Ostatni odcinek poświęcony jest wykorzystaniu umiejętności z poziomu języka PHP, zdobytych w ramach kursu. Ponieważ jest to kurs MySQL-a, a nie języka PHP, zawiera on jedynie krótki opis interfejsu API serwera i uwagi na temat bardzo popularnego ataku na bazy danych — iniekcji kodu SQL.

## Dwa interfejsy PHP serwera MySQL

MySQL zawiera dwa moduły do współpracy z PHP:

1. Moduł mysql używany w wersjach wcześniejszych niż 4.1 do współpracy z PHP 4 i 5. Jeżeli tylko masz taką możliwość, nie korzystaj z tego przestarzałego modułu.
2. Moduł mysqli (i oznacza „ulepszony”, ang. Improved) przeznaczony dla wersji 4.1.3 i późniejszych serwera MySQL, umożliwiający współpracę jedynie z wersją PHP 5. To rozszerzenie zostało poniżej krótko przedstawione. Dzięki niemu automatycznie poprawisz wydajność i bezpieczeństwo programu (czyli nic nie zmieniając w kodzie, sprawisz, że program będzie działał szybciej i bezpieczniej).

### Włączanie modułu mysqli

Żeby korzystać z opisywanego modułu, należy go włączyć do kompilowanej wersji interpretatora PHP:

./configure --with-mysqli=/usr/bin/mysql_config --with-mysql=/usr

A następnie skompilować i zainstalować PHP:

make

make install

## Mysqli

W ramach tego odcinka przedstawię kilka przykładowych skryptów pokazujących typowe techniki programowego odczytywania i przetwarzania danych. Dokładny opis modułu mysqli, włącznie z przykładami użycia większości metod wszystkich klas tego modułu, znajduje się pod adresem http://pl.php.net/mysqli.

## Przykład połączenia z bazą danych — podejście proceduralne

Skrypt z listingu 13.1 pokazuje, jak nawiązać połączenie z bazą danych. Poszczególne listingi nie są samodzielnymi, działającymi programami — lecz trzema częściami jednego skryptu, podzielonego na części ze względów edukacyjnych. Żeby go wykonać, należy połączyć je w jeden skrypt.

Listing 13.1. Łączymy się z testową bazą. Ze względów bezpieczeństwa połączenie nie jest nawiązywane w kontekście użytkownika root

```
<?php


/* Łączymy się z serwerem */

$link = mysqli_connect(

        'localhost',  /* nazwa komputera, na którym działa MySQL */

        'user',       /* nazwa użytkownika */
```

```
        'password',   /* hasło */

        'test');     /* domyślna baza danych */



if (!$link) {

  printf(&quot;Brak połączenia z serwerem MySQL. Kod błędu: %sn&quot;, mysqli_connect_error());

  exit;

}

?>
```

Zmienna $link będzie naszym uchwytem do nawiązanego połączenia. Jeżeli połączenie zostanie ustanowione, jej wartością będzie prawda, w przeciwnym razie wartością zmiennej będzie fałsz. Za pomocą warunku IF sprawdzamy, czy udało się połączyć z bazą danych, i jeżeli nie, zwracamy kod błędu i przerywamy działanie skryptu.

Po nawiązaniu połączenia możemy wykonywać dowolne instrukcje języka SQL (listing 13.2), a jego wynik wykorzystać w skrypcie.

Listing 13.2. Odczytujemy z bazy i wyświetlamy nazwy oraz ceny towarów

```
<?php



/* Wykonujemy zapytanie */

if ($result = mysqli_query($link, 'SELECT  description, sell_price FROM item ORDER BY sell_price DESC')) {



   print(&quot;Lista  towarów:n&quot;);



   /* Przetwarzanie  wierszy wyniku zapytania */

   while( $row = mysqli_fetch_assoc($result)  ){

      printf(&quot;%s (%s)n&quot;,  $row['description'], $row['sell_price']);

   }

   /* Usuwamy  z pamięci wynik zapytania */

   mysqli_free_result($result);

}

?>
```

Zwróć uwagę na kilka kwestii:
1. Zmienna $result zawiera wynik zapytania.
2. Do wykonania zapytania (wywołania procedury mysqli_query) wymagane jest otwarte połączenie z bazą.

3. Przekazana jako parametr wywołania instrukcja języka SQL jest umieszczona w apostrofach i nie jest zakończona średnikiem.
4. Ze względu na prostotę i wydajność sortowaniem danych zajął się MySQL.
5. Zapytanie zwraca tylko potrzebne dane.
6. Wynik zapytania jest zapisywany w postaci tablicy (procedura mysqli_fetch_assoc), której kolejne wiersze będą wyświetlone na ekranie.
7. Po przetworzeniu (np. wypełnieniu tabeli) wynik zapytania jest usuwany, a pamięć zwalniana.

Pozostało nam tylko zamknąć aktywne połączenie (listing 13.3).

Listing 13.3. Otwieranie połączeń jest dość wolne, dlatego warto raz nawiązać je na początku skryptu i zamknąć na jego końcu

```
<?php



/* Zamykamy połączenie z bazą */

mysqli_close($link);
```

## ?> Przykład połączenia z bazą danych — podejście obiektowe

PHP 5 umożliwia tworzenie programów zorientowanych obiektowo, a rozszerzenie mysqli nie jest tu wyjątkiem. Program z listingu 13.4 jest funkcjonalnym odpowiednikiem poprzednich skryptów, ale tym razem nawiązanie połączenia, wykonanie zapytania, przetworzenie wyników i zamknięcie sesji zrealizowane zostało obiektowo.

Listing 13.4. Odczytujemy listę towarów i ich cen

```
<?php



/* Łączymy się z serwerem */

$mysqli = new mysqli('localhost', 'user', 'password', 'test');



if (mysqli_connect_errno()) {

   printf(&amp;quot;Brak połączenia z serwerem MySQL. Kod błędu: %sn&amp;quot;, mysqli_connect_error());

   exit;

}
```

```
/* Wykonujemy zapytanie */

if ($result = $mysqli->query(SELECT  description, sell_price FROM item ORDER BY sell_price ')) {



    print(&amp;quot;Lista  towarów:n&amp;quot;);



    /* Przetwarzanie  wierszy wyniku zapytania */

    while( $row = $result->fetch_assoc()  ){

        printf(&amp;quot;%s (%s)n&amp;quot;,  $row['description'], $row['sell_price']);

    }



    /* Usuwamy  wynik zapytania z pamięci */

    $result->close();

}



/* Zamykamy  połączenie z bazą */

$mysqli->close();

?>
```

Zwróć uwagę, że:
1. W tym wypadku nie musimy przechowywać uchwytu do połączenia w osobnej zmiennej.
2. Wywołując metodę fetch_assoc(), nie musimy jawnie wskazywać na zmienną zawierającą wynik zapytania.

## Przykład przekazywania parametrów w instrukcjach SQL

W poprzednich przykładach MySQL wykonywał zapisane na trwale w skryptach instrukcje języka SQL. W większości wypadków takie rozwiązanie będzie niefunkcjonalne — nie będziemy przecież w stanie zapisać wszystkich możliwych wersji zapytań. Znacznie lepszym pomysłem byłoby użycie w kodzie SQL zmiennych, których wartości byłyby dynamicznie ustawiane z poziomu PHP. Czyli chodzi o to, żeby instrukcje typu

```
SELECT description, sell_price

FROM v_drogie

WHERE description =  'Speakers'
```

zastąpić instrukcjami typu

```
SELECT description, sell_price
```

FROM v_drogie

WHERE description =  ?

a w miejsce znaku ? dynamicznie podstawić  odpowiednią wartość przed wykonaniem przez MySQL (listing 13.5).

Listing 13.5. Przykład dynamicznego przekazywania parametrów

```
<?php

$mysqli = new mysqli('localhost', 'user',  'password', 'test');


/* Test połączenia */

if (mysqli_connect_errno()) {

    printf(&quot;Brak połączenia z  serwerem MySQL. Kod błędu: %sn&quot;,  mysqli_connect_error());

    exit();

}


/* Przygotowanie instrukcji SQL*/

$stmt = $mysqli->prepare(&quot;INSERT INTO  item (description, sell_price, cost_price) VALUES ( ?, ?,
?)&quot;);

$stmt->bind_param('sdd', $description,  $sell_price, $cost_price);


$description = 'SQL Server 2005 Standard';

$sell_price = 1500;

$cost_price = 1323.12;


/* Wykonujemy  przygotowaną instrukcję */

$stmt->execute();


printf(&quot;%d Wstawiono wierszy: n&quot;,  $stmt->affected_rows);


/* Usuwamy  wynik zapytania z pamięci */
```

```
$stmt->close();



/* Zamykamy  połączenie z bazą */

$mysqli->close();

?>
```

Omówienie:
1. Pierwszy parametr metody bind_param() określa typ wiązanych parametrów. W tym przypadku 'sdd' oznacza, że pierwszy parametr jest ciągiem znaków, a dwa następnie — liczbami dziesiętnymi.
2. Przed wykonaniem instrukcji SQL trzeba określić wartości wszystkich parametrów.
3. Odczytując atrybut affected_rows, dowiemy się, ile wierszy zostało zmodyfikowanych w ramach ostatnio wykonywanej instrukcji.

## Przykład wiązania wyników

Zwracane przez zapytania wyniki mogą być wiązane poprzez zmienne PHP z polami tekstowymi, polami wyboru czy polami w tabelach. Listing 13.6 pokazuje przykład takiego powiązania.

Listing 13.6. Wiążąc pola z wynikami zapytań, w prosty i elegancki sposób umożliwimy użytkownikom odczytywanie i modyfikowanie powiązanych danych

```
<?php

$mysqli = new mysqli(&quot;localhost&quot;,  &quot;user&quot;, &quot;password&quot;, &quot;test&quot;);



if (mysqli_connect_errno()) {

   printf(&quot;Brak połączenia z serwerem  MySQL. Kod błędu %sn&quot;, mysqli_connect_error());

   exit();

}



/* Przygotowanie  i wykonanie instrukcji SQL*/

if ($stmt = $mysqli->prepare(&quot;SELECT  description, sell_price FROM item ORDER BY sell_price DESC LIMIT 5&quot;)) {

   $stmt->execute();



   /* Powiązanie  zmiennych z wynikiem zapytania */

   $stmt->bind_result($col1, $col2);
```

```
   /* Przetwarzanie wyniku */

   while ($stmt->fetch()) {

      printf(&quot;%s %sn&quot;, $col1,  $col2);

   }


/* Usuwamy  wynik zapytania z pamięci */

$stmt->close();




/* Zamykamy  połączenie z bazą */

$mysqli->close();
```

?> **Iniekcja SQL**

Iniekcja SQL ma miejsce wtedy, gdy niesprawdzone dane wejściowe są przekazywane do serwera baz danych. Ponieważ SQL jest językiem interpretowanym, dowolny ciąg znaków będzie zinterpretowany przez serwer baz danych. Jeżeli ten ciąg będzie poprawną instrukcją języka SQL, to zostanie wykonany, w przeciwnym razie serwer zgłosi błąd. **Atak**

Iniekcja SQL pozwala atakującemu na:

1. Poznanie struktury bazy danych — jeżeli błędy zgłaszane przez serwer bazy danych będą odesłane do przeglądarki, atakujący zdobędzie informacje o nazwach tabel i kolumn, typie poszczególnych kolumn, istniejących procedurach składowanych i funkcjach itd.
2. Poszerzenie posiadanych uprawnień — wpisana przez atakującego instrukcja SQL będzie wykonana w kontekście zabezpieczeń innego, często uprzywilejowanego konta użytkownika.
3. Wykonywanie na bazie danych dowolnych instrukcji SQL — jeżeli aplikacja WWW nawiązuje połączenie z bazą w kontekście konta administratora, to każda instrukcja SQL wpisana przez użytkownika i bez sprawdzania wysłana do bazy zostanie wykonana.
4. Wywoływanie procedur składowanych — serwery baz danych zawierają procedury składowane, niektóre z nich rozszerzają funkcjonalność serwera (np. wywołują dowolny skrypt powłoki). Atakujący, wywołując te procedury, uzyska uprzywilejowany dostęp do zasobów systemowych.

**Sesje klienckie**

Zanim działające po stronie serwera WWW programy będą mogły wykonać jakąkolwiek instrukcję języka SQL, muszą nawiązać sesję z wybraną bazą danych. Wymaga to od nich potwierdzenia własnej tożsamości. W najprostszym przypadku skrypt PHP zawiera pokazany na listingu 13.7 fragment kodu.

Listing 13.7. Przykład, jak nie należy łączyć się z serwerem SQL

```
<?php

$mysqli = new mysqli('localhost', 'root', 'password', 'test');

$result = $mysqli->query(SELECT * FROM Pracownicy WHERE login = '&amp;quot; + $Login + &amp;quot;' and haslo =
'&amp;quot; + $Haslo +&amp;quot;'&amp;quot;);

$result->close();

$mysqli->close();

?>
```

Kluczowe dla bezpieczeństwa są trzy elementy:
1. Po pierwsze, sesja została nawiązana w kontekście zabezpieczeń uprzywilejowanego użytkownika root, a więc dowolne przesłane w ramach tej sesji instrukcje języka SQL będą wykonane.
2. Po drugie, wykonywana instrukcja języka SQL zawiera apostrofy (w języku SQL ciągi znaków umieszczane są w apostrofach).
3. Po trzecie, instrukcja SQL jest dynamicznie konstruowana przez łączenie wpisanych przez użytkowników ciągów znaków.

**Iniekcja kodu**

Zamiast wprowadzić swoją nazwę (zmienna Login), użytkownik przykładowej aplikacji internetowej może zaszyć we wpisywanym ciągu znaków instrukcję języka SQL. Jeżeli rezultatem będzie poprawna instrukcja, to zostanie ona wykonana z uprawnieniami administratora serwera baz danych.

Punktem wyjścia naszej analizy będzie instrukcja SELECT wykonana w przypadku niepodania ani nazwy użytkownika, ani hasła:

```
SELECT * FROM Pracownicy WHERE login = '' and haslo = ''
```

Gdyby zamiast nazwy użytkownika w zmiennej Login wpisano ciąg znaków ';drop table dane--, serwer baz danych wykonałby następujące instrukcje:

```
SELECT * FROM Pracownicy WHERE login = ';drop table Dane--' and txtHaslo = ''
```

Dzięki apostrofowi wpisane wyrażenie będzie zinterpretowane, a nie potraktowane jako ciąg znaków, a więc instrukcja DROP TABLE Dane (w języku SQL znak końca wiersza jest ignorowany) będzie pomyślnie wykonana. Bez tego apostrofu do serwera baz danych zostałyby wysłane następujące instrukcje:

```
SELECT * FROM Pracownicy WHERE login = 'drop table Dane--' and txtHaslo = ''
```

a więc wroga instrukcja byłaby użyta do przeszukania tabeli Pracownicy.

Ponieważ wszystkie znaki znajdujące się po prawej stronie znaku komentarza (w języku SQL znakiem komentarza są dwa myślniki) są ignorowane, serwer baz danych wykona poprawną instrukcję DROP TABLE dane, a nie błędną DROP TABLE Dane ' and txtHaslo = ''.

Znajomość konkretnej wersji języka SQL i umiejętne posługiwanie się apostrofem i myślnikami wystarczą do przeprowadzenia skutecznego ataku na niezabezpieczoną aplikację WWW. Aby na przykład odczytać całą zawartość tabeli bazowej, wystarczy zamiast nazwy użytkownika wpisać ciąg znaków 'or 1=1--. W efekcie zostanie wykonana poniższa instrukcja:

SELECT * FROM Pracownicy WHERE login = ''or 1=1--' and txtHaslo = ''

Ponieważ wartość jednego z argumentów operatora OR (alternatywy logicznej) będzie prawdą (1=1), wartość drugiego (login = '') nie będzie miała wpływu na wynik całego testu. **Obrona**

Obrona przed iniekcją wrogiego kodu polega na sprawdzaniu poprawności danych wprowadzanych przez użytkowników i filtrowaniu wysyłanych do nich informacji. Oprócz tego standardowym środkiem zaradczym jest łączenie się z bazą danych w kontekście konta o jak najmniejszych uprawnieniach — jeżeli użytkownik na przykład miałby jedynie prawo do odczytu danych z określonych widoków i do wywołania kilku procedur, to nawet udana iniekcja SQL nie naruszyłaby bezpieczeństwa aplikacji. **Formatowanie danych**

Jeżeli jest to możliwe, ze względów bezpieczeństwa należy zrezygnować z używania w ciągach znaków znaczników języka SQL, przede wszystkim apostrofów i myślników. Niestety, to proste zabezpieczenie może zostać w równie prosty sposób ominięte — jeżeli atakujący zorientuje, się w jaki sposób modyfikowane są wprowadzane przez niego dane, zamiast używać apostrofów, posłuży się funkcją CHAR(). Umożliwi mu to nie tylko dodawanie pojedynczych apostrofów (zamiast apostrofu wystarczy wywołać funkcję CHAR(39)), ale np. wstawienie dowolnych ciągów znaków.

Co więcej, formatując dane, możemy niekorzystnie wpłynąć na działanie mechanizmów blokujących niepoprawne dane — np. jeżeli po sprawdzeniu, czy wyrażenie nie zawiera zastrzeżonych instrukcji języka SQL, usuniemy z niego apostrofy, atakujący może przekazać następujący ciąg znaków: DRO'P T'ABLE Dane — i w ten sposób wyzyskać jeden mechanizm zabezpieczeń do pokonania innego zabezpieczenia. **Filtrowanie danych**

Zamiast usuwać z wprowadzonych danych potencjalnie niebezpieczne znaczniki języka SQL i przesyłać do serwera baz danych tak zmodyfikowane ciągi znaków, należy zablokować przesyłanie niepoprawnych danych. Takie rozwiązanie powinno być wykorzystane we wszystkich aplikacjach internetowych. Jeżeli wymagany będzie zwiększony poziom bezpieczeństwa, należy dodatkowo zezwolić na przetwarzanie wyłącznie poprawnych danych.

Poniższa funkcja (listing 13.8) zwraca prawdę, jeżeli w podanym ciągu znaków nie występuje apostrof lub myślnik. W przeciwnym razie funkcja zwraca fałsz.

Listing 13.8. Funkcja sprawdzająca poprawność danych i jej wywołanie w skrypcie PHP

```
<?php

  function testPoprawnosci($tekst) {

    if (preg_match(&amp;quot;/[-']/&amp;quot;, $tekst))

     return false; // znaleziono wystąpienie myślnika lub apostrofu

    else

     return true;  // badany ciąg nie zawiera niepożądanych znaków

  }

?>
```

Dodatkowym zabezpieczeniem powinno być ograniczenie maksymalnej długości wprowadzanego ciągu znaków. Chociaż ta technika nie uniemożliwi przeprowadzenia skutecznego ataku, to może go znacznie skomplikować

(listing 13.9).

Listing 13.9. Ograniczając rozmiar danych, utrudnimy przeprowadzenie ataku

```
<?php

  if (strlen($login) <= 15) {

    // kod budujący instrukcje języka SQL

  } else {

    // przekierowanie do strony z komunikatem błędu

  }

?>
```

Najbezpieczniejsze rozwiązanie wymaga zdefiniowania wzorca poprawnych danych albo przygotowania listy wszystkich możliwych poprawnych wartości. Z oczywistych względów lista ta nie powinna być przechowywana w tabeli po stronie serwera baz danych, tylko np. w dokumencie XML zapisanym na serwerze WWW.

**Przechwytywanie komunikatów błędów**

Zgłaszane przez serwery baz danych wewnętrzne komunikaty błędów są z reguły bardzo dokładne — zdecydowanie zbyt dokładne, żeby pokazywać je końcowym użytkownikom. Przeciętny użytkownik i tak nie będzie wiedział, czego one dotyczą, za to atakujący na ich podstawie może poznać strukturę bazy i przechowywane w bazie dane.

Wszystkie wyjątki muszą zostać przechwycone przez aplikację WWW, informacje o błędzie (w tym oryginalny komunikat błędu) powinny być zapisane w pliku dziennika, a użytkownicy przekierowani na stronę zawierającą ogólną informację o tym, że wystąpił błąd. **Funkcje i procedury składowane**

Sprawdzanie, czy wprowadzone przez użytkowników dane nie zawierają instrukcji języka SQL, jest czasochłonne i — chociaż język SQL składa się zaledwie z dziewięciu podstawowych instrukcji — nie gwarantuje stuprocentowego bezpieczeństwa. Prostszym i skuteczniejszym rozwiązaniem jest zrezygnowanie z dynamicznego budowania instrukcji języka SQL z wykorzystaniem danych wprowadzonych przez użytkowników. Zamiast tego wprowadzane przez użytkowników dane powinny być przekazywane do serwera bazodanowego jako parametry wywołania procedur.

Na przykład zamiast sprawdzać, czy użytkownik, wykonując poniższą instrukcję, podał poprawny login i hasło:

$query = "SELECT COUNT(*) FROM pracownicy WHERE login = '" + $Login + "' and haslo = '" + $Haslo +"'"

należy utworzyć funkcję wywoływaną z dwoma parametrami (loginem i hasłem), zwracającą liczbę wierszy spełniających podany warunek (listing 13.10).

Listing 13.10. Skrypty PHP powinny jedynie wywoływać funkcje lub procedury. W ten sposób wielokrotnie zmniejszymy ryzyko udanego ataku

```
CREATE FUNCTION logowanie

(p1 varchar(50), p2 varchar(50))
```

RETURNS tinyint

RETURN (SELECT COUNT(*) FROM pracownicy WHERE  login = p1 AND haslo = p2);

Od teraz zapisany w skrypcie PHP kod SQL jedynie wywołuje  funkcje i nie jest dynamicznie konstruowany (i nie ma możliwości zaszycia w nim  wrogiej instrukcji SQL):

$query = "SELECT logowanie ($Login,  $Haslo)";    **Ograniczenie uprawnień**

Niekorzystanie z rozbudowanych modeli zabezpieczeń baz  danych jest niestosowaniem się do podstawowej strategii minimalizacji ryzyka — wyzyskania  istniejącej technologii do zabezpieczenia aplikacji. Tematem książki nie jest administrowanie  bazami danych, ani ich programowanie, dlatego ograniczę się do odesłania Czytelnika do poświęconych tym zagadnieniom pozycji — czas przeznaczony na poznanie  używanego serwera baz danych z pewnością zwróci się z nawiązką.