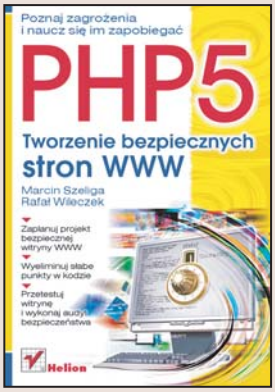
PHP5. Tworzenie bezpiecznych stron WWW

Autorzy: Marcin Szeliga, Rafał Wileczek
ISBN: 83-246-0255-0
Format: B5, stron: 296



Poznaj zagrożenia i naucz się im zapobiegać

- Zaplanuj projekt bezpiecznej witryny WWW
- Wyeliminuj słabe punkty w kodzie
- Przetestuj witrynę i wykonaj audyt bezpieczeństwa

Jedną z najpopularniejszych technologii stosowanych przez twórców witryn WWW jest język PHP, baza danych MySQL oraz serwer WWW Apache. Dynamiczny rozwój internetu spowodował pojawienie się na rynku ogromnej ilości aplikacji, za pomocą których niemal każdy może stworzyć dynamiczną witrynę WWW bez konieczności poznawania tej technologii. W efekcie tego w sieci pojawiły się tysiące witryn zbudowanych za pomocą wygodnych w użytkowaniu narzędzi graficznych. Taki stan rzeczy, poza wieloma zaletami, ma jedną ogromną wadę – twórcy takich witryn rzadko zwracają uwagę na ich bezpieczeństwo. Stworzenie bezpiecznej witryny WWW wymaga pewnej znajomości języka PHP i technik zabezpieczania kodu.

Książka „PHP5. Tworzenie bezpiecznych stron WWW" to podręcznik dla twórców witryn WWW znających język PHP. Przedstawia sposoby minimalizowania ryzyka związanego z atakami hakerskimi przeprowadzanymi dzięki „dziurom" w kodzie strony WWW. Opisuje kluczowe aspekty zabezpieczania witryn WWW i serwerów, na których są one publikowane, omawia modele zagrożeń i metody testowania kodu. Zawiera cenne informacje nie tylko dla programistów, ale również dla administratorów serwerów.

- Słabe punkty witryn WWW
- Klasyfikacja zagrożeń – model STRIDE
- Projektowanie bezpiecznej aplikacji WWW
- Sprawdzanie poprawności danych
- Metody uwierzytelniania i autoryzacji
- Bezpieczne połączenia z bazami danych
- Ochrona danych i kryptografia
- Testowanie aplikacji
- Konfigurowanie zapór sieciowych
- Przeprowadzanie audytów bezpieczeństwa witryn WWW

Jeśli chcesz tworzyć witryny WWW, które nie będą łatwym łupem dla hakerów – koniecznie przeczytaj tę książkę.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 4.

Uwierzytelnianie

Uwierzytelnianie jest procesem polegającym na sprawdzeniu, czy dane principium (osoba, komputer, urządzenie lub program) jest tym, za które się podaje. Uogólniając, uwierzytelnianie polega na sprawdzaniu wiarygodności informacji dotyczących obiektu lub osoby. W aplikacjach internetowych z reguły polega ona na zweryfikowaniu odebranych od użytkownika danych, np. zgodności wpisanych nazwy użytkownika i hasła z zapisanymi w bazie. **Osobę, której tożsamość została potwierdzona, nazywa się uwierzytelnionym użytkownikiem.**

Uwierzytelnianie jest podstawowym mechanizmem zabezpieczenia aplikacji WWW. Uzupełnione o opisaną w następnym rozdziale autoryzację pozwala zablokować dostęp do niektórych stron lub funkcjonalności nieuprawnionym osobom.

Aplikacje internetowe, jako ogólnie dostępne, są bardzo dogodnym celem ataku. Na serwerach WWW przechowywanych jest sporo mających wpływ na bezpieczeństwo danych — na przykład informacje konfiguracyjne czy nawet hasła. Dlatego musimy w jakiś sposób ograniczać do nich dostęp — niektóre pliki powinny być tylko odczytywane, inne — odczytywane i modyfikowane wyłącznie przez uprawnionych użytkowników, a jeszcze inne — zupełnie niedostępne.

Dostęp anonimowy

Czasami nie ma potrzeby sprawdzania tożsamości użytkowników — np. jeżeli chcemy, aby wszyscy mieli takie same uprawnienia, albo gdy nie interesuje nas, kto używa naszego programu. W takich przypadkach najlepszym rozwiązaniem jest zezwolenie na anonimowy dostęp.

Pierwsze wysłane do serwera WWW żądanie jest zawsze anonimowe (nie zawiera danych uwierzytelniających). Dopiero gdy serwer odrzuci takie żądanie, klient uzgadnia metodę uwierzytelniania i ponawia żądanie. Możliwe jest więc jednoczesne skonfigurowanie kilku różnych metod uwierzytelniania.

Metody uwierzytelniania

Uwierzytelnianie może zostać przeprowadzone na poziomie serwera WWW lub aplikacji internetowej. Ponieważ klienci bezpośrednio komunikują się z serwerem WWW, który przekazuje ich żądania do aplikacji WWW, w pierwszej kolejności przeprowadzone będą uwierzytelnienia na poziomie serwera, w drugiej — formularzy. ***Jeżeli planujesz sprawdzać tożsamości użytkowników na poziomie aplikacji — np. za pomocą formularza logowania — serwer WWW powinien zezwalać na anonimowy dostęp,*** w innym przypadku niektóre osoby nie zobaczą tego formularza, bo ich żądania zostaną zablokowane przez serwer WWW.

Uwierzytelnianie na poziomie serwera WWW

Właściwa konfiguracja systemu operacyjnego i serwera WWW jest jednym z najlepszych zabezpieczeń przed osobami, które próbują uzyskać dostęp do zastrzeżonych zasobów.

Standardowe zabezpieczenia obejmują uruchamianie serwera WWW w kontekście nieuprzywilejowanego konta, izolowanie procesu serwera i aplikacji WWW oraz zablokowanie użytkownikom internetowym dostępu do zasobów systemu operacyjnego.

Serwer Apache umożliwia:

1. Anonimowy dostęp — opcja umożliwiona domyślnie; można skonfigurować serwer tak, aby dostęp anonimowy był wyłączony dla poszczególnych udostępnianych stron (w sekcji <*Directory*> lub <*Location*> oraz poprzez plik *.htaccess*) lub dla wszystkich stron dostępnych na serwerze.
2. Standardowe uwierzytelniania — w tym przypadku dane uwierzytelniające przysłane są jawnym tekstem. Wybierając tę opcję, należy zabezpieczyć je za pomocą protokołu SSL/TLS. Zaletą jest zgodność ze standardem, główną wadą jest narzut czasowy związany z dostępem do pliku z hasłami, w sytuacji gdy w danej chwili wielu użytkowników wysyła żądanie dostępu do chronionych stron.
3. Uwierzytelnianie zaawansowane — m.in. w oparciu o LDAP, z wykorzystaniem MD5 i innych kryptograficznych funkcji mieszania. ***Zaawansowane uwierzytelnianie może nie być poprawnie obsługiwane przez różnego rodzaju oprogramowanie klienckie.***

Dane uwierzytelniające

Dane uwierzytelnionych użytkowników, w tym nazwa i hasło, muszą być dostępne z poziomu aplikacji WWW. ***PHP udostępnia dane uwierzytelniające w tablicy superglobalnej $_SERVER***: pod indeksem *PHP_AUTH_USER* znajduje się nazwa użytkownika, natomiast pod indeksem *PHP_AUTH_PW* — niezaszyfrowane hasło. Dane te mogą być dalej przetwarzane przez skrypt PHP.

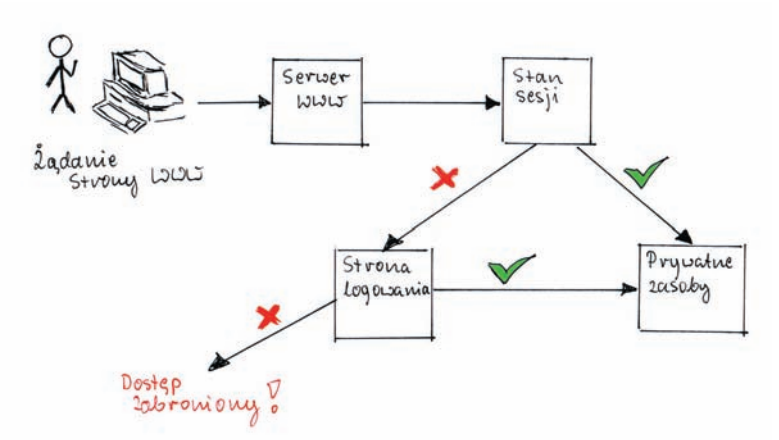
Uwierzytelnianie na poziomie formularzy

Typowy sposób ograniczenia dostępu do aplikacji WWW polega na utworzeniu strony logowania i przekierowaniu na tę stronę wszystkich żądań pochodzących od nieuwierzytelnionych użytkowników. Informacje uwierzytelniające są sprawdzane przez aplikację i z reguły przechowywane w zewnętrznym źródle danych — relacyjnej bazie danych czy dokumencie XML. Ten mechanizm uwierzytelniania jest często wykorzystywany w celu personalizacji (dostosowania wyglądu i funkcjonalności strony do prywatnych upodobań użytkownika) stron WWW.

Przebieg uwierzytelnienia

Proces uwierzytelnienia użytkownika przez aplikację WWW przebiega następująco:

1. Użytkownik wysyła żądanie wyświetlania chronionej strony WWW.
2. Ponieważ serwer WWW nie wymaga uwierzytelniania, żądanie zostaje przekazane do aplikacji WWW.
3. Następuje sprawdzenie danych sesji użytkownika. W przypadku braku danych uwierzytelniających:
   a) użytkownik zostaje przekierowany do strony logowania;
   b) aplikacja weryfikuje podane przez niego dane uwierzytelniające i jeżeli są poprawne, zapisuje je w danych sesji, a użytkownik zostaje z powrotem przekierowany do chronionej strony WWW;
   c) podanie niepoprawnych danych powoduje wyświetlenie odpowiedniego komunikatu i odmowę dostępu do chronionej strony (rysunek 4.1).



Rysunek 4.1. *Przebieg uwierzytelniania*

Wynika z tego, że uwierzytelnianie użytkowników wymaga przygotowania strony logowania i dodania do każdej chronionej strony kodu, który sprawdzi stan sesji i ewentualnie przekieruje użytkowników do strony logowania.

Strona logowania

Strona logowania powinna:

1. Umożliwić użytkownikowi podanie danych uwierzytelniających (np. nazwy i hasła).
2. Sprawdzić poprawność tych danych.
3. W przypadku ich poprawności zapisać dane uwierzytelniające w sesji użytkownika.
4. Przekierować żądania do odpowiednich stron WWW — uwierzytelnionych użytkowników do chronionych stron, nieuwierzytelnionych do strony z komunikatem błędu.

Komunikacja ze stroną logowania powinna być zabezpieczona opisanym w rozdziale 8. protokołem SSL/TLS.

Przekierowanie użytkownika

Każda chroniona strona musi zawierać kod sprawdzający stan sesji użytkownika i w razie potrzeby przekierowujący go do strony logowania.

Bardzo rzadko wszystkie strony muszą być chronione. Planując aplikację, zastanów się, które strony będą wymagać uwierzytelnienia.

Hasła

Hasła są — i wbrew pojawiającym się ostatnio opiniom przez najbliższe lata nadal będą — podstawowym sposobem uwierzytelniania użytkowników[1]. W większości przypadków hasło jest jedynym sposobem sprawdzenia, czy dana osoba jest tą, za którą się podaje. Czyli jeżeli ktokolwiek pozna hasło, to będzie mógł skutecznie podszyć się pod uwierzytelnionego użytkownika.

[1] Ostatnio jednym z ulubionych przez niby-speców od bezpieczeństwa sloganów jest: „Przy mocy obliczeniowej współczesnych komputerów zapamiętanie bezpiecznego hasła jest niemożliwe, a więc musimy zastąpić je np. certyfikatami". Błąd polega na tym, że to nie moc obliczeniowa ma decydujący wpływ na bezpieczeństwo, a zabezpieczenia aplikacji plus wiele dodatkowych czynników, choćby przepustowość sieci. Jeżeli nasze hasło liczy 6 znaków i zawiera tylko duże i małe litery, a atakujący jest w stanie przesyłać 200 haseł na minutę, odgadnięcie hasła zajmie mu — nawet na Deep Blue — około 300… lat.

Wymaganie bezpiecznych haseł

Bezpieczne są takie hasła, które znasz tylko Ty, a które nie mogą zostać zdobyte przez nieupoważnione osoby. W większości przypadków do zdobycia hasła atakujący wykorzystują specjalne programy, a więc to te programy, a nie człowieka, musimy przechytrzyć. Atakujący do zdobycia hasła może wykorzystać:

1. Swoją wiedzę o użytkowniku, np. znając datę urodzenia, przezwisko czy drugie imię, sprawdzi, czy któreś z nich nie jest jego hasłem.
2. Plik słownika — kolejne sprawdzanie wszystkich zapisanych w nim słów ujawni hasło będące dowolnym wyrazem języka polskiego albo jakiegokolwiek innego języka.
3. Program, który będzie sprawdzał wszystkie możliwe kombinacje liter, cyfr i znaków specjalnych. Atak tego typu wymaga sprawdzenia ogromnej liczby kombinacji (np. *Aa1*, *A1a*, *aA1*, *a1A*, *1Aa*, *1aA*) i w praktyce przeprowadzany jest po zdobyciu zaszyfrowanych danych uwierzytelniających.

Znając sposoby zdobywania haseł, możesz określić wymogi, jakie powinny spełniać hasła, aby ich zdobycie było praktycznie niemożliwe:

1. Po pierwsze, hasło nie może w ogóle przypominać: nazwy użytkownika, imienia, nazwiska, daty urodzenia, adresu, imienia ulubionego zwierzęcia, autora, filmu czy książki.
2. Po drugie, hasło nie może być wyrazem jakiegokolwiek języka — hasła *inwentaryzacja*, *agrokultura* czy *uporządkowywanie* są prawie tak samo łatwe do zdobycia, jak hasła typu *1234*.
3. Po trzecie, hasło nie powinno być krótsze niż ośmioznakowe, a jeżeli nie ma żadnych specjalnych przeciwwskazań, długość hasła powinna wynosić co najmniej 12 znaków.
4. Po czwarte, hasło powinno zawierać przynajmniej kilka
   a) wielkich liter alfabetu (od A do Z),
   b) małych liter alfabetu angielskiego (od a do z),
   c) cyfr (od 0 do 9),
   d) znaków specjalnych (np.:, !, $, #, %).

PHP wyposażony został w mechanizmy pozwalające na wymuszenie na użytkowniku „silnego” hasła. Listing 4.1 przedstawia zastosowanie funkcji wchodzących w skład rozszerzenia *Crack* do testowania poprawności hasła podanego przez użytkownika (listing 4.1).

Listing 4.1. *Sprawdzanie bezpieczeństwa hasła podanego przez użytkownika*

```
<?php
  // ...
  // Otwarcie pliku zawierającego słownik
```

```
    $slownik = crack_opendict($sciezka.'slownik.pwd');
    if ($slownik) {
            // Sprawdzenie poprawności hasła
            if (crack_check($slownik, $_POST['passwd'])) {
                    echo "Hasło poprawne!";
                    // ...
            } else {
                    echo "Hasło nie spełnia zasad bezpieczeństwa - spróbuj jeszcze
raz.\n";
                    // ...
            }
            crack_closedict($slownik);
    } else {
            echo "Problem z otwarciem pliku słownika\n";
    }
?>
```

Jak nauczyć użytkowników stosowania bezpiecznych haseł?

Podstawowym błędem, który większość użytkowników popełnia przy wymyślaniu haseł, jest próba znalezienia jakiegoś skomplikowanego wyrazu i dodanie na początku lub na końcu kilku cyfr. Efektem takiego podejścia są hasła typu: *Eklezjologia4*, *12indosatariuszy* czy *merkantylizacja111*. W rezultacie otrzymujemy trudne do zapamiętania hasło, którego złamanie jest równie łatwe, co złamanie hasła typu *Marcin77* albo *2razy4* — przecież atakujący dysponują programami, które z równą łatwością sprawdzają hasła obu rodzajów.

Inna popularna metoda tworzenia haseł polega na zastępowaniu określonych liter w wyrazie znakami alfanumerycznymi i specjalnymi. Na przykład hasło *hasło* może zostać zastąpione ciągiem *h@sł0*. **Takie sztuczki nie chronią już przed atakami słownikowymi** — od kiedy coraz więcej osób zaczęło stosować takie hasła, atakujący dodali opisywane permutacje do słowników i prawdopodobieństwo zdobycia obu haseł stało się prawie identyczne (np. zobacz słownik dostępny na stronie *http://www.openwall.com*).

Zamiast próbować utworzyć hasło na podstawie jednego wyrazu, wykorzystaj w tym celu całe zdania bądź frazy. Na przykład zdanie *Tylko nie dzisiaj* może zostać zamienione na *Tylk0_Nie_Dzisiaj* — bezpieczne, bo nie znajdujące się w żadnym słowniku, skomplikowane i długie, a jednocześnie łatwe do zapamiętania hasło. Albo zdanie *Lubię swoją pracę* można zmienić na hasło *LubieSw0jąPracę* — przecież żaden program do łamania haseł nie będzie sprawdzał połączenia wszystkich występujących w słowniku słów.

Innym sposobem na wymyślenie bezpiecznego hasła jest użycie początkowych liter fragmentu dobrze Ci znanego wiersza czy piosenki — np.:

Czy pamiętasz, jak z tobą tańczyłem walca,
Panno, madonno, legendo tych lat?

możesz zamienić na hasła: *CzyJakW@lca*, *Pa-ma-le-ty-La*, *Cp,jzttw* itd.

Tworząc hasła, powinieneś pamiętać, że:

1. Im dłuższe hasło, tym lepsze — nie należy bać się tworzenia długich haseł. Z doświadczenia wiemy, że im bardziej doświadczona osoba, tym dłuższe są stosowane przez nią hasła. Na przykład administrator jednego z większych polskich systemów komputerowych stosował hasło

   J@kJaNienawidzeTejCh0lernejRoboty!!!.

2. Prawie zawsze do zdobycia hasła są wykorzystywane specjalne programy, a więc to je musisz przechytrzyć, a nie atakującego — zamiast bezmyślnie używać znalezionych w słowniku skomplikowanych wyrazów, powinieneś wykorzystać swoją fantazję do przekształcenia dobrze Ci znanych zdań czy fraz.

3. **Jeżeli użytkownik z obawy, że je zapomni, stosuje słabe hasła, przekonaj go, żeby wymyślił i zapisał sobie na kartce bezpieczne hasło.** W zapisywaniu haseł nie ma nic złego, o ile kartka z hasłem nie znajdzie się w pobliżu komputera. Najlepszym miejscem dla zapisanego hasła jest portfel.

Sygnatury

Poniższe wskazówki powinny być dla Ciebie oczywiste:

1. **Nigdy nie przechowuj haseł użytkowników w jawnej postaci.** W końcu tylko oni (a nie np. administrator serwera i programista aplikacji WWW) powinni je znać. Zapisanie ich gdziekolwiek, choćby w pliku tymczasowym czy pamięci, w jawnej postaci obniża poziom bezpieczeństwa aplikacji do zera.

2. **Z tego samego powodu nigdy nie przechowuj haseł zaszyfrowanych odwracalnie,** tj. w taki sposób, że na podstawie szyfrogramu można otrzymać jawną postać hasła.

3. **Jedynym w miarę bezpiecznym sposobem przechowywania haseł jest przechowywanie ich sygnatur** (wyników funkcji mieszania). W ten sposób nie ograniczamy funkcjonalności — aby sprawdzić poprawność hasła, wystarczy porównać wyliczoną na podstawie wpisanego hasła sygnaturę z zapisaną np. w bazie danych. Jeżeli sygnatury są zgodne, to prawie na pewno użytkownik podał poprawne hasło.

Jeżeli jesteś ciekawy, z czego wynika ta pewność, zajrzyj do rozdziału 8.

To rozwiązanie nie jest bezpieczne, jest jedynie bezpieczniejsze niż poprzednie. Nie tylko dlatego, że wiedząc, w jaki sposób zostały wyliczone sygnatury, atakujący zawsze może porównywać wyliczone przez siebie i zapisane sygnatury[2], ale również dlatego, że **w pewnych sytuacjach, znając sygnaturę, atakujący może podszyć się pod użytkownika** — w końcu weryfikujemy zgodność sygnatur, prawda?

[2] Ponieważ używane algorytmy są znane, narzędzia te po prostu porównują sygnatury wszystkich kombinacji znaków albo słów ze słownika oraz ich różnych permutacji z sygnaturą hasła.

Automatyczne generowanie haseł

Zdarza Ci się zapomnieć hasła do niektórych stron WWW? Innym też, więc projektując aplikację WWW, powinieneś umożliwić użytkownikom automatyczne generowanie nowych haseł[3]. W innym przypadku administrator będzie miał sporo dodatkowej pracy.

W większości przypadków mechanizm generowania haseł wykorzystuje dwie dodatkowe związane z kontem użytkownika informacje:

1. ogólnie znane i wyświetlane anonimowym użytkownikom pytanie,
2. poufną, znaną tylko danemu użytkownikowi odpowiedź.

Jeżeli wpisany przez anonimowego użytkownika adres e-mail oraz odpowiedź są takie same jak przechowywane w bazie, aplikacja automatycznie zmieni hasło i wyśle je pod wskazany adres (listing 4.2).

Listing 4.2. *Generowanie nowego hasła*

```
<?php
  $nowe_haslo = "";
  $dlugosc = 8; // długość hasła

  for ($i = 0; $i < $dlugosc; $i++) {
        // nowe hasło ma się składać wyłącznie ze znaków ASCII o kodach z zakresu od 33 do 126
        $nowe_haslo .= chr(mt_rand(33, 126));
  }

  // Zmiana hasła w bazie danych (oczywiście hasło zostaje najpierw zakodowane)
  $zakodowane_haslo = sha1($nowe_haslo);
  // ...aktualizacja bazy danych...

  // Wysłanie niezakodowanego hasła użytkownikowi
  mail($_POST['email'], "Nowe hasło", "Twoje nowe hasło to: ".$nowe_haslo);
?>
```

[3] Oczywiście powinieneś również umożliwić im zmianę własnego hasła.