Marcin Lis

Tworzenie bezpiecznych aplikacji internetowych

(z przykładami w PHP)

Stwórz bezpieczny sejf na dane Twoich klientów i chroń swoją aplikację!

Jak projektować serwis internetowy, by zapewnić mu bezpieczeństwo?
Jakie ataki najczęściej zagrażają danym użytkowników i aplikacjom internetowym?
Jak poprawić bezpieczeństwo działającego już serwisu internetowego?

Helion

Redaktor prowadzący: Ewelina Burska
Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/twbeap
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody wykorzystane w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/twbeap.zip

ISBN: 978-83-246-8131-0

Printed in Poland.

Spis treści

Rozdział 1.

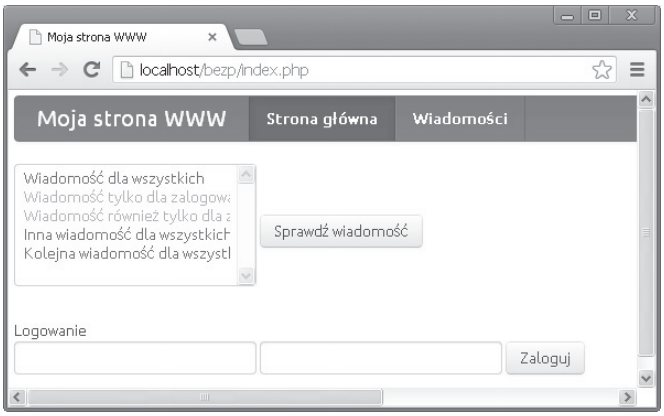
Kontrola dostępu do danych i funkcji

Modyfikacje elementów interfejsu

Pierwszy rozdział, na rozgrzewkę, jest poświęcony — wydawałoby się — prostym błędom związanym z brakiem kontroli dostępu do danych i funkcji. Typowym przykładem jest poleganie na mechanizmach zaimplementowanych w interfejsie użytkownika aplikacji działającej po stronie klienta i pominięcie weryfikacji po stronie serwera. Najczęściej jest to spotykane w aplikacjach desktopowych, gdzie nieco trudniej zmodyfikować kod klienta (co wcale nie znaczy, że jest to bardzo skomplikowane), ale dotyczy to też wielu aplikacji webowych.

Rozważmy najprostszy i najbardziej ewidentny przykład, w którym kontrola dostępu do danych odbywa się przez wyłączenie części elementów interfejsu graficznego. Przyjmijmy, że powstał portal, w którym dostępna jest lista wiadomości (oczywiście w praktyce mogą to być dowolne inne zasoby). Część z nich ma być przeznaczona tylko dla zalogowanych użytkowników, ale tytuły wszystkich wiadomości mają być wyświetlane zawsze. Tego typu zadanie realizuje się poprzez wyłączenie części elementów interfejsu graficznego. Tytuły niedostępne dla niezalogowanych gości są wyświetlane jako zwykły tekst, nieaktywne elementy list rozwijanych itp. Strona może więc wyglądać tak jak na rysunku 1.1. Wiadomości dla wszystkich są wyświetlane jako aktywne elementy listy, a wiadomości dla zalogowanych jako elementy nieaktywne.

Programista często zakłada, że wyłączenie elementu interfejsu jest wystarczające, aby zablokować dostęp osobom niepowołanym, co oczywiście jest błędem. Wbrew pozorom problemy tego typu wcale nie dotyczą tylko niedoświadczonych deweloperów. To jeden z powtarzających się błędów, stale zajmujący czołowe miejsca na listach najczęściej spotykanych luk w zabezpieczeniach. O ile początkujący programiści popełniają go z niewiedzy, o tyle ci doświadczeni zwykle z powodu przeoczenia.



Rysunek 1.1. *Widok witryny w listą tytułów wiadomości*

W tym przypadku usterka wydaje się oczywista. To jednak tylko prosta ilustracja problemu, który może się pojawiać w różnych wariantach. Fragment generujący listę widomości miałby w tym przypadku zapewne kod podobny do przedstawionego na listingu 1.1 (pełny kod przykładu znajduje się w podkatalogu *r01examp01a*).

Listing 1.1. *Kod generujący listę wiadomości*

```
<form action="index.php" method="get">
  <select size='6' name="id">
    <?php foreach($msgs as $msg):
       if(!isset($_SESSION['zalogowany']) && $msg['registered']){
         $disabledstr = 'disabled="disabled"';
       }
       else{
         $disabledstr = '';
       }
    ?>
    <option value="<?php echo $msg['id']?>"
      <?php echo $disabledstr; ?>><?php echo $msg['title']?>
    </option>
    <?php
      endforeach;
    ?>
  </select>
  <input type="hidden" name="action" value="showmsg">
  <input type="submit" value="Sprawdź wiadomość">
</form>
```

Jest to kod szablonu, do którego została przekazana lista wiadomości (np. pobrana z bazy danych) w postaci tablicy `$msgs`. Każdy element tej tablicy jest osobną tablicą asocjacyjną zawierającą klucze wskazujące identyfikator (`id`), tytuł (`title`), treść (`body`) oraz informację, czy wiadomość jest przeznaczona tylko dla zalogowanych użytkowników (`registered`). To, czy użytkownik jest zalogowany, czy też nie, jest wskazywane przez zmienną sesji o nazwie `zalogowany`.

A zatem gdy warunek `isset($_SESSION['zalogowany'])` jest nieprawdziwy (użytkownik niezalogowany) oraz wartość klucza `registered` jest interpretowana jako `true` (wiado-

mość tylko dla zalogowanych), w zmiennej `$disabledstr` zapisywany jest ciąg wyłączający dany element listy (`disabled="disabled"`); w przeciwnym przypadku zmienna ta otrzymuje pusty ciąg znaków.

Powstaną więc opcje w dwóch wersjach. Pierwsza w postaci:

```
<option value="identyfikator">tytuł wiadomości</option>
```

i druga w postaci:

```
<option value="identyfikator" disabled="disabled">tytuł wiadomości</option>
```

Przyglądając się znacznikom tworzącym formularz, można łatwo stwierdzić, że aby uzyskać treść wiadomości, skryptowi *index.php* (`action="index.php"`) należy za pomocą metody `GET` (`method="get"`) przekazać parametr `action` o wartości `showmsg` (pole `input` typu `hidden` z atrybutem `name` o wartości `action` i atrybutem `value` o wartości `showmsg`) i parametr `id` (wartość atrybutu `name` znacznika `<select>`) wskazujący identyfikator wiadomości. W kodzie źródłowym HTML takiej witryny znajdziemy zatem fragment podobny do widocznego na listingu 1.2.

Listing 1.2. *Fragment strony generowanej przez kod PHP*

```
<select size='6' name="id">
  <option value="1">Wiadomość dla wszystkich</option>
  <option value="2" disabled="disabled">Wiadomość tylko dla zalogowanych</option>
  <option value="3" disabled="disabled">Wiadomość również tylko dla
  ↪zalogowanych</option>
  <option value="4">Inna wiadomość dla wszystkich</option>
  <option value="5">Kolejna wiadomość dla wszystkich</option>
</select>
```

Wyświetlanie pojedynczych wiadomości może być realizowane w osobnym szablonie przez fragment kodu przedstawiony na listingu 1.3. Szablon otrzymuje tablicę `$msg` (o kluczach opisanych wyżej) zawierającą dane dotyczące wiadomości. Jeżeli zmienna `$msg` nie jest pusta, wyświetlane są tytuł (dane znajdujące się pod kluczem `title`) oraz treść (dane znajdujące się pod kluczem `body`). W przeciwnym przypadku wyświetlany jest tekst z informacją o braku wiadomości (tekst „Nie ma nic do wyświetlenia.”).

Listing 1.3. *Fragment szablonu wyświetlającego treść wiadomości*

```
<div>
  <?php
    if($msg):
      echo $msg['title'];
      echo '<br />';
      echo $msg['body'];
    else:
      echo 'Nie ma nic do wyświetlenia.';
    endif;
  ?>
</div>
```

Za przygotowanie wiadomości do wyświetlenia (dane dla zmiennej `$msg`) może odpowiadać funkcja `showMsg` przedstawiona na listingu 1.4. To ona jest w tym przykładzie kluczowa. W zaprezentowanej postaci sprawdzane jest, czy za pomocą metody `GET` został przekazany parametr o nazwie `id` (czy w tablicy `$_GET` znajduje się klucz o nazwie `id`), a jeśli tak, do zmiennej `$msg` trafia rezultat wywołania metody pobierającej wiadomość (metoda `getMsg`), której przekazywana jest wartość odczytana z tablicy `$_GET` (czyli wartość otrzymanego parametru).

Listing 1.4. *Funkcja odpowiedzialna za wyświetlenie wiadomości*

```
function showMsg()
{
  $msg = null;
  if(isset($_GET['id'])){
    $msg = $this->getMsg($_GET['id']);
  }
  include 'tmpl/msg.tpl';
}
```

Tak przygotowany kod portalu zawiera rzecz jasna podstawowy błąd, który w tym przypadku wydaje się wręcz oczywisty. Owszem, wyłączenie części elementów listy będzie skutecznym zabezpieczeniem w przypadku 80 – 90% typowych użytkowników internetu, ale tak naprawdę nie jest żadnym zabezpieczeniem. Wystarczy samodzielnie utworzyć odnośnik do skryptu *index.php* wraz z opisanymi wyżej parametrami, aby uzyskać dostęp do wiadomości, np.:

```
http://nazwa.domeny/index.php?id=2&action=showmsg
```

Poszczególne identyfikatory wiadomości są przecież wyraźnie widoczne w kodzie HTML wysłanym do przeglądarki (wyróżnione fragmenty na listingu 1.2). Ponieważ ani w funkcji `showMsg` (z listingu 1.4), ani w żadnym innym miejscu kodu serwisu nie występuje weryfikacja uprawnień użytkownika, wiadomość zostanie pokazana każdemu, kto użyje opisanego odnośnika, niezależnie od tego, czy jest zalogowany, czy też nie.

W sieci można napotkać informacje, że w tego typu sytuacji lepiej zamiast metody `GET` użyć metody `POST`, gdyż dane nie są wtedy dołączane do adresu URL i ich nie widać, a poza tym nie da się łatwo przygotować żądania typu `POST` (zmodyfikowane pliki są dostępne w katalogu *r01examp01b*). W kodzie formularza z listingu 1.1 miałaby się więc zmienić wartość atrybutu `method`:

```
<form action="index.php" method="post">
```

a w kodzie funkcji z listingu 1.4 pojawiłyby się odwołania do `$_POST`:

```
if(isset($_POST['id'])){
  $msg = $this->getMsg($_POST['id']);
```

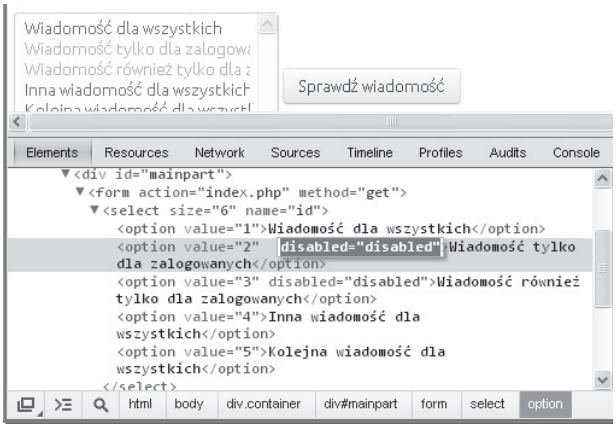
Oczywiście taki pomysł nie ma większego sensu. Po pierwsze, wcale nie trzeba samodzielnie tworzyć żądania typu `POST`. W przeglądarce można przecież dowolnie modyfikować otrzymany kod HTML, a więc i bez najmniejszych problemów da się odblokować nieaktywne elementy listy.

Wystarczy włączyć narzędzia deweloperskie dostępne standardowo już w prawie każdym produkcie i zmodyfikować żądany fragment kodu HTML. W tym przypadku wystarczyłoby usunąć atrybut `disabled` z elementu listy, który chcemy uaktywnić (rysunek 1.2). Wtedy tytuł danej wiadomości stanie się aktywny (rysunek 1.3) i będzie można użyć przycisku wyświetlającego wiadomości.

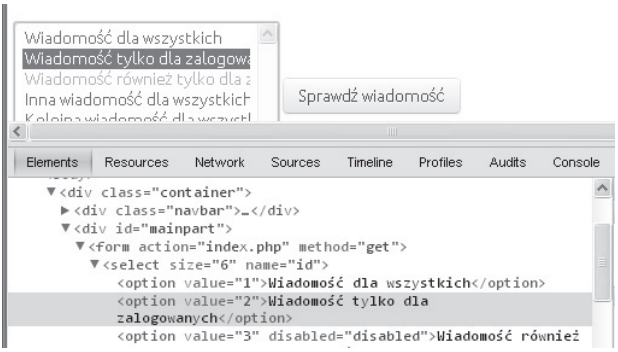
Narzędzia deweloperskie

Obecnie praktycznie w każdej współczesnej przeglądarce dostępne są narzędzia deweloperskie, które pozwalają kontrolować większość aspektów technicznych związanych z witryną, w tym bezpośrednio wpływać na zawartość strony. W przeglądarkach Chrome i Internet Explorer obecne są standardowo i można je wywołać, wciskając klawisz *F12* lub wybierając z menu podręcznego (pozycja menu *Zbadaj element* lub o podobnej nazwie). W Firefoksie najlepiej zainstalować rozszerzenie Firebug.

Rysunek 1.2. *Ingerencja w kod źródłowy strony*

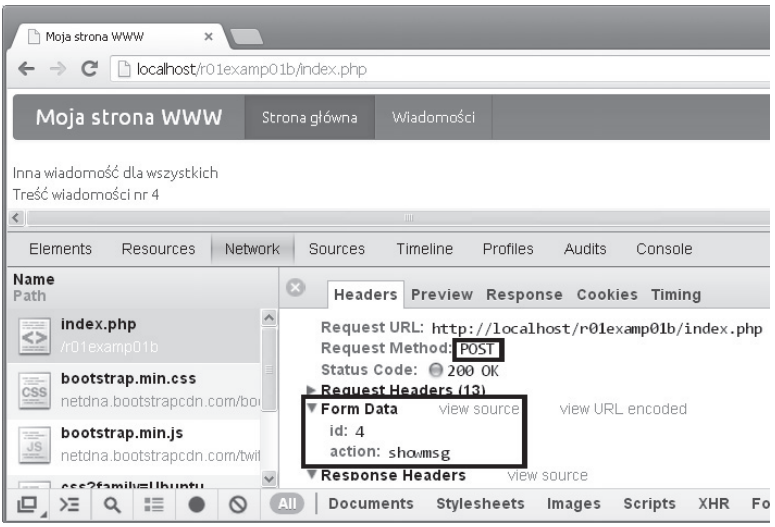


Rysunek 1.3. *Odblokowany element interfejsu*



Po drugie, metoda `POST` nie jest żadnym zabezpieczeniem, a dane nie są nigdzie ukrywane, ale przesyłane w nagłówkach żądania HTTP. Można je bez problemów obejrzeć, korzystając z narzędzi deweloperskich lub też z dodatkowych wtyczek do przeglądarek (np. Live HTTP Headers dla Firefoksa). W przypadku przeglądarki Chrome wystarczy wykonać żądanie (wskazać na stronie jedną z wiadomości i kliknąć przycisk *Sprawdź wiadomość*), wcisnąć klawisz *F12*, przejść na zakładkę *Network*, a następnie

w kolumnie *Name* wskazać URL żądania (*index.php*). W panelu po prawej stronie pojawią się wszelkie informacje dotyczące żądania oraz odpowiedzi serwera. Widoczne więc będą też parametry przekazane do serwera za pomocą metody `POST` (rysunek 1.4).



Rysunek 1.4. *Podglądanie nagłówków HTTP w przeglądarce Chrome*

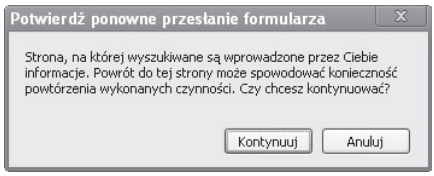
Po trzecie, konstrukcja żądania typu `POST` jest prosta. W bardziej skomplikowanych przypadkach można użyć np. skryptu PHP, ale najprostszym rozwiązaniem jest stworzenie własnego formularza z odpowiednimi parametrami. Można go przygotować nawet na kilka sposobów. Wiadomo przecież, jakie parametry mają być przesłane, więc istnieją różne możliwości ich przekazania. Aby dostać się do wiadomości numer 2, taki formularz mógłby mieć postać przedstawioną na listingu 1.5.

Listing 1.5. *Formularz pozwalający na skorzystanie z metody POST*

```
<form action="http://nazwa.domeny/r01examp01b/index.php" method="post">
  <input type="text" name="id" value="2">
  <input type="hidden" name="action" value="showmsg">
  <input type="submit" value="Sprawdź wiadomość" class="btn">
</form>
```

Po czwarte, metody `GET` i `POST` nie są przeznaczone do zamiennego stosowania. Pierwszej z nich używa się do zadań, które nie modyfikują stanu serwera, natomiast drugiej — wręcz odwrotnie — do zadań, które modyfikują jego stan (np. zmiana danych)[1]. Metody `POST` nie należy zatem stosować w przypadku zwykłego wyświetlania informacji, zwłaszcza że próba odświeżenia strony za każdym razem będzie skutkowała wyświetleniem użytkownikowi komunikatu z prośbą o potwierdzenie chęci wykonania operacji, podobnego do przedstawionego na rysunku 1.5.

[1] Nie jest to w pełni ścisła definicja. Metoda `GET` powinna być stosowana do wywołań idempotentnych, czyli takich, których wielokrotne wywołanie daje taki sam efekt, a metoda `POST` — do wywołań nieidempotentnych. Więcej informacji znajduje się w dokumentach opisujących protokół HTTP.



Rysunek 1.5. *Prośba o potwierdzenie chęci ponownego wykonania operacji*

Zabezpieczanie dostępu do danych

Skoro kod omawiany w pierwszym rozdziale zawiera błędy, trzeba je poprawić. Można w nim dostrzec co najmniej trzy nieprawidłowości. Rzecz pierwsza to ujawnianie zbyt wielu informacji, co wynika z zastosowania jednolitego schematu generowania elementów interfejsu. Skoro bowiem niezalogowany użytkownik nie powinien mieć dostępu do części wiadomości, to nie ma też potrzeby, aby ujawniać w kodzie wszystkie identyfikatory. Taką zasadę należy przyjąć jako wskazówkę ogólną — zawsze przekazujemy na zewnątrz tylko tyle informacji, ile jest faktycznie niezbędne. W tym konkretnym przypadku można całkowicie pominąć atrybut `value` znacznika `option` dla wiadomości nieaktywnych lub też zastosować pewną wartość specjalną, np. –1. Fragment kodu generującego listę wiadomości mógłby więc przyjąć postać przedstawioną na listingu 1.6.

Listing 1.6. *Zastosowanie wartości specjalnej dla części opcji*

```
<select size='6' name="id">
  <?php foreach($msgs as $msg):
     if(!isset($_SESSION['zalogowany']) && $msg['registered']){
       $disabledstr = 'disabled="disabled"';
       $id = '-1';
     }
     else{
       $disabledstr = '';
       $id = $msg['id'];
     }
  ?>
  <option value="<?php echo $id; ?>"
    <?php echo $disabledstr; ?>><?php echo $msg['title']?></option>
  <?php
    endforeach;
  ?>
</select>
```

Po takiej zmianie wynikowy kod HTML listy przyjmie postać widoczną na rysunku 1.6. Wszystkie znaczniki `<option>` odpowiadające wiadomościom niedostępnym dla niezalogowanych użytkowników otrzymały atrybut `value` o wartości –1. Ich identyfikatory nie zostaną zatem przekazane. Oczywiście to nie jest jeszcze zabezpieczenie, ale dobra praktyka, aby nie ujawniać zbyt wielu informacji. W tym przypadku można się też łatwo domyślić, że brakujące identyfikatory to prawdopodobnie 3 i 4. W praktyce, rzecz jasna, zapewne nie będą one kolejnymi liczbami i nie będzie to już tak proste do stwierdzenia (ten problem będzie poruszony w podrozdziale „Kwestia enumeracji zasobów").

```
▼<select size="6" name="id">
   <option value="1">Wiadomość dla wszystkich</option>
   <option value="-1" disabled="disabled">Wiadomość tylko dla zalogowanych</option>
   <option value="-1" disabled="disabled">Wiadomość również tylko dla zalogowanych</option>
   <option value="4">Inna wiadomość dla wszystkich</option>
   <option value="5">Kolejna wiadomość dla wszystkich</option>
</select>
```

Rysunek 1.6. *Wynikowy kod HTML generowany przez skrypt z listingu 1.6*

Jednak podstawowym błędem w dotychczasowym kodzie był brak weryfikacji uprawnień przy wyświetlaniu wiadomości. Należałoby więc poprawić kod z listingu 1.3, tak aby wiadomość z ustawioną flagą `registered` (klucz `registered` w tablicy `$msg`) była wyświetlana tylko wtedy, gdy jednocześnie ustawiona jest zmienna sesji `zalogowany`. Taką poprawkę można wprowadzić w sposób przedstawiony na listingu 1.7.

Listing 1.7. *Sprawdzanie uprawnień do wyświetlania wiadomości*

```
<?php
  if($msg):
    if(!$msg['registered'] ||
       ($msg['registered'] && isset($_SESSION['zalogowany']))):
      echo $msg['title'];
      echo '<br />';
      echo $msg['body'];
    else:
      echo 'Brak uprawnień.';
    endif;
  else:
    echo 'Nie ma nic do wyświetlenia.';
  endif;
?>
```

Tym razem, jeżeli istnieje wiadomość o wskazanym identyfikatorze (zmienna `$msg` nie jest pusta), dodatkowo badany jest złożony warunek stwierdzający, czy może być ona pokazana. Wyświetlona będzie tylko wtedy, gdy nie ma statusu `registered` lub też gdy ma status `registered`, ale użytkownik jest zalogowany (istnieje zmienna sesji `zalogowany`).

Trzeci ze wspomnianych błędów jest również związany z funkcją `showMsg`. Należy zwrócić uwagę, że wartość odczytana z indeksu `id` tablicy `$_GET` jest przekazywana bezpośrednio do funkcji pobierającej wskazaną wiadomość. W tym konkretnym przypadku nie stanowi to dużego zagrożenia, ale jest to bardzo zła praktyka. Wszelkie dane otrzymywane z zewnątrz zawsze należy traktować jako niezaufane i weryfikować ich poprawność. Nigdy nie można zakładać, że pochodzą one z „naszego" formularza, bo też nigdy nie wiadomo, kto i co nam przysłał. Można użyć jednej z funkcji filtrujących (np. `filter_input`), a jeśli wiadomo, że spodziewana jest wartość całkowita (jak w omawianym przykładzie), także zwykłej konwersji na typ `int` (warto też się zastanowić, czy nie będzie przydatne ograniczenie dopuszczalnego zakresu wartości).

Dodatkowo ze względu na zmiany wprowadzone w szablonie generującym listę i konieczność obsługi wartości –1, wskazującej, że wiadomość nie powinna być pokazywana, warto sprawdzać stan parametru `id`.

Kod funkcji showMsg można więc zmienić w sposób przedstawiony na listingu 1.8 (pełny kod przykładu znajduje się w podkatalogu *r01examp01c*).

Listing 1.8. *Poprawiona wersja funkcji showMsg*

```
function showMsg()
{
  $msg = null;
  if(isset($_GET['id'])){
    $id = intval($_GET['id']);
    //lub
    //$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT);
    if($id > 0){
      $msg = $this->getMsg($id);
    }
  }
  include 'tmpl/msg.tpl';
}
```

Po sprawdzeniu, że do skryptu za pomocą metody GET został przekazany parametr o nazwie id, jego wartość jest konwertowana na typ całkowitoliczbowy i przypisywana pomocniczej zmiennej $id. Metoda getMsg, pobierająca treść wiadomości, jest wywoływana tylko wtedy, gdy wartość zapisana w $id jest większa od 0. W przeciwnym przypadku (co oznacza nieprawidłową wartość parametru) zmienna $msg pozostaje pusta.

Gdzie sprawdzać uprawnienia?

Prezentowany przykładowy kod jest uproszczony i weryfikacja uprawnień odbywa się w szablonie wyświetlającym dane. Takie rozwiązania są co prawda stosowane, jednak w praktyce zasadne byłoby przeniesienie podejmowania decyzji o wyświetlaniu do logiki aplikacji — w tym przypadku zapewne do funkcji showMsg. Komunikat o braku uprawnień mógłby być wtedy wyświetlany w osobnej strukturze przeznaczonej dla wiadomości systemowych.

Kwestia enumeracji zasobów

Jak wspomniano w poprzednim podrozdziale przy omawianiu przykładu z listingu 1.6, samo ukrycie identyfikatorów zasobów (w tym przypadku — wiadomości) nie stanowi żadnego zabezpieczenia przed atakiem. Jeśli nie zostanie wprowadzona weryfikacja uprawnień, jak zostało to pokazane na listingach 1.7 i 1.8, ukryte informacje zawsze będzie można odczytać. Oczywiście atakujący rzadko kiedy będzie miał tak komfortową sytuację, że identyfikatory będą miały kolejne numery, ale nawet ich losowość spowodowałaby jedynie niewielkie utrudnienie. Istnieje jednak prawie stuprocentowa gwarancja, że wcześniej czy później znajdzie się ktoś, kto zechce sprawdzić, co też kryje się pod różnymi identyfikatorami.

Załóżmy zatem, że ze względu na zmianę projektu i przeniesienie weryfikacji uprawnień z szablonu do logiki aplikacji weryfikacja w szablonie została już wyłączona, ale jeszcze nie została zaimplementowana w innym miejscu. Oczywiście taki kod nigdy

nie powinien się znaleźć w środowisku produkcyjnym, ale przez przeoczenie mogło się tak zdarzyć[2]. Użytkownikom niezalogowanym nie są też pokazywane tytuły wiadomości przeznaczonych tylko dla użytkowników zalogowanych (przykład w podkatalogu *r01examp01d*) lub też są pokazywane, ale bez właściwych identyfikatorów (jak w przykładzie z listingu 1.6).

Wystarczy zatem, że jakiś użytkownik zacznie wpisywać w pasku adresowym przeglądarki adresy URL w postaci:

```
http://nazwa.domeny/index.php?id=liczba&action=showmsg
```

i będzie w stanie dotrzeć do dowolnej wiadomości. Badanie tego typu to enumeracja zasobów. Oczywiście ręczne sprawdzenie wszystkich możliwych identyfikatorów byłoby bardzo uciążliwe i czasochłonne, zatem ktoś, kto naprawdę chciałby uzyskać dostęp do wszystkich wiadomości, z pewnością napisałby odpowiedni skrypt. Nie jest to skomplikowane zadanie — wystarczy skorzystać np. z biblioteki cURL. Skrypt „wyciągający" wiadomości z systemu mógłby wyglądać tak jak na listingu 1.9.

Listing 1.9. *Enumeracja wiadomości*

```
<?php
$url = 'http://localhost/r01examp01d/index.php?action=showmsg&id=';

for($i = 1; $i <= 100; $i++){
  echo "Przetwarzanie id = $i\n";
  $ch = curl_init($url . $i);
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
  curl_setopt($ch, CURLOPT_HEADER, false);

  $str = curl_exec($ch);
  curl_close($ch);

  $res = "--- Wartość $i ---\n";
  if(preg_match('/<div id=\'mainpart\'.*?>(.*?)<\/div>/s', $str, $matches)){
    $res .= $matches[1] . "\n\n";
  }
  else{
    $res .= "Brak wzorca.\n\n";
  }
  file_put_contents('msgs.txt', $res, FILE_APPEND);
}
?>
```

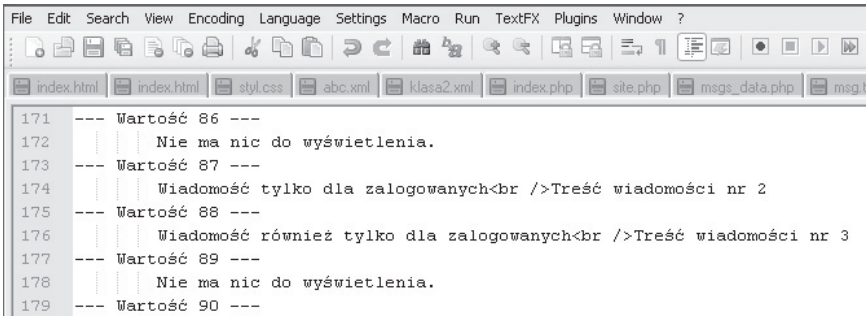
Adres URL serwisu został dla wygody zapisany w zmiennej `$url` znajdującej się na początku skryptu. W pętli `for` generowane są kolejne wartości parametru `id` — od 1 do 100. Sesja cURL jest inicjalizowana za pomocą funkcji `curl_init`, której w postaci argumentu przekazywany jest pełny adres odwołujący się do wiadomości o danym identyfikatorze. Adres ten powstaje z połączenia wartości zmiennej `$url` z wartością zmiennej `$i`. Dzięki funkcji `curl_setopt` ustawiane są opcje dodatkowe. Opcja `CURLOPT_RETURNTRANSFER` ustawiona na `true` oznacza, że dane pobrane z sieci mają być zwrócone jako rezultat działa-

[2] Niestety takie sytuacje naprawdę zdarzają się w praktyce.

nia funkcji `curl_exec`, a `CURLOPT_HEADER` ustawiona na `false` określa, że nie interesują nas nagłówki protokołu HTTP, ale sama treść pobieranej strony.

Po wywołaniu funkcji `curl_exec` następuje nawiązanie połączenia z adresem ustawionym w wywołaniu `curl_init` oraz zwrócenie pobranych danych i zapisanie ich w zmiennej `$str`. Ta zmienna będzie więc zawierała ciąg znaków będący odpowiedzią serwera, a więc treść źródłową strony WWW. Ponieważ interesuje nas jedynie część zawierająca treść wiadomości, zostało użyte wyrażenie regularne wraz z funkcją `preg_match`. Dzięki temu w zmiennej `$matches` znajdzie się tablica zawierająca fragmenty tekstu „wyciągnięte" za pomocą wyrażenia ze strony HTML. W drugiej komórce (o indeksie 1) tej tablicy będzie dostępna treść znajdująca się między znacznikiem `<div>` z atrybutem `id` o wartości `mainpart` a najbliższym znacznikiem zamykającym `</div>`.

Wyniki są zapisywane w pliku *msgs.txt* za pomocą funkcji `file_put_contents`. Flaga `FILE_APPEND` wskazuje, że zawartość ma być dopisywana na końcu pliku, tak aby poprzednia treść nie została skasowana. W rezultacie w pliku wynikowym znajdą się informacje o tym, dla których identyfikatorów są dostępne wiadomości oraz jaka jest ich treść (rysunek 1.7). Widać wyraźnie, że tym razem wiadomości dla zalogowanych użytkowników uzyskane ze względu na brak odpowiedniej weryfikacji uprawnień mają identyfikatory 87 i 88.



Rysunek 1.7. *Fragment zawartości pliku z wynikami enumeracji*

Sam skrypt najlepiej uruchamiać w konsoli (wierszu poleceń) — ze względu na zastosowanie instrukcji `echo` można będzie wtedy obserwować postępy jego działania. Warto przy tym zauważyć, że w kodzie nie zostały wprowadzone żadne ograniczenia, a więc program będzie zużywał całe dostępne dla niego zasoby, w tym czas procesora i pasmo łącza sieciowego.

Problem enumeracji nie ogranicza się tylko do odszukiwania danych, które nie zostały w odpowiedni sposób zabezpieczone, ale dotyczy wszelkich danych. W typowych sytuacjach ograniczenie liczby żądań, dostępnego pasma, czasu procesora czy pamięci jest wprowadzane z poziomu firewalla i serwera WWW, używane są również rozwiązania typu *load balancer* itp.

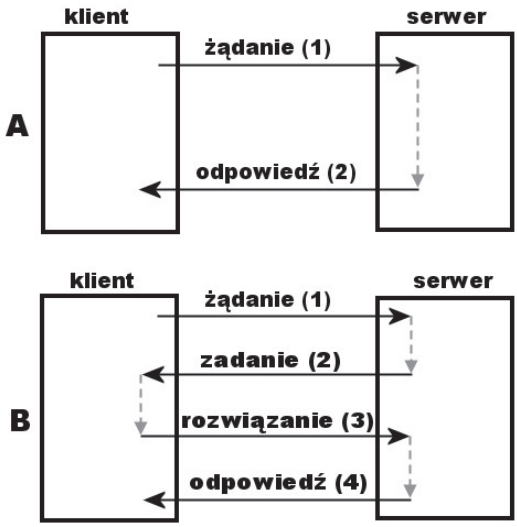
Na poziomie aplikacji można jednak dodatkowo stosować systemy typu *proof-of-work*. Są one przydatne, gdy na serwerze znajdują się zasoby, których koszt udostępniania jest znaczący, a my nie chcemy, aby można je było pobierać w sposób nieograniczony

(jak w przykładzie z listingu 1.9), ale także w takich celach jak np. zapobieganie spamowi. Ogólnie rzecz ujmując — wszędzie tam, gdzie ważne jest, aby serwer mógł limitować żądania klienta.

Wzorzec *proof-of-work* (dosł. „udowodnij wykonaną pracę") polega na wymuszeniu na kliencie (aplikacji klienta) wykonania zadania wymagającego zużycia stosunkowo dużych zasobów, którego weryfikacja jest wielokrotnie łatwiejsza do wykonania i nie wymaga dużych zasobów (jedną z implementacji jest protokół *Client Puzzle Protocol*). Zadanie (z reguły pewne obliczenie) musi być tak dobrane, żeby w przypadku zwykłego korzystania z serwisu nie było zauważalnym obciążeniem dla klienta, natomiast w przypadku próby automatycznego przetwarzania danych — spowodowało znaczne zużywanie zasobów. Tym samym potencjalny atakujący będzie musiał dysponować silnym sprzętem (duża liczba procesorów, duża ilość pamięci itp.), co może sprawić, że atak będzie nieopłacalny.

Zatem zamiast typowej obsługi żądania (rysunek 1.8A) interakcja między klientem a serwerem wygląda wtedy jak na rysunku 1.8B. Klient wysyła najpierw żądanie pobrania zasobu (1). Następnie serwer przydziela zadanie do rozwiązania (2). Klient rozwiązuje zadanie (w przypadku aplikacji działających w przeglądarce może to być zrealizowane np. w JavaScripcie) i wysyła je do serwera (3). Serwer weryfikuje odpowiedź i — gdy jest ona prawidłowa — wysyła do klienta pierwotnie żądane informacje.

Rysunek 1.8. *Schemat postępowania w systemach proof-of-work*



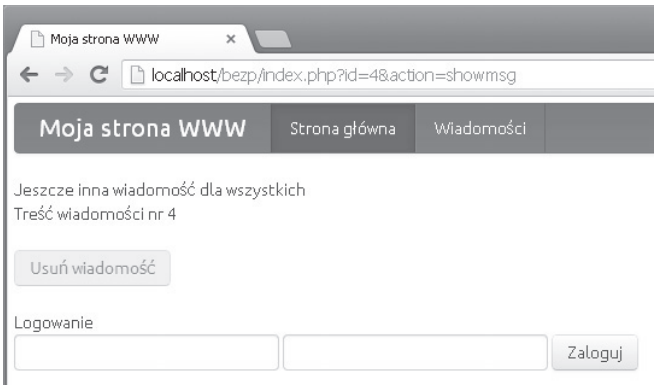
Kontrola dostępu do funkcji

Kontrola dostępu do funkcji udostępnianych użytkownikom portalu jest równie ważna jak kontrola dostępu do danych. To analogiczna sytuacja do opisywanych wyżej. Załóżmy, że lista wiadomości jest wyświetlana w sposób podobny do pierwszego przykładu (listing 1.1 i kolejne, rysunek 1.1), ale nie występuje podział na informacje dla

zalogowanych i niezalogowanych. Każdy może zatem przeglądać dane. Fragment kodu odpowiedzialnego za utworzenie listy wiadomości mógłby wyglądać następująco:

```
<select size='6' name="id">
  <?php foreach($msgs as $msg):?>
    <option value="<?php echo $msg['id']; ?>"><?php echo $msg['title']?></option>
  <?php endforeach; ?>
</select>
```

Jednak zalogowani użytkownicy będą mieli dodatkowo prawo do usuwania wiadomości, a do wykonania tej czynności posłuży np. przycisk dostępny na ekranie konkretnej wiadomości. Ekran ten mógłby mieć więc postać przedstawioną na rysunku 1.9.



Rysunek 1.9. *Wiadomość z przyciskiem pozwalającym na jej usunięcie*

Mogą tu wystąpić dwa warianty związane z interfejsem aplikacji. W pierwszym przycisk jest wyświetlany zawsze, niezależnie od tego, czy użytkownik jest zalogowany, czy też nie. Zmienia się jedynie stan tego elementu, który będzie aktywny po zalogowaniu. Kod szablonu generującego ekran wiadomości mógłby mieć więc postać przedstawioną na listingu 1.10 (pełny kod przykładu znajduje się w katalogu *r01examp02a*).

Listing 1.10. *Podejmowanie decyzji o stanie przycisku*

```
<div id='mainpart'>
  <?php
    if($msg):
      echo $msg['title'];
      echo '<br />';
      echo $msg['body'];
      if(!isset($_SESSION['zalogowany'])):
        $disabled = 'disabled="disabled"';
      else:
        $disabled = '';
      endif;
   ?>
   <div>
    <form action="index.php" method="post">
      <input type="submit" value="Usuń wiadomość"
        <?php echo $disabled; ?>>
      <input type="hidden" name="action" value="deletemsg">
      <input type="hidden" name="id" value="<?php echo $msg['id']; ?>">
    </form>
```

```
    </div>
     <?php
      else:
        echo 'Nie ma nic do wyświetlenia.';
      endif;
    ?>
  </div>
```

Jeżeli zmienna `$msg` jest ustawiona (czyli są dostępne dane), wiadomość jest wyświetlana oraz ustalany jest stan pomocniczej zmiennej `$disabled`, która pozwoli wyłączyć przycisk usuwania dla niezalogowanych użytkowników. A zatem gdy nie istnieje zmienna sesji `zalogowany`, w zmiennej `$disabled` zapisywany jest atrybut `disabled` (dla znacznika `<input>`), a w przeciwnym przypadku zmienna ta otrzymuje pusty ciąg znaków. Pod wiadomością znajduje się formularz z przyciskiem *Usuń wiadomość*. Formularz jest przekazywany do skryptu *index.php* za pomocą metody `POST`. Widać, że przekazywane są parametry `action` oraz `id`. Pierwszy wskazuje akcję, która ma być wykonana (w tym przypadku `deletemsg`), a drugi — identyfikator wiadomości.

W drugim wariancie przycisk jest wyświetlany tylko zalogowanym użytkownikom. Kod szablonu będzie więc podobny do przedstawionego na listingu 1.10, formularz z przyciskiem będzie jednak w innym miejscu, a warunek instrukcji `if` będzie odwrotny. Omawiany fragment skryptu przyjmie zatem postać widoczną na listingu 1.11 (powtarzające się fragmenty zostały pominięte; pełny kod przykładu znajduje się w katalogu *r01examp02b*).

Listing 1.11. *Ustalenie, czy przycisk ma się znaleźć w formularzu*

```
<div id='mainpart'>
  <?php
    if($msg):
      //tutaj instrukcje wyświetlające wiadomość
      if(isset($_SESSION['zalogowany'])):
      ?>
      <div>
        <!-- tutaj formularz z przyciskiem usuwania -->
      </div>
      <?php
      endif;
   ?>
   <?php
    else:
      echo 'Nie ma nic do wyświetlenia.';
    endif;
  ?>
</div>
```

A zatem po wyświetleniu wiadomości badane jest, czy istnieje zmienna sesji `zalogowany` (czyli czy mamy do czynienia z zalogowanym użytkownikiem). Jeśli tak, do kodu wynikowego dołączany jest formularz. Jeśli nie, formularz jest pomijany (formularz będzie miał taką samą treść jak na listingu 1.10).

Można założyć, że wiadomość będzie usuwana za pomocą funkcji `deletemsg` wywoływanej bezpośrednio po otrzymaniu przez skrypt parametru `action` o wartości `deletemsg`. Funkcja ta będzie sprawdzała, czy dostępny jest parametr `id`, usuwała wiadomość i ustawiała komunikat wyświetlany użytkownikowi w odpowiedzi na wykonaną akcję. Załóżmy, że kod wygląda tak jak na listingu 1.12.

Listing 1.12. *Funkcja usuwająca wiadomość*

```
function deletemsg()
{
  $komunikat = 'Błędne dane. Akcja anulowana.';
  if(isset($_POST['id'])){
    $id = intval($_POST['id']);
    if($id > 0){
      //tutaj kod usuwający wiadomość z bazy
      $komunikat = 'Wiadomość została usunięta.';
    }
  }
  $this->setMessage($komunikat);
}
```

Sprawdzamy, czy w tablicy `$_POST` istnieje klucz (indeks) `id`. Jeśli tak jest, jego wartość jest odczytywana, konwertowana na typ całkowity i zapisywana w zmiennej `$id`. Gdy wartość tej zmiennej jest większa od 0, wiadomość o wskazanym identyfikatorze jest usuwana (w przykładowych plikach dostępnych na FTP został zawarty przykładowy kod symulujący usunięcie wiadomości).

Widać wyraźnie, że przy takiej realizacji w projekcie występują błędy tego samego typu, jaki był omawiany w podrozdziałach dotyczących kontroli dostępu do danych. Jeśli została zastosowana wersja z listingu 1.10, niezalogowany użytkownik na podstawie analizy kodu źródłowego jest w stanie łatwo stwierdzić, jakie parametry należy przesłać do skryptu, aby usunąć wybraną wiadomość. Może więc wysłać je bezpośrednio, np. konstruując własny formularz. Problemu nie stwarzają również modyfikacja interfejsu strony i uaktywnienie przycisku. Samo zalogowanie nie będzie konieczne, gdyż w funkcji `deletemsg` nie jest przeprowadzana weryfikacja uprawnień.

W przypadku kodu z listingu 10.11 atakujący będzie miał nieco większy problem, ponieważ nie jest w stanie w prosty sposób (bez zalogowania) stwierdzić, jakie parametry należy wysłać do serwera, aby wykasować wiadomość. Jeśli jednak zauważy, że do wyświetlenia stosowany jest parametr `action` o wartości `showmsg`, to zapewne wypróbuje różne kombinacje ze słowami „del", „delete" itp. Poza tym zalogowany użytkownik widzi już parametry, więc może je upowszechnić. Sama zmiana wartości parametru `action` z `deletemsg` na dowolną inną (mógłby to być przecież nic nie znaczący ciąg) też nie może być traktowana jako zabezpieczenie, a jedynie utrudnienie dla włamywacza. Weryfikacja uprawnień jest więc niezbędna i nie można o niej zapomnieć.

Sprawdzenie uprawnień użytkownika do usuwania wiadomości może być zrealizowane np. w sposób przedstawiony na listingu 1.13 (pełny kod przykładu dostępny jest w katalogu *r01examp02c*).

Listing 1.13. *Sprawdzanie uprawnień użytkownika do wykonania funkcji*

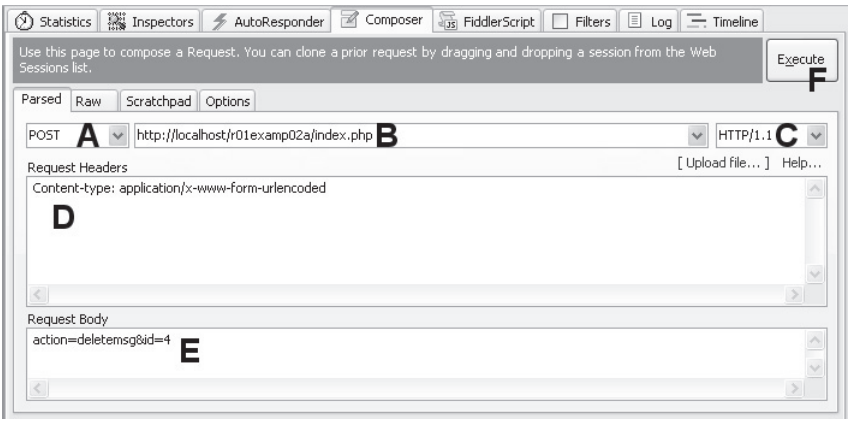
```
function deletemsg()
{
  if(!isset($_SESSION['zalogowany'])){
    $komunikat = 'Brak uprawnień. Akcja anulowana.';
  }
  else if(isset($_POST['id'])){
    //tutaj kod usuwający wiadomość z listingu 1.12
    $komunikat = 'Wiadomość została usunięta.';
  }
  else{
    $komunikat = 'Błędne dane. Akcja anulowana.';
  }
  $this->setMessage($komunikat);
}
```

Tym razem w funkcji `deletemsg` znajduje się złożona instrukcja warunkowa rozpatrująca możliwe sytuacje. Najpierw badane jest, czy mamy do czynienia z zalogowanym użytkownikiem, czyli czy istnieje zmienna sesji `zalogowany` (w tablicy `$_SESSION` znajduje się klucz o nazwie `zalogowany`). Jeśli zmiennej nie ma, bieżący użytkownik nie ma też uprawnień do kasowania wiadomości. Ustawiany jest wtedy odpowiedni komunikat i funkcja kończy działanie. Jeżeli użytkownik jest zalogowany, sprawdzane jest, czy za pomocą metody `POST` został przekazany parametr, a dalej wykonywany jest taki sam kod jak w przypadku przykładu z listingu 1.12 — usunięcie wiadomości (o ile wartość parametru jest prawidłowa). Ostatni przypadek ma miejsce, gdy użytkownik jest zalogowany, została wykonana akcja `deletemsg`, ale brakuje parametru `id`. Wtedy ustawiany jest komunikat informujący o nieprawidłowych danych.

Modyfikowanie żądań HTTP

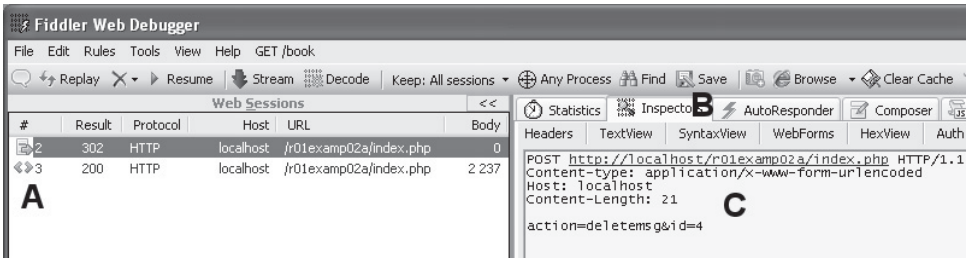
Na początku rozdziału zaprezentowano, że do utworzenia żądania typu `POST` można użyć zwykłego formularza (listing 1.5). Wygodniejsze jest jednak skorzystanie z narzędzi pozwalających na manipulację transmisją HTTP. Jednym z nich, często używanym przy testach penetracyjnych aplikacji internetowych, jest Fiddler. Po uruchomieniu tego programu w zakładce *Composer* (rysunek 1.10) można przygotować dowolne żądanie HTTP przesyłane za pomocą różnych wersji tego protokołu oraz różnych metod (w tym najpopularniejszych `GET` i `POST`, ale także innych).

W polu adresu (B) wpisujemy adres strony, np. *http://nazwa.domeny/r01examp02a/index.php*, jeśli chcemy skorzystać z przykładu z katalogu *r01examp02a*. Z listy z lewej strony (A) wybieramy metodę transmisji danych; dla wspomnianego przykładu będzie to `POST`. Lista z prawej strony (C) pozwala na wskazanie wersji protokołu HTTP. Powszechnie stosowana jest wersja HTTP/1.1 i jest to opcja domyślna, nie trzeba więc jej zmieniać. W polu *Request Headers* (D) należy wprowadzić nagłówki HTTP, które zostaną wysłane do serwera. Ponieważ za pomocą metody `POST` chcemy przekazać parametry — będą to `action` i `id` (przykład *r01examp02a*) — niezbędny będzie nagłówek `Content-Type`. W polu *Request Body* (E) należy podać nazwy parametrów i ich wartości zgodne z wybranym typem kodowania (w tym przypadku *x-www-form-urlencoded*).



Rysunek 1.10. *Tworzenie żądania HTTP*

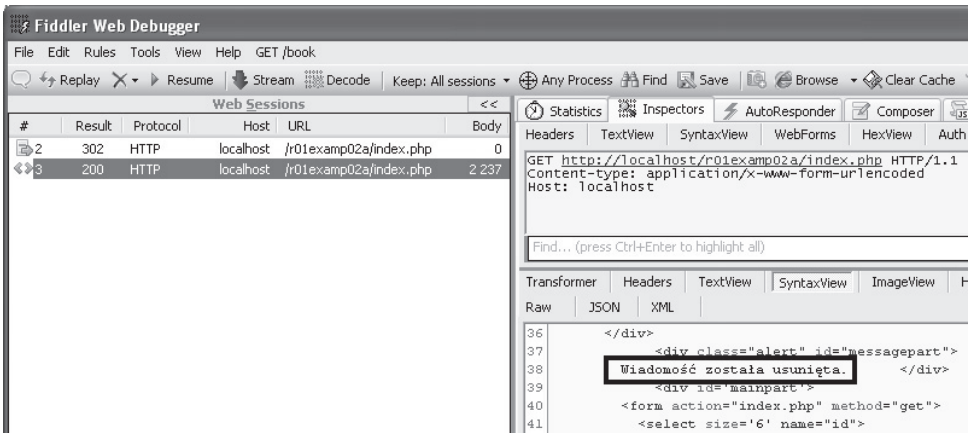
Po wprowadzeniu danych można wykonać żądanie, wciskając przycisk *Execute* (F). Na liście widocznej z lewej strony okna (rysunek 1.11A) pojawią się sesje nawiązane w związku z żądaniem. Widać wyraźnie, że żądanie wysłane za pomocą metody POST spowodowało zwrócenie nagłówka przekierowującego (kod odpowiedzi serwera 302), a więc aplikacja korzysta z wzorca PRG (ang. *Post/Redirect/Get*). Dopiero kolejne żądanie (kod odpowiedzi 200) zawiera treść strony. Po kliknięciu pierwszej sesji i wybraniu zakładki *Inspectors* (B) w polu źródłowym (C) można też zobaczyć szczegóły wysłanego żądania. Widać, że Fiddler dodał nagłówki *Host* (formalnie wymagany przez protokół HTTP w wersji 1.1) i *Content-Length*.



Rysunek 1.11. *Szczegóły żądania POST wysłanego do serwera*

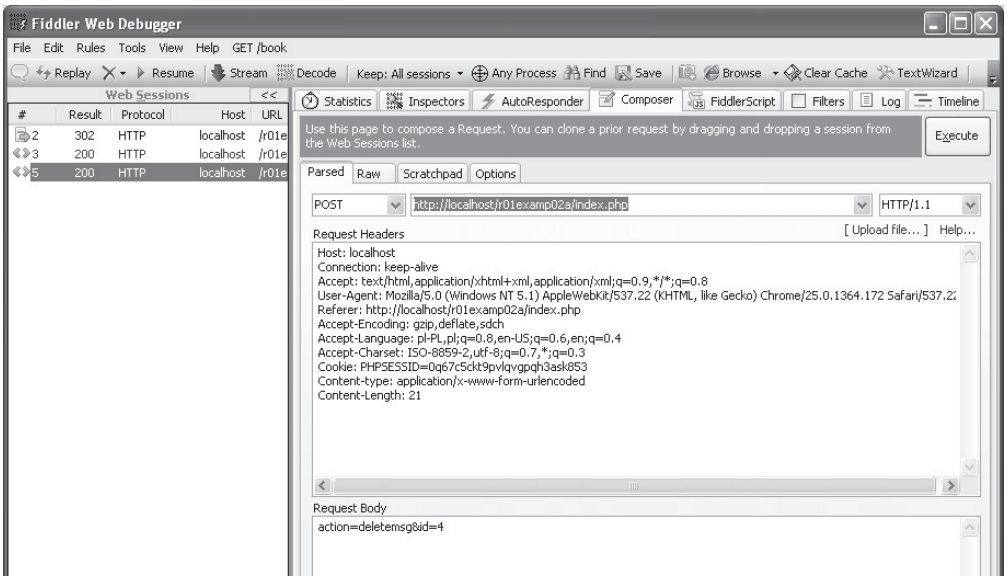
Klikając drugie odwołanie (rysunek 1.12) wykonane po przekierowaniu, oprócz treści żądania można w polu odpowiedzi serwera obejrzeć treść strony wynikowej. Po przejściu na zakładkę *SyntaxView* (o ile zostało zainstalowane odpowiednie rozszerzenie) widoczny będzie pokolorowany kod źródłowy, w którym da się odnaleźć komunikat o usunięciu z serwera wiadomości o identyfikatorze przekazanym w pierwszym żądaniu (oczywiście można też skorzystać z widoku tekstowego *TextView*, jednak nie będzie on tak czytelny).

W podanym przykładzie zostały wysłane jedynie trzy nagłówki: jeden wprowadzony ręcznie i dwa dodane przez Fiddlera ze względu na wymagania protokołu HTTP. Z reguły jednak serwer otrzymuje od przeglądarki cały zestaw nagłówków. Ich ręczne wpisywanie byłoby niewygodne i czasochłonne. Można jednak w prosty sposób użyć nagłówków



Rysunek 1.12. *Podgląd źródła strony pobranej po przekierowaniu*

z wcześniejszego żądania. Wystarczy w przeglądarce (lub w Fiddlerze) odwołać się do strony, a następnie przeciągnąć sesję, która pojawi się wtedy na liście z lewej strony, do zakładki *Composer*. Wszystkie nagłówki zostaną wówczas skopiowane do pola *Request Headers* (rysunek 1.13) i będzie można je dowolnie modyfikować. Jest to o tyle wygodne, że w ten sposób można podłączyć się pod istniejącą na serwerze sesję zainicjowaną przez zwykłe wywołanie strony w przeglądarce (skopiowany zostanie bowiem również nagłówek *Cookie* z identyfikatorem sesji, jak widać na rysunku 1.13).



Rysunek 1.13. *Kopiowanie istniejących nagłówków HTTP*

Bardzo użyteczną cechą Fiddlera jest możliwość zatrzymywania żądania oraz odpowiedzi serwera spełniających określony warunek. Po zatrzymaniu transmisji można dokonać dowolnych modyfikacji danych. Załóżmy np., że chcemy wyświetlić pewną

wiadomość z przykładu *r01examp02a*. W przeglądarce wpisujemy adres *http://localhost/r01examp02a/index.php* — w Fiddlerze pojawi się sesja odpowiadająca temu odwołaniu. Takie żądanie zostanie obsłużone w standardowy sposób, a treść wynikowa pojawi się w przeglądarce (treść żądania oraz odpowiedzi można przeglądać w odpowiednich panelach).

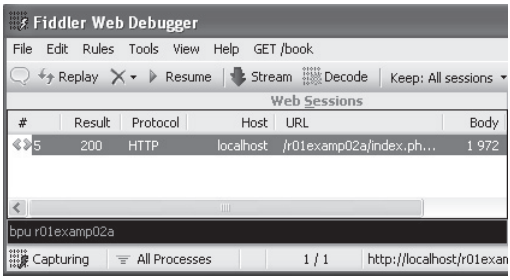
Ustawmy teraz pułapkę, która zatrzyma żądanie wyświetlenia konkretnej wiadomości (dzięki temu będzie można je podejrzeć i zmodyfikować). Powinna ona dotyczyć tylko odwołań do strony tego konkretnego przykładu. W polu pod listą (rysunek 1.14) wpisujemy ciąg:

```
bpu r01examp02a
```

lub pełny URL:

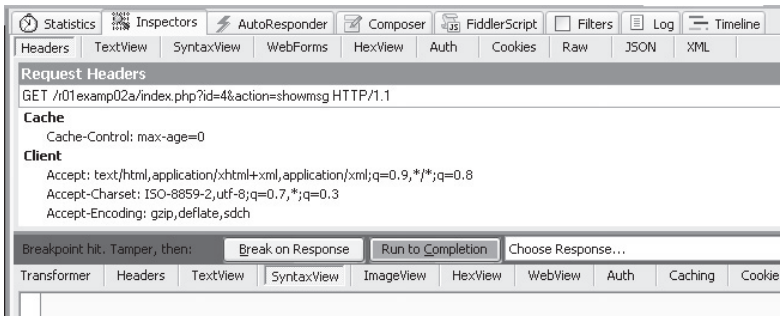
```
bpu http://nazwa.domeny/r01examp02a/index.php
```

Rysunek 1.14. *Ustawianie pułapki w Fiddlerze*

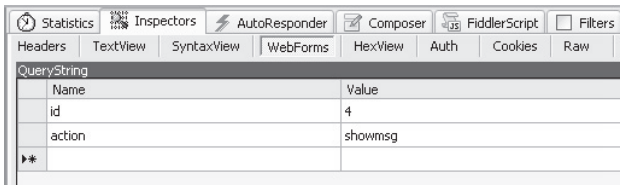


Następnie wracamy do przeglądarki i wskazujemy na liście dowolną wiadomość, po czym wciskamy przycisk *Sprawdź wiadomość* (używamy więc standardowego interfejsu strony WWW). Żądanie pobrania strony zostanie wtedy zatrzymane i uaktywni się okno Fiddlera. Transmitowane dane można dowolnie modyfikować. W zakładce *Headers* widać przesyłane nagłówki oraz pełny URL — możliwa jest jego zmiana (rysunek 1.15), a w zakładce *WebForms* można w wygodny sposób przeglądać i modyfikować przesyłane parametry (rysunek 1.16). Można więc zmodyfikować treść żądania, zmieniając np. wartość parametru `id`.

Rysunek 1.15. *Zatrzymanie żądania typu GET*

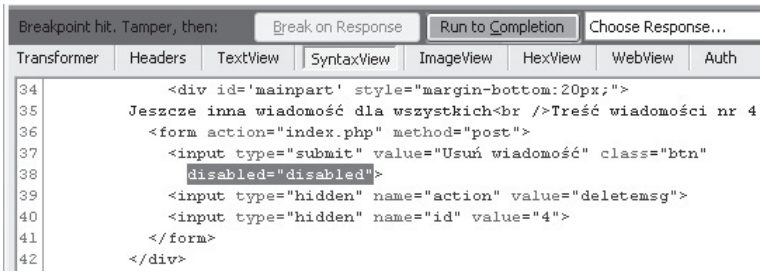


Po wykonaniu zmian można kontynuować żądanie i uzyskać odpowiedź — służy do tego przycisk *Run to Completion*, lub też automatycznie ustawić pułapkę na odpowiedzi serwera — służy do tego przycisk *Break on Response*. Użyjmy tej drugiej opcji.



Rysunek 1.16. *Widok parametrów przesyłanych w żądaniu*

Pozwoli to na wprowadzenie modyfikacji do strony wynikowej generowanej przez serwer. W oknie odpowiedzi pojawi się treść tej strony. Została przechwycona przez Fiddlera, a przeglądarka cały czas czeka na odpowiedź. Zanim jednak wypuścimy kod strony do przeglądarki, zmodyfikujmy kod źródłowy, odblokowując przycisk usuwania wiadomości. Wystarczy usunąć atrybut `disabled` z odpowiedniego znacznika `<input>` (rysunek 1.17). Po dokonaniu tej zmiany klikamy *Run to Completion*. Strona (już zmodyfikowana) powędruje do przeglądarki, gdzie będzie można sprawdzić, że przycisk, który miał być zablokowany, stał się aktywny.



Rysunek 1.17. *Modyfikacja odpowiedzi serwera*

Skorowidz

P

R

S

Ś

T

U

W

X

Z

Bezpieczeństwo złożonych, dynamicznych, interaktywnych współczesnych serwisów internetowych niejednokrotnie spędza sen z powiek projektującym je programistom. Niestety, bywa, że ich nocne koszmary zmieniają się w rzeczywistość i przygotowana przez nich strona WWW pada ofiarą złodziei (kradnących dane użytkowników) albo po prostu złośliwców, którzy czerpią przyjemność z niszczenia efektów cudzej pracy. Internet nie jest miejscem, w którym można pozwolić sobie na beztroskę i błędy — zwłaszcza jeśli serwis przechowuje wrażliwe dane (finansowe, zdrowotne, społeczne) albo gdy od jego działania zależy powodzenie dużego przedsięwzięcia biznesowego czy na przykład sprawne funkcjonowanie szpitala.

W tej książce znajdziesz omówienie dwunastu najbardziej typowych błędów programistycznych, wystawiających serwis internetowy na ataki hakerów. W każdym rozdziale prezentowana jest inna klasa błędów — na przykładach pochodzących z realnych, działających w sieci aplikacji — wraz ze sposobami zaradzenia tym błędom na etapie projektowania lub poprawiania strony WWW. Znajdziesz tu opis kwestii dotyczących kontroli dostępu do danych, wstrzykiwania kodu, przechowywania haseł użytkowników, właściwej autoryzacji, błędów transakcyjnych. Dowiesz się, jak zabezpieczać serwis przed atakami na sesję i na logowanie, atakami XSS czy Path Traversal. Jeśli tylko znasz PHP, MySQL, HTML i CSS w stopniu pozwalającym zaprojektować serwis internetowy, ta książka może uchronić Cię przed wieloma przykrymi niespodziankami...

- Kontrola dostępu do danych i funkcji
- SQL Injection
- Przechowywanie haseł użytkowników
- Ataki na logowanie
- Ataki typu XSS
- Dane z zewnętrznych źródeł
- Ataki CSRF i błędy transakcyjne
- Ataki Path Traversal
- Brak właściwej autoryzacji
- Dane u klienta
- Ataki na sesję
- Ładowanie plików na serwer

Zabezpiecz serwis i śpij spokojnie!

helion.pl
księgarnia internetowa
Nr katalogowy: 16693
Księgarnia internetowa:
http://helion.pl
Zamówienia telefoniczne:
0 801 339900
0 601 339900
Helion
Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci
Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl
Informatyka w najlepszym wydaniu
ISBN: 978-83-246-8131-0
9 788324 681310
Cena: 39,90 zł