

PHP: Interazione con l'ambiente lato server

Interazione con le richieste

- ❑ Gli script PHP vengono attivati quando arriva una richiesta diretta alla risorsa corrispondente (file .php)
- ❑ Si deve poter interagire con
 - ❖ Ambiente (Environment)
 - ❖ GET
 - ❖ POST
 - ❖ Cookie
 - ❖ Server(cosiddetto EGPCS)

Variabili predefinite

□ Il server crea 6 array associativi per le informazioni EGPCS

- ❖ `$_ENV` valori delle variabili di ambiente
- ❖ `$_GET` valori passati da un modulo
- ❖ `$_POST` valori passati da un modulo
- ❖ `$_COOKIE` eventuali cookies nella richiesta
- ❖ `$_SERVER` informazioni sul server
- ❖ `$_FILE` informazioni su eventuali files trasmessi nella richiesta

□ Se si vuole sapere la struttura di queste variabili si può usare

```
var_dump(variable) o print_r(variable)
```

Stampa degli elementi

```
<h2 style="text-align:center; color:red;">
Stampa Variabili $_SERVER</h2>
<table border cellpadding=2>
<tr> <th> Variabile </th> <th> Valore </th>
  </tr>
<?php
  foreach($_SERVER as $indice =>$valore) {
    echo "<tr><td>$indice</td>
    <td>$valore</td></tr>\n";
  }
?>
</table>
```

[table.php](#)

Esercizio

- ❑ Modificare l'istruzione dell'esempio visto in precedenza sugli array cosicché che prenda indice e nome da un form HTML:

```
$indice=2; $nome="homer";
```

diventa:

```
$indice=$_GET['indice'];
```

```
$nome=$_GET['nome'];
```

- ❑ Introduciamo un file HTML con un form per trasmettere le informazioni

`exercise2.php`

`exercise2.html`

```

<?php
$passwords=array(
    'bart' => 'calzino',
    'homer' => 'birra',
    'lisa' => 'nobel',
    'marge' => 'caspiterina',
    'maggie' => '' );
$numprimi[0] = 1;$numprimi[1] = 2;$numprimi[2] = 3;
?>
<html>
    <head><title>Esempio 2</title></head>
    <body>
        <h1>Array</h1>
        <?php
            $indice=$_GET['indice']; $nome=$_GET['nome'];
            echo "<p>Password di $nome:
            $passwords[$nome]</p>";
            echo "<p>Il $indice numero primo:
                $numprimi[$indice]</p>";
        ?>
    </body>
</html>

```

Risposta a GET e POST

- ❑ I dati di un form possono essere inviati sia tramite GET sia tramite POST
- ❑ Come conoscere quale metodo è stato usato?
 - ❖ Controllare entrambe le variabili
 - ❖ Usare la variabile per conoscere il metodo:
`$_SERVER['REQUEST_METHOD']`
 - ❖ Usare l'array `$_REQUEST` che include i valori di `$_GET`, `$_POST`, `$_COOKIE`

Revisione esercizio

```
if (isset($_GET['index']))
    $index=$_GET['index'];
else
    $index=$_POST['index'];

switch($_SERVER['REQUEST_METHOD']) {
    case 'GET': $index=$_GET['index'];
                break;
    case 'POST': $index=$_POST['index'];
                break;
}

$index = $_REQUEST['index'];
// Meglio:
if (isset($_REQUEST['index']))
    $index = $_REQUEST['index'];
```

[exercise2rev1.php](#)

[exercise2post.html](#)

[exercise2rev2.php](#)

[exercise2post2.html](#)

Esempio con selezione multipla

Caso checkbox multiple o <select>:

```
<form action="..." method="...">
```

```
Red<input type="checkbox" name="color[]" id="color" value="red">
```

```
Green<input type="checkbox" name="color[]" id="color" value="green">
```

```
Blue<input type="checkbox" name="color[]" id="color" value="blue">
```

```
<input type="submit" value="submit">
```

```
</form>
```

`$_REQUEST["color"]` è un array
che contiene:

Nel caso di selezione di rosso e blu:

```
color[0]='red'
```

```
color[1]='blue'
```

Nel caso di selezione di verde:

```
color[0]='green'
```

Nel caso di nessuna selezione:

non c'è la chiave «color» in `$_REQUEST`

```
<select multiple name="color[]">  
<option value="red">Red</option>  
<option value="green">Green</option>  
<option value="blue">Blue</option>  
</select>
```

Funzione header

- ❑ Permette di scrivere una intestazione nella risposta
- ❑ **Attenzione!!** deve precedere ogni altro tipo di output

`<html>` ← **output precedenti**
`<body>` ← **output precedenti**
`<?php header (.....) ;` **errore!!**

❑ Sintassi

```
header( $string [, bool $replace = true  
        [,int $http_response_code ]] )
```


Accesso protetto

- ❑ Si possono utilizzare meccanismi standard dei server/browser (pop-up di una finestra sul browser)
 - ❖ Basic HTTP authentication
 - ❖ Digest HTTP authentication
- ❑ Per leggere i valori inseriti si usano le variabili
 - ❖ `$_SERVER['PHP_AUTH_USER']`
 - ❖ `$_SERVER['PHP_AUTH_PW']`
 - ❖ `$_SERVER['AUTH_TYPE']`
- ❑ Si può anche costruire un meccanismo proprio
- ❑ Per non autenticare ad ogni richiesta, conviene usare le **sessioni** (maggiori dettagli dopo)

Esempio

```
<?php
function verify($a,$b)
{return($a== "user" && $b==13);};
if(!isset($_SERVER['PHP_AUTH_USER']) ||
    verify($_SERVER['PHP_AUTH_USER'],$_SERVER['PHP_AUTH_PW'])
    ==false){
    header('WWW-Authenticate: Basic realm="MyRealm"');
    header('HTTP/1.1 401 Unauthorized');
    echo 'Text to appear if user hits cancel';
    exit;
}
else {
    echo 'Correctly authenticated';
}
?>
```

HTTP header syntax



[authentication.php](#)

Gestione dei form: un esercizio

- ❑ Un meccanismo spesso utilizzato per la gestione di form semplici prevede che ci sia una sola pagina per il form e per la sua gestione,
 - ❖ una condizione verifica se le variabili sono settate,
 - ❖ se sì, vengono elaborate e costruita la risposta;
 - ❖ se no, viene restituito il form

Esercizio

- Sviluppiamo una calcolatrice, costituita da
 - ❖ un form con tre controlli (due operandi e un'operazione)
 - ❖ una pagina per la risposta

Possibile soluzione

□ Struttura:

```
<h1>Calculator:</h1>
```

```
<?php
```

```
if(isset($_GET['op1']) && isset($_GET['op2']) &&  
    $_GET['op1']!=" " && $_GET['op2']!=" ") {
```

```
// variables are set, process them
```

```
... Processare l'input e creare la risposta qui
```

```
} else {
```

```
// variables are not set, show form
```

```
?>
```

```
...
```

Mettere qui il form HTML

```
<?php
```

```
} // end of else branch
```

```
?>
```

[calculator.php](#)

Trattare i file caricati

- ❑ **Attenzione:** permettere a chiunque di caricare files può produrre inconvenienti
- ❑ I files caricati da un form con POST sono descritti nell 'array associativo `$_FILES`
- ❑ Esempio

```
<form action="http://myserver/files.php"
      enctype="multipart/form-data"
      method="post">
file <input type=file name=immagine>
<input type=submit value="INVIA">
</form>
```


Trattare i file caricati

- ❑ I files caricati da un form con POST sono descritti nell 'array associativo `$_FILES`

- ❑ Esempio

```
<form action="http://myserver/files.php"
      enctype="multipart/form-data"
      method="post">
```

```
file <input type=file name=immagine>
```

```
<input type=submit value="INVIA">
```

```
</form>
```

- ❑ La variabile `$_FILE['immagine']` contiene le informazioni sul file caricato (se è stato caricato)

- ❑ `$_FILE['immagine']` è un array associativo

Elementi \$_FILE['immagine']

- ❑ **name** filename originale (completo)
- ❑ **tmp_name** nome del file temporaneo locale
- ❑ **type** tipo MIME del file
- ❑ **size** dimensione in bytes del file
- ❑ **error** codice di errore
- ❑ Prima di leggere un file, è buona pratica controllare che:
 - ❖ il file non sia troppo grande o troppo piccolo di quanto atteso
 - ❖ il file sia del tipo atteso
 - ❖ non vi siano stati errori

Esempio

□ Come salvare il file

```
$dest="./img/";//dir. immagini
$dest=$dest.basename($_FILE['image']
                                ['name']);
if(move_uploaded_file($_FILE['image']
                    ['tmp_name'],$dest))
    { echo "file uploaded";}
else
    { echo "errore durante uploading
del file";}
}
```

estrae solo il nome

Altro esempio

□ Come controllare il tipo del file

❖ si può usare la function

```
stripos (pagliaio, ago[, offset=0])
```

`pagliaio` stringa in cui cercare

`ago` stringa da cercare

`offset` punto di partenza della ricerca (default 0)

❖ restituisce la posizione del primo `ago` trovato nel `pagliaio`, oppure falso

❖ Esempio

```
if (stripos($_FILE['immagine']['type'],  
           'image/')===false) //il file non ha  
    il tipo atteso...
```

Iniezione di codice

- ❑ Tecnica adottata da utenti maliziosi per modificare il comportamento normale di un sito
- ❑ Consiste nell'inserire del codice (es. HTML, JavaScript, PHP, SQL) nei campi di un form e farlo eseguire

❑ Esempio

```
<body>
```

```
<p> Ciao <?php echo $_POST['nome']; ?> ,
```

```
<br> tu hai <?php echo $_POST['eta']; ?> anni. </p>
```

```
</body>
```

- ❑ Un utente può forzare il server ad eseguire del codice arbitrario semplicemente mettendolo nei campi del form

Es. PHP Injection Vulnerability

<?php

```
if(isset($_GET['expr'])) { // expression set: evaluate
    eval("\$res=".$_GET['expr'].";");
    echo "<p>".$_GET['expr']."' = '".\$res."</p>";
    $script= $_SERVER['PHP_SELF'];
    echo "<p><a href=\"\$script\">Continue</a></p>";
    exit;
} else { // expression unset: show form
```

?>

```
<form method="get" action="vuln_php.php">
    <p><input type="text" name="expr" >
        <input type="submit" value="">
    </p>
</form>
```

<?php } // end of else branch ?>

[vuln_php.php](#)

Es: Command Injection Vulnerability

```
<?php
```

```
if(isset($_GET['name'])) { // name is set, process it
    system("nslookup ".$_GET['name']);
    $script= $_SERVER['PHP_SELF'] ;
    echo "<p><a href=\"".$script.\">Continue</a></p>";
    exit;
} else { // name is not set, show form
```

```
?>
```

```
<form method="get" action="vul_cmd.php">
<p> Name to be resolved:
    <input type="text" name="name" >
    <input type="submit">
</p>
</form>
```

```
<?php
```

```
} // end of else branch?>
```

[vuln_cmd.php](#)

Es: HTML Injection Vulnerability

<?php

```
if(isset($_GET['name'])) { // name is set, process it
    echo "Hello ".$_GET['name'].
        ". Your name has been saved!";
    $script= $_SERVER['PHP_SELF'] ;
    echo "<p><a href=\"".$script.">Continue</a></p>";
    exit;
} else { // name is not set, show form
```

?>

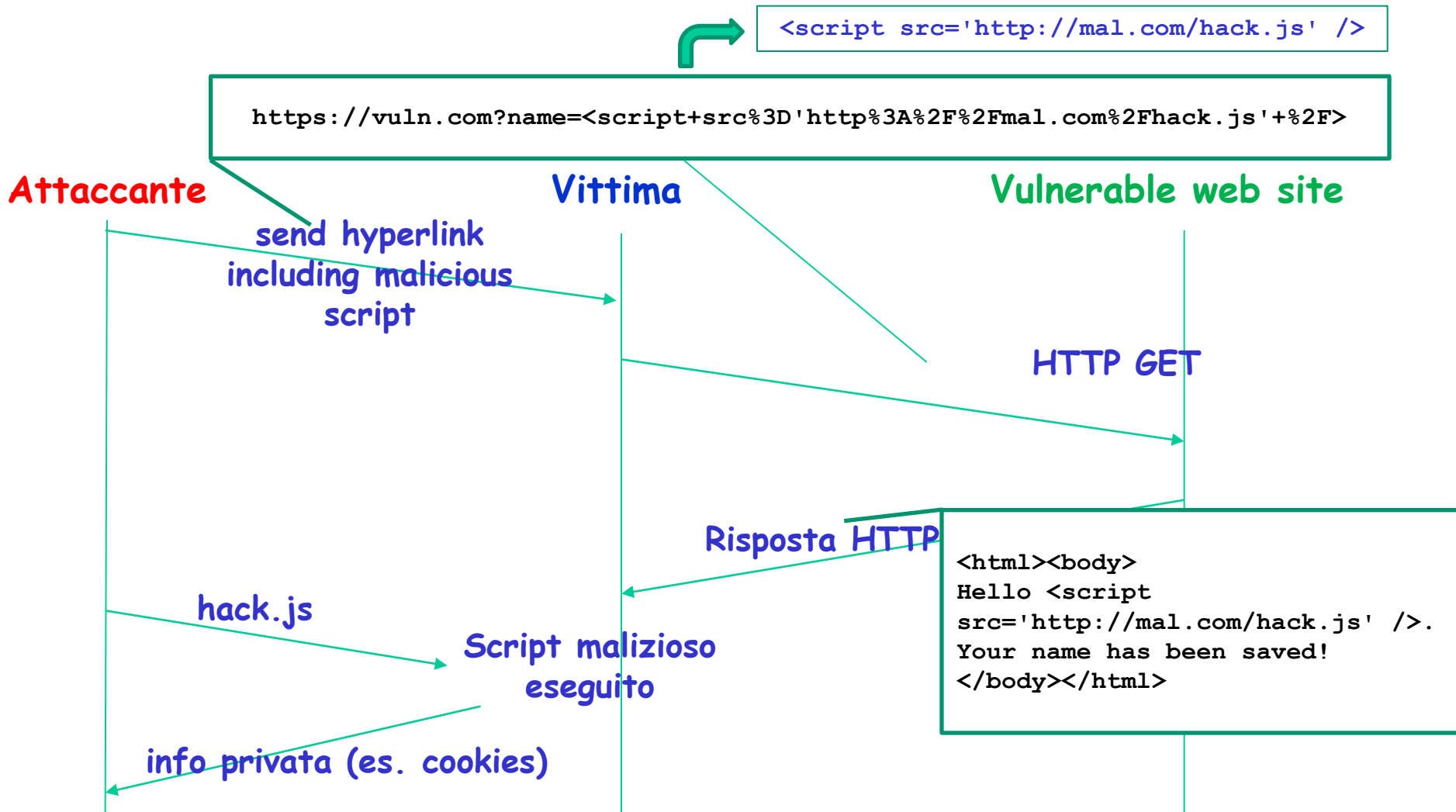
```
<form method="get" action="vul_xss.php">
<p> Enter your namne:
    <input type="text" name="name" >
    <input type="submit">
</p>
</form>
```

<?php

```
} // end of else branch?>
```

vuln_xss.php

Es: HTML Injection Vulnerability (Cross-Site Scripting - XSS)



Possibili contromisure

❑ Analisi delle stringhe di input

- ❖ Accettare solo stringhe che sono conformi a quanto atteso
- ❖ Usare tecniche di white-listing è meglio di usare black-listing
 - Cioè: accettare solo quanto esplicitamente permesso

❑ Sanificare le stringhe di input

- ❖ Rimuovere caratteristiche dannose dalle stringhe di input
- ❖ Una funzione utile per sanificare è:

`strip_tags (<string>[, <tag_adm>])`

- ❖ Ritorna la stringa `<string>` con tutti i tag HTML e PHP rimossi, e rimuove tutti i caratteri NULL
- ❖ `tag_adm` è una stringa con i tag che non devono essere rimossi

Esempio d'uso di strip_tags()

```
<?php
    $text = '<p>Test paragraph.</p><!--Comment-->
            <a href="#fragment">Other text</a>';
    echo strip_tags($text);
    echo "\n";
    // Allow <p> and <a>
    echo strip_tags($text, '<p><a>');
    ?>
```

Risultato

Test paragraph. Other text

<p>Test paragraph.</p>

Other text

Un'altra utile funzione

- ❑ **htmlspecialchars()** converte tutti i caratteri speciali in entità HTML, per esempio converte `<` in `<`;
- ❑ La differenza rispetto a **strip_tags** è che in questo caso i tokens sono convertiti invece di essere cancellati
- ❑ Conclusioni:
 - ❖ **TUTTO** l'input proveniente dall'esterno (es. utente) va **validato lato server** (ovviamente non e' sufficiente validazione lato client es. javascript)
- ❑ **Esercizio:** rimuovere le vulnerabilità degli es. precedenti

Accesso a file

- ❑ PHP include molte funzioni per questo scopo
 - ❖ Funzioni C-like:
 - fopen, fclose, fread, fwrite, fseek, ftell, feof, rewind, fgets, fputs, flock
 - ❖ Funzioni tipo shell:
 - copy, rename, unlink, file_exists, file (legge tutto il file)
- ❑ Molte di queste funzionano anche con
 - ❖ File remoti (via URLs)
 - ❖ Sockets
- ❑ L'accesso a file remoti può essere disabilitato nel file di configurazione php.ini

Funzioni principali per lettura

- ❑ **fopen** (*nome*, *modo* [, *include* [, *context*]])
 - ❖ *nome* stringa con il nome del file
 - ❖ *modo* stringa con il modo (simile al C)
 - ❖ *include* da mettere a 1 o TRUE se si vuole cercare anche nel direttorio specificato da `include_path`
 - ❖ *context* gruppo di variabili che specificano parametri per gli stream
- ❑ **fgets** (*handle* [, *lunghezza*])
 - ❖ Ritorna una stringa (la linea letta) o FALSE
 - ❖ Se *lunghezza* è specificata, legge solo fino a *lunghezza*-1 caratteri

Esempi

❑ Leggere un file linea per linea

```
<?php
```

```
$handle = @fopen("/tmp/inputfile.txt", "r");  
if ($handle) {  
    while (($buffer = fgets($handle, 4096)) !== false) {  
        echo $buffer;  
    }  
    if (!feof($handle)) {  
        echo "Error: unexpected fgets() fail\n";  
    }  
    fclose($handle);  
}
```

```
?>
```

Lettura di un file intero

- ❑ Leggere un file intero con una sola istruzione

`file (nome [, flag=0 [, context]])`

- ❖ Il contenuto del file viene letto e ritornato come un array di stringhe
- ❖ Ogni stringa include il fine linea (EOL), a meno che il flag `FILE_IGNORE_NEWLINE` sia specificato

Esempi

- ❑ Leggere e stampare un file di testo

```
<?php
```

```
$lines = file('http://www.example.com/');  
foreach ($lines as $line_num => $line)  
    {echo "Line #<b>{$line_num}</b> : ".  
        htmlspecialchars($line) . "<br>\n";  
    } ?>
```

- ❑ Leggere il file eliminando fine riga e righe vuote

```
<?php
```

```
$trimmed = file('somefile.txt',  
FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);  
?>
```

Scrittura di file

- ❑ **`fwrite(handle, stringa[, lunghezza])`**
 - ❖ Scrive la stringa nel file
 - ❖ Se *lunghezza* è specificato, scrive al max *lunghezza* caratteri
 - ❖ Ritorna il numero di caratteri scritti o FALSE se non è possibile scrivere

Sessioni

- ❑ HTTP è un protocollo **stateless**, ogni richiesta è indipendente dalle precedenti
- ❑ Il meccanismo delle sessioni è stato introdotto per consentire al server di riconoscere richieste collegate (es. dallo stesso browser)
- ❑ Utilità delle sessioni (esempi)
 - ❖ Autenticazione
 - ❖ Preferenze utente
 - ❖ Carrello della spesa
 - ❖ ...

Implementazione delle sessioni

- Una sessione è iniziata dal server
 - ❖ Il server crea un ID di sessione univoco e mantiene informazioni di sessione associate con questo ID
- L'ID è comunicato al client
 - ❖ Il client include l'ID in ogni richiesta successiva che deve essere riconosciuta come appartenente alla sessione
- La sessione è terminata dal server
 - ❖ Può accadere all'esecuzione di un comando di logout o quando scade un timeout (la sessione è rimasta inattiva per un certo tempo)

Implementazione delle sessioni

- ❑ Due meccanismi principali per l'implementazione delle sessioni
 - ❖ L'ID è inviato dal server al client come un «**cookie**», solamente con la prima risposta. Il cookie è immagazzinato sul client, che automaticamente lo invia al server con ogni richiesta
 - ❖ L'ID è codificato dal server negli **URLs** che sono scritti in ogni risposta, così che il client lo invia automaticamente quando fa accesso a quegli URLs
- ❑ NB: la sessione identifica il browser, non l'utente o l'host!

Sessioni in PHP

- ❑ Entrambi i meccanismi sono supportati. Il meccanismo dei cookies è quello di default
- ❑ Le informazioni di sessione possono includere un numero arbitrario di variabili
- ❑ L'ID di sessione può essere scelto
 - ❖ Automaticamente dal sistema
 - ❖ Dal programmatore (ma non deve essere facilmente «indovinabile»)
- ❑ L'accesso agli oggetti (variabili) nella sessione è in **mutua esclusione** (può avere impatto negativo sulle performance)

Sessioni con ID di sistema

- ❑ Bisogna usare la function

`session_start()`

❖ IMPORTANTE: la chiamata deve precedere qualsiasi output dello script

- ❑ Se una sessione non esiste ne crea una, oppure recupera la sessione già esistente
- ❑ L'ID proviene (di default) da un **cookie** nella richiesta
- ❑ Si possono definire le variabili da salvare semplicemente aggiungendo elementi all'array

`$_SESSION`

dopo aver chiamato `session_start()`

Esempio

```
<?php
```

```
    session_start(); //crea o recup. sess.
```

```
    $_SESSION['bgcolor']='yellow';
```

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

```
    $a=$_SESSION['date']; //se esiste!
```

```
.....
```

```
?>
```

NB: Attenzione ai nomi delle variabili usate: se il server è unico e il browser non viene chiuso la sessione rimane la stessa anche se l'applicazione è diversa (server in lab)

Esempio: contare i contatti

```
<?php
    session_start();
    if (isset($_SESSION['count']) {
        $i=$_SESSION['count'];
    } else {
        $i=0;
    }
?>
<html>
<head><title>Esempio di concorrenza</title></head>
<body>
<h1> questo &grave; il collegamento numero:</h1>
<p>
<?php
    echo $i;
    $_SESSION['count']=$i+1;
?>
</body>
</html>
```

Commenti sull'esempio

- ❑ Modificare l'esempio precedente così che un ritardo sia introdotto tra la lettura del contatore e il suo incremento
- ❑ Caricare la pagina più volte in maniera concorrente
- ❑ Cosa si verifica?
 - ❖ Le richieste accedono ai dati di SESSIONE in mutua esclusione (gli accessi sono serializzati)

Variante con sessione unica

```
<?php
    session_id("sessione-unica");    // ID impostato dal programmatore
    session_start();
    if (isset($_SESSION['count']))
        { $i=$_SESSION['count'];
          }
        else { $i=0;} ?>

<html>
<head><title>Esempio di concorrenza</title></head>
<body>
<h1> questo è il collegamento numero:</h1>
<p>
<?php
echo $i;
$_SESSION['count']=$i+1;
session_write_close();    // Rilascia la mutua esclusione sulla sessione
if (isset($_REQUEST['time'])) {sleep($_REQUEST['time']);}
    else {sleep(20);};    // Consente di verificare la mutua esclusione
echo "<p>".session_id();
?>
    </body>
</html>
```

[concurrency1.php](#)

Gestione a tempo della sessione

- ❑ I dati delle sessioni sono in appositi files sul server
 - ❖ Se non sono distrutte automaticamente dopo un certo tempo di inattività possono creare leakages
- ❑ In PHP le sessioni troppo inattive sono eliminate da un **garbage collector**
- ❑ La vita garantita di una sessione può essere controllata con la function

```
ini_set('session.gc_maxlifetime',  
1800);
```

↑
secondi

↑
parametro nel file php.ini

Altri problemi

- ❑ Tutte le sessioni sono memorizzate insieme
- ❑ Il tempo di vita può essere differente per diversi siti co-localizzati
- ❑ Si possono memorizzare cartelle diverse le sessioni di siti differenti
- ❑ Esempio:

```
/* imposta cartella per le sess. di questo sito */  
session_save_path('/tmp/mydir');  
/* imposta periodo di gc a 7 giorni*/  
ini_set('session.gc_maxlifetime', 7*24*3600);  
/*imposta durata e path nei cookie */  
session_set_cookie_params(1800, '/');  
session_start();
```

Gestione esplicita di inattività

```
<?php
session_start(); $t=time(); $diff=0; $new=false;
if (isset($_SESSION['time'])) {
    $t0=$_SESSION['time']; $diff=($t-$t0); // inactivity
} else {
    $new=true;
}
if ($new || ($diff > 10)) { // new or with inactivity period too long
    //session_unset(); // Deprecated
    $_SESSION=array();
    // If it's desired to kill the session, also delete the session cookie.
    // Note: This will destroy the session, and not just the session data!
    if (ini_get("session.use_cookies")) { // PHP using cookies to handle session
        $params = session_get_cookie_params();
        setcookie(session_name(), '', time() - 3600*24, $params["path"],
            $params["domain"], $params["secure"], $params["httponly"]);
    }
    session_destroy(); // destroy session
    // redirect client to login page
    header('HTTP/1.1 307 temporary redirect');
    header('Location: login.php?msg=SessionTimeOut');
    exit; // IMPORTANT to avoid further output from the script
} else {
    $_SESSION['time']=time(); /* update time */
    echo '<html><body>Tempo ultimo accesso aggiornato: '
        .$_SESSION['time'].'</body></html>';
}
?>
```

Evita che il cookie non scaduto sia inviato dal browser al server PHP e venga usato per creare una nuova sessione alla chiamata di session_start()

[sessions.php](#)

Altre funzioni legate alle sessioni

❑ `session_get_cookie_params()`

❖ restituisce un array associativo con indici

• `lifetime` , `path` , `domain` , `secure` , `httponly`

❑ `session_set_cookie_params()`

❖ Utile (per esempio) per restringere l'utilizzo del cookie da parte del client al solo utilizzo su canale sicuro

❑ `session_id ([sid])`

❖ restituisce il valore corrente dell'id

❖ setta un nuovo id al valore del parametro (se usato), da usare prima di `session_start()`

❑ `session_regenerate_id ()`

❖ Rigenera il valore dell'ID di sessione (es. cookie)

Funzioni per controllo cache

- ❑ I cookies sono in una cache nel **cliente**
- ❑ **`session_cache_expire`**(*durata*)
 - ❖ imposta la durata di tutti i cookie della sessione
 - ❖ durata in minuti
- ❑ **`session_cache_delimiter`**(*tipo*)
 - ❖ imposta parametri per il cache-control di pagine HTML
 - ❖ alcuni valori
 - *public*, normale (anche da parte di proxies)
 - *nocache*, non si può mettere la risposta in cache
 - *private*, in cache private
 - *private_no_expire*, senza scadenza, in cache private

«Include» di codice PHP

- ❑ E' possibile includere e interpretare il contenuto di un altro file tramite **include**
 - ❖ Esempio: **include 'file.php' ;**
- ❑ Particolarmente utile per porzioni di codice PHP che devono essere replicate in più pagine
 - ❖ Funzioni di utilità
 - ❖ Controllo dei diritti di accesso (es. presenza di sessioni, timeout delle sessioni, ecc.)
 - ❖ Usare questo meccanismo e non il «copia e incolla» che facilita errori e rende difficile la manutenzione del codice

Controllo richiesta su HTTPS

```
if ( !empty($_SERVER['HTTPS']) && $_SERVER['HTTPS'] !== 'off') ) {  
    // La richiesta e' stata fatta su HTTPS  
} else {  
    // Redirect su HTTPS  
    // eventuale distruzione sessione e cookie relativo  
    $redirect = 'https://' . $_SERVER['HTTP_HOST'] .  
        $_SERVER['REQUEST_URI'];  
    header('HTTP/1.1 301 Moved Permanently');  
    header('Location: ' . $redirect);  
    exit();  
}
```