ustawienia serwera i stosując wartości umożliwiające prawidłowy przebieg instalacji programu. Wpisz w wierszu poleceń wyrażenie **`./configure`**.

   Niektóre rozszerzenia wymagają do właściwego działania stosowania dodatkowych flag za poleceniem `configure`. Na przykład rozszerzenie Mcrypt wymaga od nas wydania polecenia `./configure --disable-nls --disable-posix-threads`, ponieważ tylko w ten sposób można zagwarantować pełną zgodność z serwerem Apache. Ponieważ dodatkowe opcje zależą od samej biblioteki, warto się zapoznać z przewodnikami i plikami *README*, gdzie można znaleźć precyzyjną dokumentację wszystkich niezbędnych flag konfiguracyjnych.

6. **Kompilacja i instalacja danego pakietu.** W systemach UNIX standardowym narzędziem kompilującym i instalującym pakiety jest `make`. Najpierw musimy wydać właśnie polecenie **`make`**, aby skompilować nasz pakiet. Na ekranie zostaną wyświetlone wykonywane polecenia, które składają się na proces kompilacji. Następnie powinniśmy użyć polecenia **`make check`** celem wykonania na tym pakiecie automatycznych testów (niektóre pakiety nie zawierają testów, czym jednak nie powinniśmy się przejmować). I wreszcie należy wpisać polecenie **`make install`**, aby ostatecznie zainstalować rozszerzenie. Także proces instalacji będzie dokumentowany na ekranie. Po wykonaniu polecenia **`make install`** proces instalacji rozszerzenia będzie zakończony.

7. **Utworzenie skryptu `phpinfo()`.** Ach, pewnie myślałeś, że to już koniec, prawda? Przykro mi, ale opisana powyżej procedura prowadzi tylko do instalacji rozszerzenia na serwerze. Musimy jeszcze ponownie zainstalować środowisko PHP i wskazać, gdzie znajduje się nowe rozszerzenie i jak z niego korzystać.

   Za pomocą funkcji `phpinfo()` (patrz podrozdział „Skrypt nr 8: Odkrywanie wszystkich ustawień PHP" we wcześniejszej części tego rozdziału) możemy uzyskać kompletny wykaz ustawień serwera. Gdzieś na początku pierwszej

strony danych wygenerowanych przez tę funkcję można znaleźć sekcję zatytułowaną *Configure Command* i zawierającą tajemniczą listę elementów podobną do poniższej:

```
'./configure' '--with-apxs=/usr/local/apache/bin/apxs' '--with-xml'
↪'--enable-bcmath' '--enable-calendar' '--enable-ftp' '--enable-magic-
↪quotes' '--with-mysql' '--enable-discard-path' '--with-pear' '--enable-
↪sockets' '--enable-track-vars' '--enable-versioning' '--with-zlib'
```

Gdybyśmy chcieli ponownie zainstalować środowisko PHP w stanie, w którym znajduje się obecnie, dysponowalibyśmy gotowym poleceniem (a przynajmniej prawie gotowym). W pierwszej kolejności należy usunąć apostrofy wokół polecenia `configure`, aby otrzymać polecenie w postaci:

```
./configure '--with-apxs=/usr/local/apache/bin/apxs' '--with-xml'
↪'--enable-bcmath' '--enable-calendar' '--enable-ftp' '--enable-magic-
↪quotes' '--with-mysql' '--enable-discard-path' '--with-pear' '--enable-
↪sockets' '--enable-track-vars' '--enable-versioning' '--with-zlib'
```

Celem tego kroku jest zachowanie już zainstalowanych rozszerzeń — jeśli dodajemy rozszerzenie GD, nie chcemy przecież przy tej okazji utracić innych, zainstalowanych wcześniej rozszerzeń. Gotowe polecenie `configure` należy skopiować do pliku tekstowego i dopisać na jego końcu odpowiednie wyrażenia `--with`. Jeśli na przykład dodajemy do serwera rozszerzenie Mcrypt, powinniśmy dopisać wyrażenie `--with-mcrypt`. Właściwy parametr `--with` z reguły można znaleźć w dokumentacji instalowanego rozszerzenia.

UWAGA *Jeśli zastąpimy oryginalną strukturę katalogów zawartą w pliku tar i umieścimy naszą bibliotekę w folderze innym niż domyślny, będziemy musieli dodać do flagi `--with` ścieżkę, aby środowisko PHP mogło tę bibliotekę odnaleźć. W powyższym przykładzie taka sytuacja miała miejsce w przypadku biblioteki apxs (Apache Extension Tool Synopsis), gdzie flaga `--with-apxs=/usr/local/apache/bin/apxs` określała, że wspomniana biblioteka jest składowana w katalogu /usr/local/apache/bin/apxs.*

8. **Pobranie i rozpakowanie plików źródłowych nowej dystrybucji PHP oraz przejście do katalogu, w którym umieszczono rozpakowane pliki.** Kod źródłowy PHP można rozpakować dokładnie tak, jak wcześniej rozpakowaliśmy kod źródłowy biblioteki. Jeśli dysponujesz już utworzonym wcześniej drzewem kodu PHP, możesz je wykorzystać, jednak w takim przypadku koniecznie użyj polecenia `make clean`.
9. **Skopiowanie polecenia `configure` utworzonego wcześniej w pliku tekstowym, wklejenie go w wierszu poleceń i naciśnięcie klawisza *Enter*, aby je wykonać.** W ten sposób ponownie skonfigurujemy środowisko PHP z nową biblioteką i wszystkimi dotychczasowymi bibliotekami.

10. **Kompilacja kodu źródłowego PHP.** Należy wykonać kolejno polecenia `make` i `make install`. Warto się przygotować na długie oczekiwanie, aż wymienione polecenia odpowiednio skompilują i zainstalują wszystkie komponenty PHP.

UWAGA *W razie dokonania jakichkolwiek zmian w plikach .ini (podobnych do tych pokazanych we wcześniejszej części tego rozdziału) wprowadzone modyfikacje mogą zostać nadpisane ustawieniami domyślnymi w czasie ponownego kompilowania PHP. W takim przypadku warto do tych ustawień wrócić, aby mieć pewność, że nasza konfiguracja nie została zmieniona.*

11. **Ponowne uruchomienie serwera Apache.** Należy wykonać polecenie `apachectl graceful`.

12. **Przetestowanie środowiska PHP.** W pierwszej kolejności warto uruchomić skrypt *Witaj świecie!*, aby sprawdzić, czy właśnie zainstalowane środowisko działa właściwie. Następnie dobrze jest poeksperymentować z wywołaniami rozmaitych funkcji definiowanych przez biblioteki, aby mieć pewność, że także nowe biblioteki nie stwarzają niespodziewanych problemów.

Co może pójść nie po naszej myśli?

Liczba problemów, które mogą wystąpić w czasie kompilacji, jest tak długa, że omówienie ich wszystkich jest niemal niemożliwe. Chociaż wiele błędów jest dość skomplikowanych, a znaczna ich część jest ściśle związana z poszczególnymi bibliotekami (i tym samym wymaga bardzo specjalistycznych porad), trzy typowe problemy występują niemal zawsze.

Pierwszym poważnym problemem, z którym możemy się zetknąć, jest brak zainstalowanych pakietów wytwarzania oprogramowania w pobranej dystrybucji lub wersji. W takim przypadku będziemy potrzebowali kompilatora języka C i rozmaitych wersji „deweloperskich" wielu innych bibliotek potrzebnych do skompilowania kodu.

Po drugie, możemy stanąć przed koniecznością skonfigurowania środowiska PHP z wykorzystaniem parametru `--with` definiującego wprost ścieżkę do dołączanej biblioteki, np. `--with-mcrypt=/usr/lib/mcrypt`.

Innym powszechnym źródłem problemów są źle skonfigurowane pakiety bibliotek rozszerzeń. Jak już wspomniano, bibliotekę Mcrypt należy skonfigurować z flagami `--disable-nls --disable-posix-threads`; w przeciwnym razie stosowanie tego pakietu może prowadzić nawet do awarii serwera Apache. Także inne biblioteki wymagają do prawidłowego funkcjonowania w środowisku PHP i na serwerze Apache pewnych dodatkowych zabiegów. Szczegółowych informacji na ten temat należy szukać na stronach internetowych z najczęściej zadawanymi pytaniami, stronach pomocy systemowej oraz w plikach *README*.