

Online forms - security, validation and functionality

Andrzej Barczak, Dariusz Zacharczuk

Institute of Computer Science, University of Podlasie,
St. 3 Maja 54, 08-110 Siedlce, Poland

Abstract: The article discusses the problem of web forms security, and how the verification of data entry.

Key words: html, web forms

1 Introduction

Forms encounter at every step on the Internet. From the simplest to enable typing an email and subscribe to a newsletter, a very complex forms such as multipages orders, where we give contact details, etc. As a users we never wonder on what safeguards - if any - have the field against intentional or accidental introduction of erroneous data. It is look differently from behind. A good programmer should think not only about the functionality of created forms, but also about its security. After all, the contents of the fields usually consist of a query to the database and this may pose a potential danger.

The principle of the forms is simple. Having a code:

```
<form action="zapisz.php" method="post">  
    <input type="text" name="email" />  
    <input type="submit" />  
</form>
```

Our form will send data to a file `zapisz.php` using method `post`. Let's say the destination file execute SQL query content:

```
„INSERT INTO emaile (email) VALUES ('$_POST[email])“;
```


and after save, return to the page with the form:

```
header("HTTP/1.0 302 Redirect");  
header("location: index.php");
```

Is something wrong can stand up to that (how trivial) case? Whether such a form needs protection? Are we able to increase its functionality in itself, since it barely has issued with one? Yes, yes and yes. We'll explain later.

2 Characteristics of 'user-friendly' forms

Consider what you might expect, use our form, but Let us reject the wishes about interface:

- small number of fields to fill,
- precise information about what happens after when you click (eg. such as icon: ) ,
- in case of error did not have to reenter all that data.

Number of fields depends on the task that form realize and usually it can not be very minimized. However, you can divide the long form to several smaller and fill it in several steps. You willingly enters data in two or three steps every time, seeing a few fields than a dozen on a single page that you need to constantly scroll to get to the next position.

If your form contains only one field set a clear indication next to icon to a confirmatory entry example: "Click to save your data", otherwise you just confuse and scare the user. Of course, if the matter concerns the form discussed in the introduction, place icons like above should be enough :-).

However, there is nothing irritates internet user - such as the need to re-enter data, which laboriously introduced in the last 10 minutes, pointing his finger on the keyboard one letter after another. Even if contain only two or three fields, we will ensure that the errors do not cause the reset form.

These three simple rules should always be taken into account when designing web pages, so that our forms are user-friendly.

In order to discuss all aspects, we will assume that we have to create a form, which will contain a variety of fields. In addition, divide it into three pages. The basic version will look like this:

```
<!-- form - krok1.php -->
<form action="krok2.php" method="post" >
  <select name="wybor">
    <option value="1">jeden</option>
    <option value="2">dwa</option>
    <option value="3">trzy</option>
  </select>
  <input type="submit" value="go to krok2" />
</form>

<!-- form - krok2.php -->
<form action="krok2.php" method="post" >
  <input type="text" name="email" />
```

```

        <input type="submit" value="go to krok3" />
</form>

<!-- form - krok3.php -->
<form action="zapisz.php" method="post" >
    <input type="checkbox" name="zgoda"
value="tak" />
    <textarea name="opis"></textarea>
    <input type="submit" value="save all" />
</form>

```

The first thing we will be the third - the most important point from the previous chapter, remembering data entered by the user, in case of error or other circumstances. There are several methods to such action:

1. write to the database each time after sending the form of a step,
 - [+] Saved data may remain on a very long time - we have full control over time,
 - [-] save a record each time it sent by form,
 - [-] Need to check (in step 1) and possibly load data from database before displaying any form,
 - [-] The mechanism is not self-sufficient, for example, use the session, cookies to remember a unique identifier under which the data are stored in the database,
 - [-] A longer time taken to generate the page,
 - [-] We need to develop mechanisms for the treatment of temporary data base that will remain after the forms are not complete.
2. session, and remember the fields values into sessions variables,
 - [+] Short time taken to generate the page,
 - [/] We have control over time, however, limited (see the following subitem),
 - [-] Session expires after a few/several minutes, depending on server settings (usually we do not make any impact), if something turns the user's attention for a few moments, after the return all of data (regardless of which step we are) may be lost,
 - [-] Browser must accept cookies.
3. transmission of data in a form - hidden fields
 - [+] Does not write/read from the database - so a shorter time taken to generate the site,
 - [+] We have full control over the keeping of the data (for example, we can put timestamp in a hidden field),
 - [+] Does not need additional mechanisms to delete temporary data,
 - [+] Do not need cookies,
 - [-] That does not work out that we found an ideal method need to mention the two small drawbacks, the first: the data transmission method `post` in the refresh time pop out message like: ".. to see the page your browser must resubmit the data ..." and `get` makes the url can grow very,

[-] All data will remain confidential and we must reckon with the fact that having the right tools you can even modify the value of `hidden` fields eg using Web Developer plugin for Firefox.

Taking into account all pros and cons, in most cases the best choice would be the third option and it will also continue to analyze. There is nothing but an obstacle to combine several methods of exploiting their advantages. Transmission of data in a form can be realized by `post` or `get`. Table 2.1 presents the basic characteristics of both methods.

Table 2.1 POST vs. GET

Method POST	Method GET
Does not affect the url address so you can say that it is more elegant.	Attaches to the url string `key=value` for each parameter in the form. In the case of a large number of fields url address can reach hundreds of characters long. The advantage is that you can keep the address, which (because it contains string data) automatically complete the form.
Data is transmitted implicitly.	Field names and values are explicitly transmitted in the URL.
Complicated redirects structure. The POST method is available only with the form. Therefore, if necessary, redirect the POST method, we need to generate an intermediary page. It will contain a form with hidden fields, which we need and will be automatically sent using JavaScript such as: <pre><script> document.forms.form_js.submit(); </script></pre> <p>In the absence of scripting we have to make sure there is a submit button that the user will need to click.</p>	Very simple design redirects, links, etc. - just add to the address of the pair `key=value`. The GET method can be used in forms, and links directly to the PHP script code example: <pre>header("location: index.php?klucz=wartość");</pre>
Last but perhaps most importantly - this method is more secure. For example, if the form you enter your personal password, secret data transfer protects them (at least in the basic degree) before the eyes of third parties. Remember that POST calls record in the history of the browser and can be re-called - and thus to read the data set using appropriate tools.	Taking on the form field type `password`, which is used to hide input password (such as when you register a new user) password that become completely visible to outsiders people, as it is displayed directly in the URL. Not to mention the fact, that such a link with all the information that we gave up recorded in history, and unlike POST - do not need any tools to read it.

Selecting one of the methods depends on various factors. We will use both simultaneously. We abandon the separate files `krok1.php`, `krok2.php`, `krok3.php`, in order to distinguish step, we use the GET method (see the argument of `action`):

```
<form action="krok.php?krok=2" method="post" >
```

The data form will be sent implicitly and remembering the value realize of simple instructions:

```
<input type="text" name="email"
      value="<?php echo $_POST[email] ?>" />
```

In addition, in order to increase security we use session variables. At the time of first entry in `krok.php` regardless of the value of the parameter `step` we start the session and create a timestamp:

```
session_start();
$_SESSION['start'] = time();
```

Timestamp use to overcome the disadvantages of the POST method. Imagine that fill in a form and after a few minutes, another person uses the same computer. If you have not cleaned the history, that person (even if quite by accident) can re-send the data to form, which will appear them on the page (as ensured the opportunity to improve the functionality). Using the session, we can first check that has not expired and if so - to reset the form, eg like this:

```
session_start();
if ($_SESSION['start']==' ' ||
    time() - $_SESSION['start'] > 60 * 5)
    foreach ($_POST as $k => $v)
        $_POST[$k] = ' ';
```

Why such a solution:

- PHP code will not be duplicated in several files - making it easier to do a possible modification,
- thanks to the variable `step` in the URL will continue to have a direct link to the form,
- using advantages of session form will stay safer - after more than 5 minutes of inactivity data sent back by the browser does not display the in form, so access to them will be severely hampered.

After applying the above tips, form becomes more user friendly. Another action will be validation.

3 Validation of data

For each field we expect the value of a certain type such as asking about the zip code you want the person gave a string in the form of 5 digits and a hyphen: `08-110`. Validation of data helps us users:

- protects against the intentional introduction of erroneous data,
- helps to catch errors and random errors,
- allows you to filter the data before writing to the database.

The validation process may take place on the user side (using JavaScript) or server side (using PHP function). The best solution is to .. use both ways. With the javascript does not relieve the server side, does not need to overload the page when errors were detected. But we can not trust this method on 100% because support javyscript can be simply turn off. Thus, mandatory check the data just before writing them into the database.

There are ways to tell user about the need to integrate javyscript example:

```
<div id="js">
    Site to work properly requires JavaScript!
</div>
<script type="text/javascript">
    document.getElementById('js').style.display
="none";
</script>
```

This code will be visible on the site only when the service is disabled javyscript. If you want to be more persuasive and somehow force them to enable this option, we can use the following code:

```
<style> #form_js { display:none; } </style>
<div id="js">Enable JavaScript!</div>
<form      action="krok.php?krok=2"      method="post"
id="form_js">
    ...
<script type="text/javascript">
    document.getElementById('js').style.display
="none";
    document.getElementById('form_js').style.display
="block";
</script>
```

In this case, without javyscript form in general does not appear, and instead will could see the words "Enable JavaScript". When the 'power of persuasion' have convinced user to use javascript, we can begin to check the data entered by him. For this purpose, use the `onsubmit` event:

```
<form      action="krok.php?krok=2"      method="post"
id="form_js"
    onsubmit="return spr_form()">
```

This construction implies that after the approval of the form function is called `spr_form ()`, which will carry out validation of the fields and if everything is in order returns `true`. Otherwise, we will get `false` and not execute the form - the data will not be forwarded and you will have to correct the errors found.

For data validation (both using javascript and PHP) is best to use regular expressions. Table 3.1 shows examples of functions that verify the correctness of the given parameter values.

Table 3.1 Features that use regular expressions in order to verify the figures provided

Language	Description/code
	Verify your email address:
JS	<pre>function czyEmail(e) { if (e.match(/^([0-9a-z_.-]+@([0-9a-z-]+\.)+[a-z]{2,6})\$/)) return true; else return false; }</pre>
PHP	<pre>function czy_email (\$e) { return (preg_match("/^[a-zA-Z0-9_\\-]+@[a-zA-Z0-9\\-]+\\.([a-zA-Z0-9\\-\\.]+".\$e)>0); } }</pre>
	Verify zip code:
JS	<pre>function czyKod(t) { if (t.match(/^([0-9]{2}-[0-9]{3})\$/)) return true; else return false; }</pre>
PHP	<pre>function czy_kod (\$e) { return (preg_match("/^[d\\d-\\d\\d\\d\$/", \$e)>0); }</pre>
	Verify PESEL number:
JS	<pre>function check_pesel(PESEL) { var factor = new Array(1,3,7,9,1,3,7,9,1,3); s = 0; for (i=0;i<=9;i++) s += PESEL.charAt(i)*factor[i]; eleven = (10-s%10)%10; return (eleven==PESEL.charAt(10)); }</pre>
PHP	<pre>function czy_PESEL(\$str) { if (!preg_match('/^[0-9]{11}\$/', \$str)) return false; //tablica z wagami \$arrSteps = array(1, 3, 7, 9, 1, 3, 7, 9, 1, 3); \$intSum = 0; for (\$i = 0; \$i < 10; \$i++) { \$intSum += \$arrSteps[\$i] * \$str[\$i]; } \$int = 10 - \$intSum % 10; \$intControlNr = (\$int == 10)?0:\$int; if (\$intControlNr == \$str[10]) return true; return false; }</pre>

	Validation of a telephone number. Sometimes we have to give the user more freedom. The telephone number is a good example. You can make a webuser to provide the exact number of the form: +48 (25) 123 456 78, but whether would be the terrible fault if someone type it as: 25-12345678? Probably not. In such cases we just restrict the verification characters used and the length of a string - which is more than enough.
JS	<pre>function czyTel(t) { if (t.match(/^[\\(\\)0-9-]+\$/)) return true; else return false; }</pre>
PHP	<pre>function czy_telefon (\$e) { return (preg_match("/^[\\d\\(\\)\\+\\-]{9,20}\$/", \$e)>0); }</pre>

While identifying errors in the form it is good not stop at the first encounter position. Very annoying thing is when you improve the indicated error and soon learns the next, etc. So show at once which errors occurred and where. Denote the fields that should be corrected and set the cursor to the first incorrectly completed the field.

According to the second point section 2 let's also do the proper information when you click the `submit`. Deactivate button that could not be click again (in case of nerve users, or a slow internet connection) and bring up some text in the style of "The form is sending", it was known that "what happens".

4 Security

What are we to understand about the concept of security in the forms? There are two issues to which we must draw attention but both have a common denominator: dangerous characters. Referred to here is the ` ` and `"` - quotation marks and apostrophe, respectively. At first glance, do not look too dangerous but they can complicate life well, the html code or content of the database! Small example, we have the form:

```
<form action="zapisz.php" method="post" id="form_js">
  <input type="text" name="login"
    value="<?php echo $_POST['login']; ?>" />
  <input type="password" name="haslo" />
</form>
```

and the `zapisz.php` with the sql query:

```
„SELECT * FROM tabela WHERE
login = '$_POST[login]' AND
haslo = '". md5($_POST['haslo']) .'”;
```

Imagine now the two situations:

1. User complete box login value: `moj`login` and password: `abc`.
2. User gives a login: `my`username` and password `forgot`.

In the first case, after sending data to php file, and substituting them into a SQL query we get:

```
„SELECT * FROM tabela WHERE  
login = 'moj'login' AND  
haslo = '". md5('nie znam') .'”;
```

Query, of course, will fail because of the string: 'moj'login'. Such termination is fairly mild, imagine a situation where the query is performed to deleting records and someone enters for fun the string: ` 'OR 1`. The consequences can be much more serious.

The second case. User has entered incorrect data and login could not be done. However, our userfriendly form, which is obviously remembers the value of the username and complements them automatically, so surfer does not have to re-enter it. HTML code now look like this:

```
<input type="text" name="login" value="moj"login" />
```

Browser displays the value of `moj`. So we have a double problem: wrong value for the login field and an error in your HTML code. Such cases could be multiplied and the solution is trivial. In the case of PHP and SQL just add the escape character to ` ` or ` ``, depending on what character you are using the string or use a ready-made function `mysql_escape_string`.

In the case of HTML is sufficient to replace the apostrophe to the `'` and quote to `"`. Sometimes you may also need to convert `<` and `>` respectively `<` and `>`.

After all of the aforementioned endeavors amount of junk e-mail records in our database should be sought to zero.

5 Summary

At the end we have to remember: how far advanced our security and validation was not, if the surfer wishes to deceive us – he cheat and give false data in spite of everything. All the methods discussed above, increased functionality and safety facilitating our life, help to avoid simple mistakes and "attacks" frustrated Internet. Nothing, however, do not give us a 100% guarantee.

There are many other methods, more efficient, eg by sending a verification email message with activation link or captcha technology, verification by SMS. However, the "stronger" method for the greater reluctance of person completing form.

References

1. “Tworzenie stron WWW. Biblia. Wydanie III”, One Press, Październik 2009.
2. “Projektowanie stron WWW. Użyteczność w praktyce”, Wydawnictwo Helion, Grudzień 2008.

3. “Projektowanie nawigacji strony WWW. Optymalizacja funkcjonalności witryny”, Wydawnictwo Helion, 25 Czerwiec 2008.
4. Open-source free PHP CAPTCHA scripts: <http://www.phpcaptcha.org/>
5. Form Validation Using PHP: <http://www.php-mysql-tutorial.com/wikis/php-tutorial/form-validation-using-php.aspx>
6. JavaScript Form Validation: http://www.w3schools.com/jS/js_form_validation.asp