

## Bezpieczeństwo PHP

*Paweł Krawczyk*

W ciągu kilku ostatnich miesięcy opublikowano kilkadziesiąt dziur w aplikacjach napisanych w PHP oraz kilka w samym interpreterze tego języka. Czy to oznacza że PHP jest dziurawe i nie należy go używać? Na szczęście nie jest tak źle – PHP jest przede wszystkim nowoczesnym i bardzo bogatym językiem programowania aplikacji webowych w którym, jak we wszystkich językach należy przestrzegać pewnych „zasad BHP”.

Język PHP w pewnym stopniu padł ofiarą własnej popularności. Sam interpreter ewoluując w ciągu ostatnich kilka lat od prostego języka skryptowego PHP/FI zyskał obiektywność, automatyczną alokację pamięci pod zmienne, uniwersalne odnośniki do danych zewnętrznych oraz – co najważniejsze dla autora aplikacji webowych – dziesiątki rozszerzeń, przekładających się na setki funkcji o nieraz skomplikowanej składni, pisanych przez różnych autorów dla których kluczowa zwykle jest funkcjonalność a nie bezpieczeństwo.

Od kiedy Oracle włączył Apache z PHP do zestawu aplikacji oferowanym użytkownikom bazy uzyskali oni potężne narzędzie do łatwego pisania aplikacji webowych dających natychmiast możliwość dostępu do danych w bazie przez interfejs WWW. Dla PHP istnieją również setki zewnętrznych aplikacji, dostępnych w sieci za darmo – włącznie z systemami Content Management System, sklepami internetowymi, programami wspomagającymi dla biznesu (CRM, groupware) i innymi.

Skorzystanie z udostępnianej przez PHP platformy do uruchamiania aplikacji webowych w domyślnej konfiguracji wiąże się ze sporym ryzykiem, nawet jeśli dostępna wersja Apache i PHP jest pozbawiona błędów specyficznych dla tych dwóch elementów. Większym problemem są tutaj same aplikacje PHP.

### **Specyficzna architektura**

Z punktu widzenia bezpieczeństwa PHP ma jeden bardzo poważny problem, nierozwiązany do dziś – jest to brak separacji przywilejów. Wynika to ze sposobu działania serwera Apache, który pracuje z prawami nieuprzywilejowanego pseudoużytkownika - „www”, „apache”, „httpd” lub innego, w zależności od dystrybucji Linuksa. Dla osiągnięcia maksymalnej wydajności PHP jest zwykle uruchamiane jako dynamicznie ładowany moduł binarny (DSO) i działa z uprawnieniami Apache.

Jest to poważny problem zwłaszcza w systemach, w których swoje aplikacje PHP uruchamia wielu użytkowników – jeśli Apache działa jako „www” to aplikacja PHP każdego użytkownika również będzie uruchamiana jako „www”. Stwarza to szereg problemów związanych z bezpieczeństwem – jeśli aplikacja użytkownika „piotr” chce czytać plik „kartyKredytoweKlientow.txt” to powinien on mieć prawa dostępu, które dają prawa czytania go użytkownikowi „www” bo z takimi uprawnieniami będzie faktycznie działać aplikacja. Ale co jeśli użytkownik „paweł” zechce przeczytać ten sam plik? Niestety, w ramach jednego systemu będzie to możliwe ponieważ program w PHP tego użytkownika również działa jako „www”.

Rozwiązaniem tego problemu jest uruchamianie interpretera PHP jako programu CGI – wówczas można wykorzystać jeden z dostępnych w Apache mechanizmów (suexec, cgid) do zmiany uprawnień procesu tak by PHP zawsze działało z prawami użytkownika, który jest właścicielem skryptów – skrypty „piotra” będą uruchamiane z prawami „piotra”, a skrypty „pawła” z prawami „pawła”. Dzięki temu żaden z nich nie będzie musiał udostępniać swoich plików niezaufanemu użytkownikowi „www”.

Niestety – coś za coś. Uruchamianie PHP jako CGI jest obciążone większym narzutem ponieważ, w odróżnieniu od modułu DSO wymaga uruchomienia oddzielnego procesu dla każdego programu PHP. Istnieją rozwiązania rozwiązujące również i ten problem – na przykład moduł FastCGI uruchamia po jednej instancji interpretera dla każdego użytkownika. Działa ona w pamięci ładując i wykonując programy PHP praktycznie bez narzutu wydajnościowego. Inaczej działa suPHP, które działa jako moduł i wywołuje zewnętrzny program w celu zmiany uprawnień aktualnie działającego skryptu.

Wszystkie te rozwiązania mają jednak jedną podstawową wadę – nie są dostępne domyślnie w dystrybucji PHP i ich zastosowanie wymaga skompilowania PHP ze źródeł.

## Przyszłość PHP

Czy należy spodziewać się że w PHP będą się pojawiać kolejne błędy? Na pewno tak i nie należy mieć złudzeń że uda się załatać wszystkie problemy. O ile sam interpreter języka można doprowadzić do stanu w którym każda nowa funkcja jest analizowana pod względem bezpieczeństwa, o tyle poszczególnych modułów jest zbyt wiele by dało się przypilnować wszystkich autorów. Dlatego nadchodzące lata będą obfitowały głównie w dziury w rozszerzeniach PHP oraz aplikacjach pisanych w PHP.

Te ostatnie są nie do uniknięcia ze względu na to że PHP dawno przekroczyło masę krytyczną i obecnie każdy może pisać – i pisze darmowe aplikacje w tym języku. Funkcjonalnie mogą być one bardzo dobre ale bezpieczeństwo nadal pozostawia bardzo wiele do życzenia. Przykładem może być tutaj phpBB lub phpMyAdmin, które doświadczają od dłuższego czasu prawdziwego wysypu dziur związanych z bezpieczeństwem. Co gorsza, nie wiadomo kiedy się on skończy.

## Jak pozostać bezpiecznym?

Kluczem do bezpiecznego serwera WWW jest założenie że prędzej czy później ktoś znajdzie błędy w oprogramowaniu samego serwera (Apache), interpreterze języka (PHP), jego rozszerzeniach (np. moduł Oracle) lub samej aplikacji. Założenie to ma niemal stuprocentową szansę spełnienia się – ale administrator, który jest na to gotowy ma również niemal stuprocentową szansę uniknięcia problemów.

Charakterystyczną cechą open-source jest to że rozwiązanie każdego problemu, który pojawia się w związku z jego używaniem leży w możliwościach samego zainteresowanego – nie trzeba prosić „supportu” i czekać na poprawkę. Dzięki temu każdy z opisanych wyżej problemów doczekał się rozwiązań stworzonych przez niezależnych autorów, często były one włączane do głównej dystrybucji. Warto stosować wszystkie te środki lub przynajmniej część:

- Ⓞ **Hardening systemu operacyjnego.** Projekt Grsecurity [grsec] udostępnia łatę na kernel Linuksa, która znacznie zwiększa bezpieczeństwo systemu przez wprowadzenie ochrony pamięci przez atakami przez przepełnienie bufora oraz innymi klasycznymi atakami. Rozbudowany system MAC (*Mandatory Access Control*) umożliwia zamknięcie serwera Apache wraz z interpreterem PHP w „klatce”, z której nie będzie on w stanie zaszkodzić reszcie systemu nawet w razie włamania przez WWW. Warto również skompilować serwer Apache oraz pozostałe aplikacje sieciowe (PHP, Zope) za pomocą specjalnego kompilatora GCC z dodaną przez IBM funkcją ochrony pamięci SSP [ssp]. W obu przypadkach narzut wydajnościowy jest minimalny a poziom ochrony – duży. Co ważniejsze, są to mechanizmy uniwersalne, chroniące przed szerokim spektrum nowych ataków.
- Ⓞ **Seperacja przywilejów PHP.** W przypadku serwerów nie pracujących pod dużym obciążeniem i uruchamiających skrypty PHP wielu użytkowników można wykorzystać wbudowany w Apache

moduł CGI oraz mechanizm suExec. Jeśli wydajność jest ważnym kryterium to można ją podnieść przy zachowaniu separacji za pomocą modułu FastCGI lub zewnętrznego modułu suPHP [suphp].

Ⓞ **Ochrona aplikacji PHP przed atakami za pomocą mod\_security.** Zewnętrzny moduł mod\_security [modsec] jest pozycją, która powinna się znaleźć w każdym serwerze Apache. Pozwala on chronić uruchamiane aplikacje przed atakami z sieci wykorzystującymi typowe problemy w aplikacjach webowych. Umożliwia wykrywanie oraz blokowanie zapytań HTTP, które są typowe dla ataków wykorzystujących problemy z kodowaniem zmiennych, czytanie plików z systemi, CSS (*Cross-Site-Scripting*) oraz inne klasyczne błędy. Dużym plusem mod\_security jest fakt że umożliwia on nie tylko profilaktycznie zabezpieczenie serwera przed nowymi atakami, ale także szybkie założenie filtra na nową klasę ataków nawet bez poprawiania samej aplikacji. Znakomicie zmniejsza to okres, w którym nasz serwer jest podatny na ataki z zewnątrz.

[grsec] <http://www.grsecurity.net/>

[ssp] <http://www.research.ibm.com/trl/projects/security/ssp/>

[suphp] <http://www.suphp.org/>