Institiúid Teicneolaíochta Chorcaí
Cork Institute of Technology

SOFT7008
Server Side Web Development

Handling User Input –Forms

Méabh O'Connor
meabh.oconnor@cit.ie

PHP Associative Arrays

By default – PHP creates numerically indexed arrays with a start index of 0.

PHP also facilitates **associative arrays** in which elements are referred to with an *alphanumeric key* instead of an index number.

- **e.g.** an associative array containing a company's payroll data could use each employee's surname instead of an index number to refer to elements in the array.

```
$PayRoll["Looby"] = 1000;
$PayRoll["Shaw"] = 2000;
$PayRoll["Cummins"] = 3000;
```


PHP Associative Arrays

- With associative arrays, you can specify an element's key by using the array operator (=>)
  - The syntax for declaring and initializing an associative array using the `array()` function is :
  - ```
    $array_name = array(key=>value, ...);
    ```
  - So to the $PayRoll[] array from the previous slide could be declared and initialized as follows:

```
$PayRoll = array( "Looby" => 1000,
                  "Shaw"  => 2000,
                  "Cummins" => 3000);
```


PHP Associative Arrays

- To refer to an element in an associative array – you place the element's key in single or double quotation marks in the array bracket.

```
echo "The CEO's daily rate is: ", $PayRoll["Cummins"];
```

! Associative arrays are best used when the array key provides additional information about the value of the array element. They are particularly useful when retrieving data from a database and storing the result set in an associative array where each database field name can become the associative key index.

PHP Autoglobals

- PHP includes various predefined arrays called **autoglobals** or **superglobals**.
  - Contain client, server and environment information that you can use in your scripts.
  - These arrays are *automatically* populated by PHP running on the web server.
  - Autoglobals are **associative arrays.**

PHP Autoglobals

Array	Description
$_COOKIE	An array of values passed to the current script as HTTP cookies
$_ENV	An array of environment information
$_FILES	An array of information about uploaded files
$_GET	An array of values from a form submitted with the "get" method
$_POST	An array of values from a form submitted with the "post" method
$_REQUEST	An array of all the elements in the $_COOKIE, $_GET, and $_POST arrays
$_SERVER	An array of information about the Web server that served the current script
$_SESSION	An array of session variables that are available to the current script
$GLOBALS	An array of references to all variables that are defined with global scope

Table 4-1 PHP autoglobals
```

PHP Autoglobals

- Always available – regardless of scope

e.g. display the SCRIPT_NAME element of the `$_SERVER` autoglobal

```
//contains the path and name of the current script
echo $_SERVER["SCRIPT_NAME"];
```


PHP $GLOBALS Autoglobal

- `$GLOBALS` is a PHP super global variable which is used to access global variables from anywhere in the PHP script *including from within functions or methods*.
- PHP stores *all global variables* in an array called `$GLOBALS[index]`. The *index* holds the name of the variable.

! Note that unlike the other PHP autoglobals – the `$GLOBALS` identifier does NOT contain an underscore _

PHP $GLOBAL Autoglobal

- The example below shows how the super global variable `$GLOBAL` can be used.

```
<?php
$x = 75;
$y = 25;

function addition()
{
    // add new global variable z to $GLOBAL array
    $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}
addition();
echo $z; // z now accessible outside function scope
//  not necessarily good programming practice!
?>
```


PHP $_SERVER Autoglobal

- `$_SERVER` is a PHP super global variable which holds information about headers, paths, and script locations.

```
<?php
// filename of the currently executing script
echo $_SERVER['PHP_SELF'];
echo "<br>";
// name of the host server such as www.w3schools.com
echo $_SERVER['SERVER_NAME'];
echo "<br>";
//  path of the current script
echo $_SERVER['SCRIPT_NAME'];
echo "<br>";
?>
```

A full list of `$_SERVER` key names is available at:
http://www.w3schools.com/php/php_superglobals.asp

PHP $_FILES, $ENV Autoglobals

- **`$_FILES`** is a superglobal that contains a list of items uploaded to the current script via the HTTP POST method.
- **`$_ENV`** is a superglobal that contains environment variables set for the operating system on the machine that hosts the web server.
  - Unlike `$_SERVER` which contains a predefined list of elements – the `$_ENV` super global elements change depending on the operating system and the machine's configuration.
  - The `phpinfo()` function used in Lab 1displays the elements of the `$_ENV` array and their values.

PHP $_SERVER, $ENV Autoglobals

Do not display the values stored in the `$_SERVER` and `$ENV` autoglobals on a web page as they contain important information that a hacker could use to identify weaknesses in the server.

PHP Handling User Input

- Two of the most common ways that PHP interfaces with the user are by:
  - accessing values from web forms that are submitted to a PHP script
  - handling events such as dynamically displaying pages when the user clicks a hyperlink

Building HTML Web Forms

- Processing data submitted from an HTML form on a web page is probably the most fundamental task performed by a PHP script.
- A web form is a standard HTML form with two required **attributes** in the opening `<form>` tag:
  - **action attribute:** identifies the script on the web server that will *process* the form data when it is submitted
  - **method attribute:** specifies **how** the form data will be **sent** to the processing script

Adding an `action` Attribute

- To nominate the PHP script that will "handle" the data submitted from the web form set the `action` attribute within the opening `form` tag.
- The **value** of the `action` attribute *identifies the program* on the web server that will process the form data

```
<form action="http://www.example.com/HandleFormInput.php">
```


Adding the `method` Attribute

- The `method` attribute defines *how* the form data is submitted to the server.
- The value of the `method` attribute must be either `"post"` or `"get"`
  - The `"post"` method **embeds** the form data within the HTTP request message
  - The `"get"` method **appends** the form data to the URL specified in the form's action attribute

Adding the `method` Attribute

- When a Web form is submitted using the `"post"` method, PHP **automatically creates and populates** a `$_POST` array
- When the `"get"` method is used, PHP **automatically creates and populates** a `$_GET` array
- The values sent by the browser client are then available to the appropriate PHP script on the web server

Adding the `method` Attribute

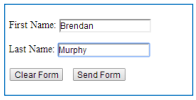
- When the user hits the **Submit** button – data in the form field(s) are sent to the web server as a *name/value* pair
  - The *name* portion of the *name/value* pair becomes the key of an element in the `$_POST` or `$_GET` array, depending on which method was used to submit the data
  - The *value* portion of the name/value pair is *populated by the data that the user enters* in the input field on the web form

Form name/value Pairs

- The POST method is used to submit this HTML form- PHP will populate the `$_POST` superglobal array with name/value pairs based on the form field names and corresponding data entered by the user.

```
<form name = "scholarship" action = "process_Scholarship.php" method = "post">
    <p>First Name: <input type="text" name="fName" /></p>
    <p>Last Name: <input type="text" name="lName" /></p>
    <p><input type="reset" value="Clear Form" />   
    <input type="submit" name="Submit" value="Send Form" />
</form>
```


Form name/value Pairs



Client side

field name	value
fname	Brendan
lname	Murphy

Server side

Form name/value Pairs

Field name	value
fname	Brendan
lname	Murphy

- If the submit method is by "`post`" the **`$_POST`** array will contain:

`$_POST['fname']`	Brendan
`$_POST['lname']`	Murphy

Form name/value Pairs

Field name	value
fname	Brendan
lname	Murphy

- If the submit method is by "`get`" the **`$_GET`** array will contain:

`$_GET['fname']`	Brendan
`$_GET['lname']`	Murphy

Adding the `method` Attribute

- When submitting data using the "`get`" method, *form data is appended to the URL specified by the* `action` *attribute*
- Name/value pairs appended to the URL are called **URL tokens**

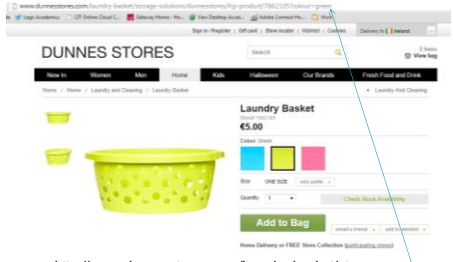
Submitting Data by '`get`' `method`

- **When data is sent from the browser to the web server using the `get` method :**
- the form data is separated from the URL by a question mark (?)
- the individual elements are separated by an ampersand (&)
- the element name is separated from the value by an equal sign (=).
- spaces in the *name* and *value* fields are encoded as plus signs (+)

Submitting Data by 'get' method

- all other characters except letters, numbers, hyphens (-), underscores (_) and periods (.) are encoded using a percent sign (%) followed by the two-digit hexadecimal representation of the character's ASCII value
- e.g. a space can be `%20`

Sample Submission of Data by 'get' method on Dunnes Stores Website



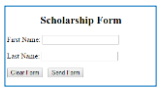
http://www.dunnesstores.com/laundry-basket/storage-solutions/dunnesstores/fcp-product/7862105**?colour=green**

Submitting Data by 'get' method

- Consider the **Scholarship Form** created in the lab on basic web form processing:

```
<form name = "scholarship" action = "process_Scholarship.php" method = "get">
<p>First Name: <input type="text" name="fName" /></p>
<p>Last Name: <input type="text" name="lName" /></p>
<p><input type="reset" value="Clear Form" />   <input type="submit" name="Submit" value="Send Form" />
</form>
```

- When the user enters data for this form and clicks `Submit` button labelled "Send Form" …



Submitting Data by 'get' method

- …The data is appended to the URL to build a ***URL query string*** as follows:

```
http://www.example.net/process_Scholarship.php?
fName=Brendan&lName=Murphy&Submit=Send+Form
```


Submitting Data by 'get' method

The target script `process_Scholarship.php` on the web server generates a personalised **Thank You** message based on the name data entered:

```
// value submitted from input field name="fName"
$firstName = $_GET['fName'];
// value submitted from input field name="lName"
$lastName =  $_GET['lName'];

 echo "Thank you for filling out the scholarship form,
                    ".$firstName." ".$lastName  . ".";
```


Security Weakness Alert

- The problem in the previous example stems from the fact that the code is generating web page content under the *control of the user*
- A malicious user could edit the contents of the query string in the address bar as below and hit Enter

```
http://www.example.net/process_Scholarship.php?
fName=<b>Brendan<\b>&lName=<i>Murphy</i>&Submit
=Send+Form
```

Security Weakness Alert

This will set the content of the Thank You message to:

```
echo "Thank you for filling out the scholarship
      form, <b>Brendan</b>  <i>Murphy</i>.";
```

The browser will parse the string **literally** – interpreting the <b> and <i> as HTML tags to display:

Thank you for filling out the scholarship form, **Brendan** *Murphy*.

This is an innocuous example but a devious user could include **sophisticated JavaScript code** that could – for example – steal a user's password.

Security Weakness Alert

- Need to modify the coding style to ensure that all data entered by the user is processed as **plain text** to be displayed on the web page rather than as **HTML** to be included in the web page code

⇒ Use the `htmlspecialchars()` built-in PHP function to convert special HTML characters like "<" and ">" into HTML character entities `&lt` and `&gt` which **prevents** them from being interpreted as HTML code by the browser.

```
$firstName =
  htmlspecialchars($_GET['fName'],ENT_QUOTES,'UTF-8');
$lastName
  =htmlspecialchars($_GET['lName'],ENT_QUOTES,'UTF-8');
 echo "Thank you for filling out the scholarship form,"
            .$firstName." ".$lastName  . ".";
```


Security Weakness Alert

- It is good programming practice to call the `htmlspecialchars()` function on **any** (by `get` or `post`)incoming *user generated data* to strengthen your PHP code against malicious attack.

Function Definition:

`htmlspecialchars(`*`string,flags,character-set,double_encode`*`)`

string	Required. Specifies the string to convert
flags	Optional. Specifies how to handle quotes, invalid encoding and the used document type.
character-set	Optional. A string that specifies which character-set to use.
double_encode	Optional. A boolean value that specifies whether to encode existing html entities or not.

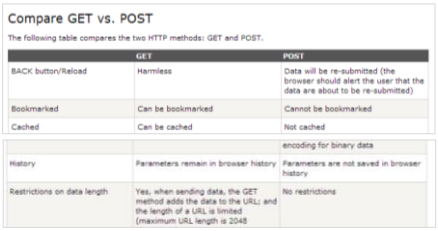
Submitting Data by 'get' method

- Advantages of the "get" method for submitting form data
  - Passed values are visible in the address bar of the browser – good for **debugging** purposes
  - Good for creating static links to a dynamic server process – *see later in slides.*

Submitting Data by 'get' method

- Limitations of the "get" method for submitting form data
  - The form values are appended to the URL in **plain text**, making a URL request **insecure**
    - Do NOT use this method to send sensitive data like PPS or credit card numbers to the server!!
  - Restricted in the number of characters that can be appended to a single variable up to 100 and the total max. length of the URL query string is 2048.
  - Can only send ASCII character data not binary data

Compare get vs post



Compare GET vs. POST

The following table compares the two HTTP methods: GET and POST.

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
		encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048	No restrictions

Compare get vs post

Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

Handling Special Characters

- Older versions of PHP (pre v5.3) contained a feature called **Magic Quotes** which **automatically** add a backslash character to any:
  - single quote,
  - double quote,
  - `NULL` character,
  - backslash

contained in form data that a user submits to a PHP script.

The Thank You message would become:

```
Thank you for filling out the scholarship form,
Meabh O\'Connor.
```


Handling Special Characters

- Magic quotes are enabled within the `php.ini` configuration file with the directives listed below:

Directive	Description
`magic_quotes_gpc`	Applies magic quotes to any user-submitted data
`magic_quotes_runtime`	Applies magic quotes to runtime-generated data, such as data received from a database
`magic_quotes_sybase`	Applies Sybase-style magic quotes, which escape special characters with a single quote (') instead of a backslash (\)

Table 4-2 Magic quote directives

Handling Special Characters

- By default – `magic_quotes_gpc` is the only magic quote directive enabled in the `php.ini` file.
- Call the `get_magic_quotes_gpc()` function to determine if magic quotes are enabled on the server hosting your script.
- Many PHP programmers have spent hours trying to determine why backslashes were being added to data their scripts received – only to discover that the culprit was a magic quote directive in the `php.ini` file!

Handling Special Characters

- The `addslashes()` function returns a string with backslashes in front of predefined characters. The predefined characters are:
  - single quote (')
  - double quote (")
  - backslash (\)
  - NULL
- *The* `stripslashes()` function removes backslashes added by the addslashes() function. If magic quotes is enabled, this is required before outputting a string with the `echo` statement.

Handling Special Characters

- If magic quotes are enabled and an input string **e.g** `O'Connor` is *escaped* (backslashes inserted before special characters) with a call to `addslashes()` function then the :
  - Magic quotes directive will add a back slash before the apostrophe to give –
    `O\'Connor`
  - Then a call to the `addslashes()` function will add a backslash before BOTH the backslash and the apostrophe to give:
    `O\\'Connor`

The existence of the `addslashes()` function is actually another reason why **magic quotes** are unpopular within the PHP developer community.

Handling Special Characters

- The best way to deal with **magic quotes** is to **detect** if it is enabled on your server and then **undo** the modifications it has made to submitted values.

⇒ Insert the following code snippet available on the www.php.net website:

```
if (get_magic_quotes_gpc())
{
  function stripslashes_deep($value)
  {
    $value = is_array($value) ?
        array_map('stripslashes_deep', $value) :
        stripslashes($value);

    return $value;
  }

  $_POST = array_map('stripslashes_deep', $_POST);
  $_GET = array_map('stripslashes_deep', $_GET);
  $_COOKIE = array_map('stripslashes_deep', $_COOKIE);
  $_REQUEST = array_map('stripslashes_deep', $_REQUEST);
}
```


Handling Special Characters

- Further reading on Magic Quotes at:

http://php.net/manual/en/security.magicquotes.php

Processing Form Data

- A **form handler** is a program or script that processes the information submitted from a web form

```
<form name = "scholarship" action = "process_Scholarship.php" method = "post">
```

- **A form handler usually performs the following:**
  - *Verifies* that the user entered the minimum amount of data to process the form
  - *Validates* form data
  - *Processes* the submitted data
  - *Returns* appropriate output as a web page

Handling Submitted Form Data

- It is necessary to **validate** web form data to ensure PHP can use the data
- The optimal way to ensure valid form data is *only allow the user to enter an acceptable response*
  - Good practice to set the `<input type="value">` on the HTML form to the appropriate **type** to limit data entered by user to correct format on browser side before it is submitted to web server.
- Examples of data validation include verifying that
  - the user did not leave any required fields blank
  - an e-mail address was entered in the correct format
  - the user did not exceed the word limit in a comment box

Determining if Form Variables Contain Values

- The first step in validation form data is to determine if fields have data entered in them.
- When form data is posted using the "`post`" or "`get`" method, all controls except unchecked radio buttons and checkboxes get sent to the server *even if they do not contain data*
- The `empty()` function is used to determine if a variable contains a value
- The `empty()` function returns:
  - `FALSE` if the variable being checked has a nonempty and nonzero value i.e. contains data
  - `TRUE` if the variable has an empty or zero value

Determining if Form Variables Contain Values

```
// check if the required name field is empty
if (empty($_POST["name"]))
    {$nameErr = "Name is required";}
else
    {$name = test_input($_POST["name"]);}
```

The empty() function returns a value of FALSE for a numeric value of 0. If you are validating a numeric field for which 0 is a valid entry – you must check for a value of 0 separately.

Validating Entered Data

- Validating form data refers to *verifying that the value entered in a field is appropriate for the data type that should have been entered*
- The best way to ensure valid form data is **first** to design the web form with controls (such as check boxes, radio buttons, and selection lists) *that only allow the user to select valid responses*
- However – this method only works if the user is confined to a predefined set of responses.

Validating Entered Data

- Unique information, such as user name, password, or e-mail must be validated to ensure that the entered values are acceptable for the type of data required.

Validating Numeric Data

- All data in a web form is string data and PHP automatically converts string data to numeric data if the string is in numeric format
  - The `is_*()` family of functions can be used to ensure that the user enters numeric values where necessary
  - The `is_numeric()` function is used to determine if a variable contains a number
  - Good practice to use the trim function before `is_numeric()`
  - Ensure values are within a required range

```
if(($data >= 1900) && ($data <= 2100))
```


Validating String Data

- Regular expression functions are some of the best tools for verifying that string data meets the strict formatting required for e-mail addresses, web page URLs, or date values
- New HTML5 form input elements like email, range and number will perform a significant amount of data validation on the client side.
- But not all browsers support all the new validation functionality of these input elements so for now you need to continue with server side validation.

Redisplaying the Web Form

- It is common but **poor programming practice** is to **stop** processing a form when an error is found and display the error to the user.
- The user corrects the error only to find another field in the form also has an error.
- For a complex form with multiple fields this can result in many frustrating attempts by the user to fill the form successfully.

Redisplaying the Web Form

- It is good programming practice to *record the error and continue processing* the form until all fields have been validated.
- This allows the program to display a complete list of all the errors found.
- Users can then correct all the errors at **one** time when the form is redisplayed.
- See upgraded version of `process_Scholarship.php` with validation.

Handling Multiple Errors

- You can use a global variable to count errors on the form during processing

```
$errorCount = 0;
```

- Create a function to validate input and increment the error counter each time an error is detected

```
++errorCount;
```


Basic Validation

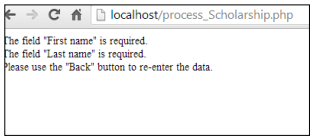
```
<?php
function displayRequired($fieldName) {
    echo "The field \"$fieldName\" is required.<br />\n";
}
function validateInput($data, $fieldName)
{   // track number of errors
    global $errorCount;
    if (empty($data)) {
        displayRequired($fieldName);
        ++$errorCount;
        $retval = "";
    } else { // Only clean up the input if it isn't empty
        $retval = trim($data);
        $retval = htmlspecialchars($retval, ENT_QUOTES);
    }
    return($retval);
}
```


Basic Validation

```
$errorCount = 0;
$firstName = validateInput($_POST['fName'], "First name");
$lastName = validateInput($_POST['lName'], "Last name");

if ($errorCount>0)
    echo "Please use the \"Back\" button to re-enter the
        data.<br />\n";
else
{
    echo "Thank you for filling out the scholarship form,
        ".$firstName." ".$lastName  . ".";
}
?>
```


Basic Validation



Sticky Forms

- A **sticky form** is used to redisplay the form with the *controls set to the values the user entered the last time the form was submitted*
- The following syntax illustrates how to use the **value** attribute to display previously submitted values in a sticky form:

```
<p>First Name: <input type="text"
 name="fName" value="<?php echo $firstName;
 ?>" /></p>
```


Sticky Forms

- A **sticky form** is used to redisplay the form with the *controls set to the values the user entered the last time the form was submitted*
- The following syntax illustrates how to use the **value** attribute to display previously submitted values in a sticky form:

```
<p>First Name: <input type="text"
 name="fName" value="<?php echo $firstName;
 ?>" /></p>
```

Advanced Escaping from HTML

- This code example uses **advanced escaping from HTML** to embed large portions of HTML – within the PHP script.
- With advance escaping – you close one PHP script section – insert some HTML elements and then open another PHP script section to continue the script.
- Any HTML code between the 2 script sections is considered output – as it would have been using an echo statement.

Advanced Escaping from HTML

- If the closing tag for the first PHP script section is within a function or control block for a conditional structure – the *HTML will only be displayed* if the function is called or the conditional or control block is executed.

Sticky Forms with Advanced Escaping

```
function redisplayForm($firstName, $lastName)
{
?>
<!--Sticky Form Defintion-->
<h2 style = "text-align:center">Scholarship Form</h2>
<form name="scholarship" action="process_Scholarship.php"
          method = "post">
    <p>First Name: <input type="text" name="fName" value="<?php
                    echo $firstName; ?>" /></p>
    <p>Last Name: <input type="text" name="lName" value="<?php
                    echo $lastName; ?>" /></p>
    <p><input type="reset" value="Clear Form" /> 
                     <input type="submit" name="Submit" value="Send Form" />
</form>
<?php
}
```


Sticky Forms

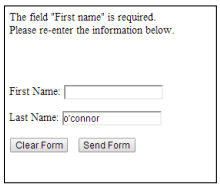
- Call the function `redisplayForm()` if errors have been detected *but fill in fields with previously entered valid data*:

```
$errorCount = 0;
$firstName = validateInput($_POST['fName'], "First name");
$lastName = validateInput($_POST['lName'], "Last name");

if ($errorCount>0)
{
    echo "Please re-enter the information below.<br />\n";
    redisplayForm($firstName, $lastName);
}
else
{
    echo "Thank you for filling out the scholarship form,
      ".$firstName." ".$lastName  . ".";
}
?>
```


Sticky Forms

- When the form is redisplayed – the valid data the user entered is automatically filled in :



Using the Submitted Data

- Exactly how the validated data is used varies depending on the purpose of the form.

e.g:

- It may be written to a database, initiate a file download to the user or it may be used to generate an email message and display a confirmation message to the user.

Emailing the Web Form

- The `mail()` function is used to send an e-mail message containing form data in PHP
- The basic syntax for this function is

  `mail(recipient(s), subject, message)`
- To use the PHP `mail()` function, PHP requires an installed and working email system. The program to be used is defined by the **configuration settings** in the `php.ini` file.

Emailing the Web Form

- The value assigned to the `recipient(s)` argument is a string of one or more email addresses in the standard format for an **Address Specifier** defined by the Internet Message Format documentation :
  - Plain e-mail address: `jdoe@example.net`
  - Recipients name and e-mail address: `Mary Smith <mary.smith@example.com>`

Emailing the Web Form

- The `subject` argument of the `mail()` function must include *only plain text* with no HTML tags
- The `message` argument of the `mail()` function is a text string that must also be in plain text
- A fourth, optional `additional_headers` argument can include headers that are standard in most e-mail editors - `From`, `Cc`, `Bcc` and `Date`.

Emailing the Web Form

With the `additional_headers` argument

- Each header must be on its own line (carriage return)
- Each line must start with the header name, followed by a colon, a space, and the value of the header element

```
Date: Fri, 03 Apr 2009 16:05:50 -0400
From: Linda M. Jones linda@jones.example.com
CC: Mary R. Jones <mary@jones.example.com>
```

- vulnerable to **email injection attack**!

- A successful e-mail message returns a value of `TRUE`

Creating an All-in-One Form

- The previous code example implements a separate document for the web form (`Scholarship.html`) and a separate form handler (`process_Scholarship.php`)
- This is called a **two-part form** where there is one page that displays the form and one page that processes the form data
- For **simple** forms that require only minimal processing, it's often easier to use an **All-in-One form**—a single script used display a Web form **and** process its data

Validating an All-in-One Form

- Consider the simple program `NumberForm.php` which will display the square of a number entered.
- When the user clicks the submit button - the script submits the form data to the **current** script.
- It uses a **conditional** to *determine if the form has already been submitted or if it is being viewed for the first time.*
  - The first conditional determines if the data has been submitted and needs to be validated.
  - The second conditional determines if the form needs to be redisplayed - either because of a validation error or because the user is opening the page for the first time.

Redisplaying the Web Form

- If the submitted data did not pass all validation checks or no data has been entered, the All-in-One form will display the web form, for the user to enter data for the first time or re-enter data that did not pass validation

```
if (isset ($_POST['Submit']))
{
// Process the data
}
else
{
// Display the Web form
}
```


isset() Function

- The `isset()` function can be used to determine if the `Submit` button on the form has been pressed.

```
bool isset ( mixed $var [, mixed $... ] )
```

- This function returns TRUE if the variable(s) passed as an argument has been initialized and is not NULL.
- In this case – the argument of the `isset()` function is the name assigned to the **Submit** button in the Web form (`name='Submit'`)
- So – a variable that contains an empty string (""), 0, NULL or FALSE will return TRUE from `isset()` as it has been set.

Validating an All-in-One Form

- Declare and initialize a flag to TRUE to indicate that the form should be displayed.

```
$DisplayForm = TRUE;// flag
```

- Call the `isset()` function to determine if the `$_POST('Submit')` variable has been **set**

```
if (isset($_POST('Submit'))
{
  // Validate the data
      //clear display flag if data valid
}
```


Validating an All-in-One Form

- if the `displayForm` flag is set to TRUE
  - -> display the form
- otherwise
  - -> the data is valid so display a Thank You message
  - -> display a hypertext link start processing again

Displaying Dynamic Content Based on a URL Token

- By passing URL tokens to a PHP script, many different types of information can be displayed from the same script
- Use the HTML `<a>` tag to create a *hyperlink* to another file.

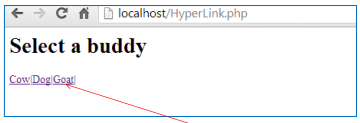
HTML <a> Tag

The `<a>` tag defines a *hyperlink*, which is used to setup a link from one page to another.

- The most important **attribute** of the `<a>` element is the `href` attribute, *which indicates the link's destination.*
- The HTML syntax to create a hyperlink to www.w3schools.com which displays the text "Visit W3Schools.com!" is:

```
<a href="http://www.w3schools.com">
                Visit W3Schools.com!</a>
```

Displaying Dynamic Content Based on a URL Token



- When the user selects a hyperlink – the `id` of the selected item is sent in the URL query string to the `HyperLink.php` file

```
echo '<a href="HyperLink.php?id=2">Dog</a>|';
```


Displaying Dynamic Content Based on a URL Token

```
<?php
$id = 0;
if (isset ($_GET['id']))
{
    $id = $_GET['id'];

    // display a different message based on hyperlink selected
    switch ($id)
    {
        case 1: echo 'Cow selected. <hr>'; break;
        case 2: echo 'Dog selected. <hr> '; break;
        case 3: echo 'Goat selected. <hr>'; break;
    }

}
echo '<h1>Select a buddy</h1>';
echo '<p> <a href="HyperLink.php?id=1">Cow</a>|';
echo '<a href="HyperLink.php?id=2">Dog</a>|';
echo '<a href="HyperLink.php?id=3">Goat</a></p>';
?>
```

user selection passed in in a query string – so data in $_GET[]

displays hyperlinked options to user

Displaying Dynamic Content Based on a URL Token

- `id=2` set in URL query string

Organizing Files in Large PHP Projects

- As the code base for your PHP projects increases it is important to **divide** up the code into files along common functionality.
- To assist in optimizing code organization and reuse PHP provides language constructs to include code files within each other.
- The `include`, `require`, `include_once` and `require_once` statements allow you to *insert the content of an external file* – called an include file – *in your PHP scripts*.

Organizing Files in Large PHP Projects

- The difference between the 2 statements is that the `include` statement only generates a **warning** if the include file cannot be found
- **BUT**
- The `require` statement **halts** the processing of the web page and displays an error message if the include file cannot be found

Organizing Files in Large PHP Projects

- The `include_once` and `require_once` statements are similar to the `include` and `require` statements except that they assure that the *external file is added to the script only once* – this helps avoid conflicts with variable values or function names that might occur if the file is included multiple times.
- The PHP script engine starts fresh for each include file.
  - if you use PHP code in the include file – it must be contained within a PHP script section.
  - the 4 basic HTML tags <DOCTYPE html>, <head>, <title> and <body> *should only be in the containing PHP file.*

Organizing Files in Large PHP Projects

- Good practice to save the include file with a **prefix** of `inc_` to identify it as an include file.
- An extension of `.php` is still required so that the file will be processed by the PHP engine.
- The include family of statements supports relative and absolute path notation.
  - You can include a file from the parent folder using the `"../"` notation.
  - e.g. `include("../inc_CommonHeader.php");`

  Or:
  - e.g. `include("Includes/inc_CommonHeader.php");`

    to access files in an Includes sub directory.

Using Web Templates

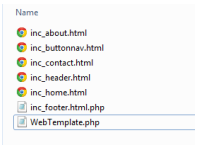
- A common use of the include and require statements is to *display common header and footer content at the top and bottom of every page in your web site.*
  - Instead of copying and pasting the header and footer code into each individual page –
    - put your header content in one include file
    - put your footer content in another include file
    - on each page that you want the header and footer content to appear – *simply include the file(s) with either the include or require statement.*
  - *Only 1 version of header and/or footer code needs to be created/maintained*

Using Web Templates

- Web pages can be divided into separate sections such as:
  - Header
  - Button Navigation
  - Dynamic Content
  - Footer
- The contents of the individual sections are populated using include files

Simple Project File Structure

- A simple project file structure using include files and templates:



Simple Project File Structure

WebTemplate.php

```
include ("inc_header.html")
include ("inc_buttonnav.html")
conditionally include
        include('inc_about.html')
        include('inc_contact.html')
        include('inc_home.html')
include('inc_home.html')
include ("inc_footer.html.php")
```


Simple Project File Structure

- In this example – the file `WebTemplate.php` includes code files to display constant
  - header
  - button navigation
  - footer
- It also conditionally **includes** *different templates under different conditions – dependent on user selection*
- This type of PHP script is called a **controller** as it controls or *drives* the core execution of the PHP program.

Simple Project File Structure

- The controller code file is often named **`index.php`**
- By default – many web servers will automatically execute the file called `index.php` when the folder is browsed.
- So we could rename the file `WebTemplate.php` as `index.php`
- **Remember** to replace references to the `WebTemplate.php` file in the `inc_buttonnav.html` with `index.php`

```
<form action="WebTemplate.php" method="get">
```


Simple Project File Structure

- Now when we navigate to :

`http://localhost/ProjectStructure/`

…the controller code in `index.php` will *automatically* execute

Simple Project File Structure

- `index.php` code:

```
<!DOCTYPE html>
<head>
<title>Web Template</title>
<meta charset="utf-8" />
</head>
<body>
<?php include ("inc_header.html"); ?>
    <div style = "width:20%; text-align:center; float:left">
<?php include ("inc_buttonnav.html"); ?>
</div>
```


Simple Project File Structure

- `index.php` code:

```
<!-- Start of Dynamic Content section -->
<?php
if (isset($_GET['content']))
{
    switch ($_GET['content'])
    {
        case 'About Me':
            include('inc_about.html');
            break;
        case 'Contact Me':
            include('inc_contact.html');
            break;
        case 'Home': // A value of 'Home' means to
                     // display the default page
        default:
            include('inc_home.html');
            break;
    }
```


Simple Project File Structure

- `index.php` code:

```
}
else // No button has been selected
    include('inc_home.html');
?>
<!-- End of Dynamic Content section -->
<?php include ("inc_footer.html.php"); ?>

</body>
</html>
```


Using Web Templates

Files that contain mostly HTML with only very **small snippets of PHP code** that *insert dynamically generated values* into an otherwise static HTML page can be further distinguished by saving them with the `.html.php` extension.

- Using PHP templates like this enables you to hand your templates over to HTML web page designers without worrying what they might do to the PHP code.
- It also lets you focus on writing your PHP code without being distracted by the surrounding HTML code.