Bloque I: Seguridad web

1. ¿Por qué es importante la seguridad?
2. Validación
3. Escapado
4. Ataques XSS
5. Ataques CSRF
5. SQL-Injection
6. Code Injection
7. Session riding

*POO = Programación Orientada a Objetos

Bloque I: Seguridad web

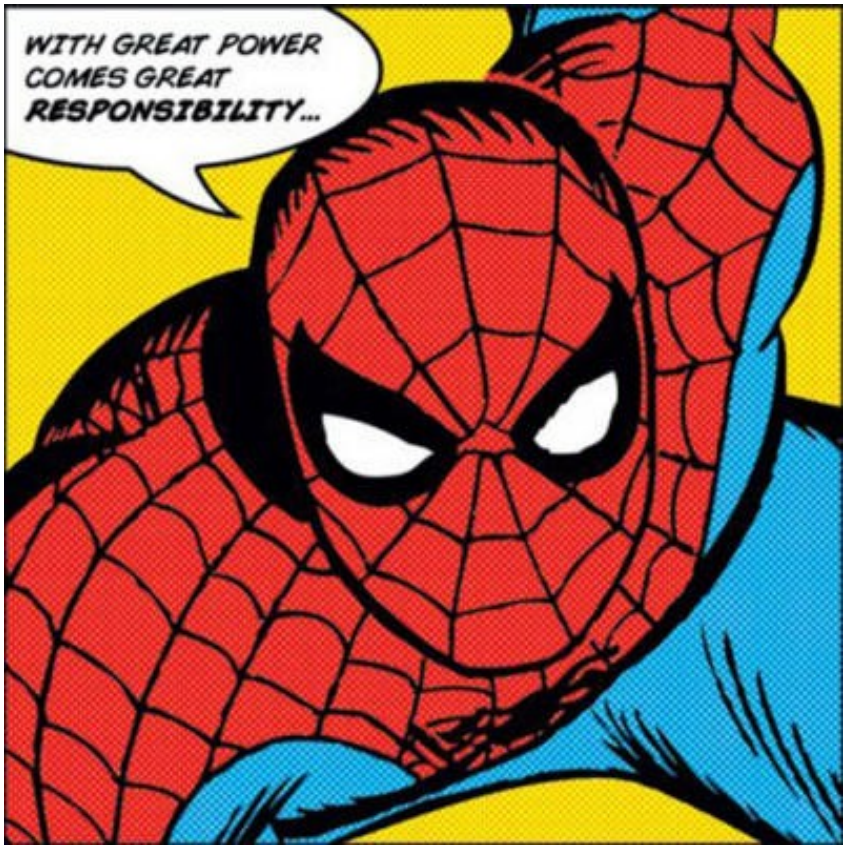
1. ¿Por qué es importante la seguridad?
2. Validación
3. Escapado
4. Ataques XSS
5. Ataques CSRF
5. SQL-Injection
6. Code Injection
7. Session riding

*POO = Programación Orientada a Objetos

3.1. ¿Por qué es importante la seguridad?

Un gran poder conlleva

Una gran responsabilidad



3.1. ¿Por qué es importante la seguridad?

Prácticamente todas las aplicaciones PHP están desarrolladas para la web

Y la web es un sitio peligroso

PHP es gratuito y fácil de aprender

PHP es muy atractivo para amateurs

PHP está siendo utilizado cada vez en sitios más grandes

Y la información que gestiona es cada vez más sensible

3.1. ¿Por qué es importante la seguridad?

Las principales prácticas se puede agrupar en dos ideas

Validación

Escapado

Bloque I: Seguridad web

1. ¿Por qué es importante la seguridad?
2. Validación
3. Escapado
4. Ataques XSS
5. Ataques CSRF
5. SQL-Injection
6. Code Injection
7. Session riding

*POO = Programación Orientada a Objetos

3.2. Validación

Todos los datos de entrada están “contaminados”

TODOS LOS DATOS DE ENTRADA

3.2. Validación

Mis entradas de datos

Query string
http://www.google.com/**?search=hola**

Canales RSS
http://www.loalf.com/feed.rss

API's de terceros
http://dev.twitter.com

Cookies

Formularios

3.2. Validación

¿Por qué hay que validar los datos de entrada?

Evitar comportamientos no esperados de la aplicación

Evitar ganar acceso a una aplicación a la que no está autorizado

Evitar que la aplicación “se rompa” de manera malintencionada

3.2. Validación

Register Globals

Es una directiva de configuración que automáticamente inyecta variables en el script.

```php
<?php
// http://www.misitio.com/index.php?name=javi
echo $name; // javi
?>
```

Desactivado por defecto desde PHP 4.2.0 y no debería activarse

3.2. Validación

Un ejemplo

```
<form action="process.php" method="post" >
  Nombre: <input type="text" name="nombre" /><br />
  Edad: <input type="text" name="edad" /><br />
  Color preferido:
  <select name="color" >
    <option value="azul">Azul</option>
    <option value="verde">Verde</option>
    <option value="amarillo">Amarillo</option>
    <option value="rojo">Rojo</option>
  </select>
  <input type="submit" />
</form>
```

3.2. Validación

Validación numérica

Todos los datos recibidos por PHP (GET/POST/COOKIE) son cadenas de texto.
Hay que convertirlos a números para que sea más eficiente y menos peligroso.

Deben ser convertidos a números mediante un casting

```
(int) $_POST['edad'];

(float) $_POST['precio'];
```

3.2. Validación

Validación de texto

PHP viene por defecto con CTYPE, una extensión que permite de manera cómoda validar cadenas de texto

```
if(!ctype_alnum($_POST['password'])){
  echo “Sólo caracteres A-Za-z0-9
permitidos”;
}

if(!ctype_alpha($_POST['username'])){
  echo “Sólo caracteres A-Za-z permitidos”;
}
```

Y otras muchas más funciones
http://php.net/manual/en/book.ctype.php

3.2. Validación

Validación complejas

¿Como validar un email, una URL, una IP, un DNI, una fecha? Expresiones regulares.

```
if(!preg_match(
    '/^\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,3}$/',
    $_POST['email'])
){

    echo “Email no válido”;
}
```

No hace falta saber expresiones regulares, sólo saber dónde encontrarlas
http://regexlib.com

3.2. Validación

Todo cambia

Con letra? España?

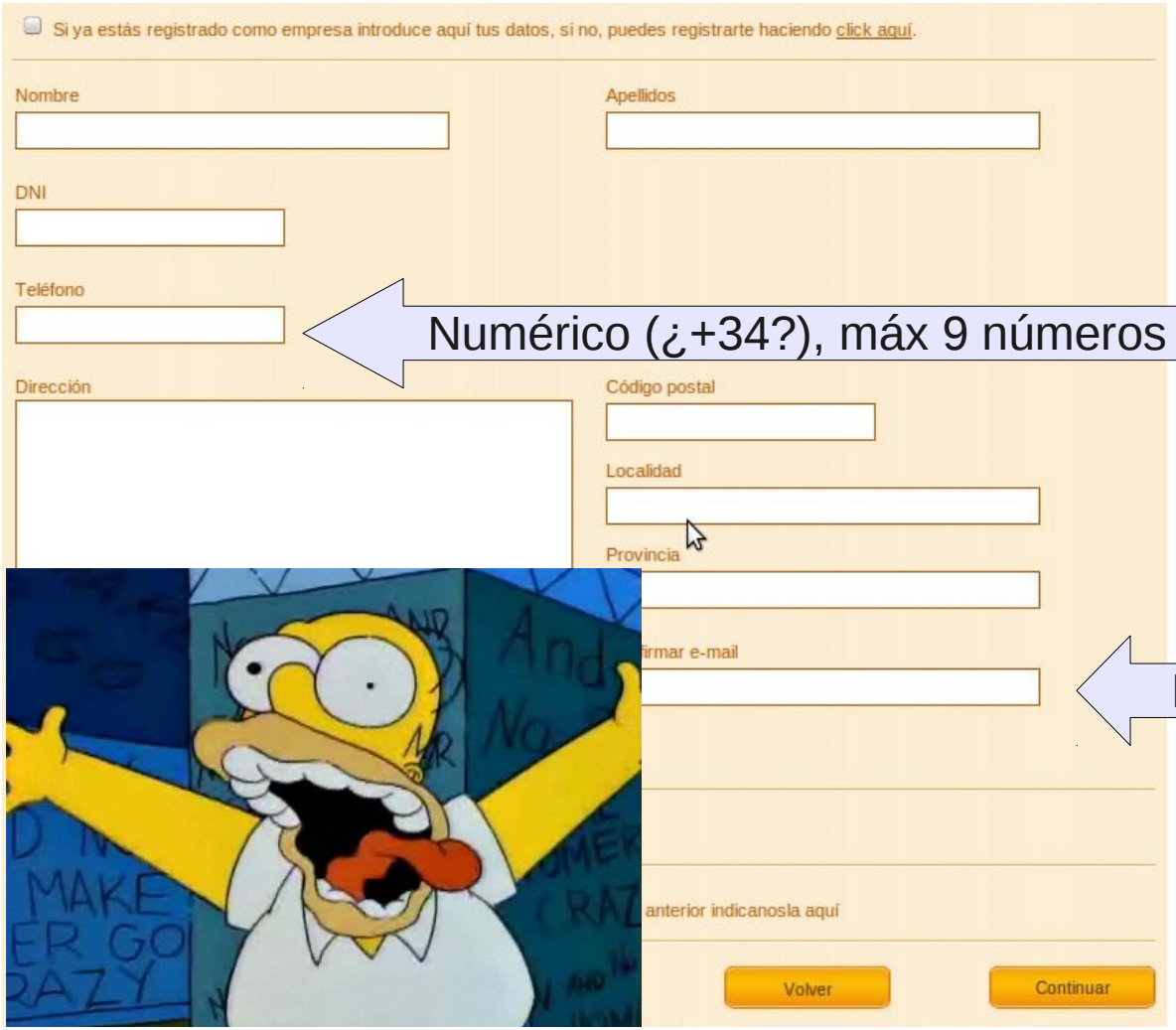
Numérico (¿+34?), máx 9 números

Un email

Igual que el otro

Obligatorio

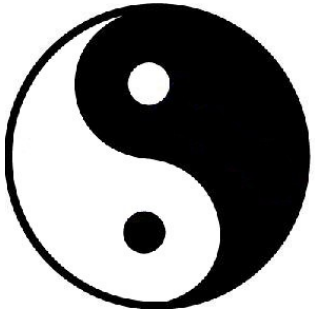
Más campos ...



3.2. Validación

Validación en cliente vs servidor

Validación cliente
Usabilidad



Validación cliente
Seguridad

Nadie dijo que se excluyentes

3.2. Validación

Burlando formularios

1. Deshabilitar Javascript

2. Enviar formularios desde web

3. Instalando Firebug

Bloque I: Seguridad web

1. ¿Por qué es importante la seguridad?
2. Validación
3. Escapado
4. Ataques XSS
5. Ataques CSRF
5. SQL-Injection
6. Code Injection
7. Session riding

*POO = Programación Orientada a Objetos

3.3. Escapado

¿En que consiste?

Evitar que el código generado por nuestra aplicación pueda resultar dañino

Dependiendo del dónde imprimamos la información las reglas de escapado son diferentes

3.3. Escapado

Escapando datos para generar HTML

`htmlentities()`

Ejemplo

```
echo htmlentities('<b>Texto en negrita</b>');
// &gt;b&lt;Texto en negrita&gt;/b&lt;
```

3.3. Escapado

Escapando datos para generar SQL

```
*_escape_string()
```

Para el caso concreto de mysql

```
echo mysql_escape_string(“WHERE name='javi'”);
// WHERE name = \'javi\'
```

3.3. Escapado

Evitad cosas como estas

```
$query = “SELECT * FROM users WHERE
          username = '{$_POST['username']}'”;
```

Utilizad en su lugar Prepared Statements

http://www.slideshare.net/flaiwebnected/iniciacin-php-5-php-y-mysql

3.3. Escapado

```
$sql = 'SELECT * FROM users
        WHERE username = :username';

$stmt = $dbh->prepare($sql);
$stmt->bindParam(':username', $clean['username'])

$stmt->execute();
$results = $stmt->fetchAll();
```

Bloque I: Seguridad web

1. ¿Por qué es importante la seguridad?
2. Validación
3. Escapado
4. Ataques XSS
5. Ataques CSRF
5. SQL-Injection
6. Code Injection
7. Session riding

*POO = Programación Orientada a Objetos

3.4. Ataque XSS

XSS = Cross Site Scripting

Estos ataques se aprovechan de la confianza del usuario en la aplicación

El atacante inyecta código JavaScript en nuestra web

3.4. Ataque XSS

Algunas cosas que podría lograr mediante un ataque XSS

- Robar las cookies de un usuario y entrar en su lugar
- Mostrar una página falsa de login en tu web (phising)
- Llevar a cabo cualquier acción como si fuera otro usuario

3.4. Ataque XSS

Dos tipos de ataques XSS

Persistentes

Son “menos” frecuentes

Datos introducidos por el usuario son almacenados de manera PERSISTENTE (db) y utilizados para generar una página HTML

Reflejados

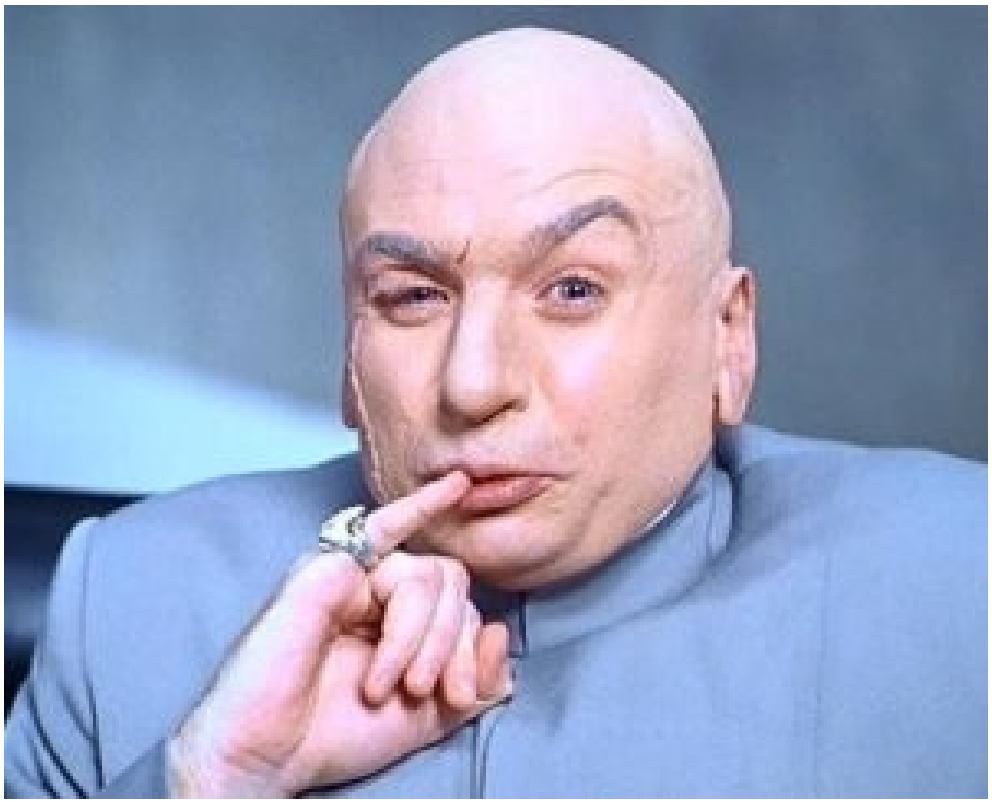
Son los más frecuentes

Datos introducidos por el usuario son utilizados INMEDIATAMENTE para generar una página HTML

3.4. Ataque XSS

Algunos ejemplos...

… malignos



3.4. Ataque XSS

¿Quién es Samy? http://namb.la/popular/

Un agujero XSS en el filtro HTML de MySpace

Cuando visitabas el perfil de Samy

- Te añadía a tí como amigo suyo
- Se clonaba en tu propio perfil

En 20 y tras un millón de peticiones amigos MySpace se vino abajo

Bloque I: Seguridad web

1. ¿Por qué es importante la seguridad?
2. Validación
3. Escapado
4. Ataques XSS
5. Ataques CSRF
5. SQL-Injection
6. Code Injection
7. Session riding

*POO = Programación Orientada a Objetos

3.5. Ataque CSRF

CSRF = Cross site request forgery

Explota la confianza de un sitio web en un usuario

La víctima del ataque lanza una petición HTTP sin saberlo, normalmente a URLs Que requieren de un acceso privilegiado.

El escapado evitará que tu aplicación sirva de vehículo para ataques CSRF pero no evitará que los recibas.

3.5. Ataque CSRF

Un ejemplo de ataque CSRF

3.5. Ataque CSRF

Cómo protegerse de este tipo de ataques

```
<?php
session_start();
$token = md5(uniqid(rand(), TRUE));
$_SESSION['token'] = $token;
?>

<form action="miform.php" method="POST">
  ...
  <input type="hidden"
         name="token"
         value="<?php echo $token; ?>"
  />
</form>
```

3.5. Ataque CSRF

Cómo protegerse de este tipo de ataques

```
<?php

if(isset($_SESSION['token'])
   && isset($_POST['token'])
   && $_POST['token'] == $_SESSION['token']
){
   //No es un ataque CSRF
}

?>
```

3.5. Ataque CSRF

Google tampoco se salva

http://www.securiteam.com/securitynews/5ZP010UQKK.html

Mediante un ataque CSRF se podía cambiar la contraseña de un usuario

Bloque I: Seguridad web

1. ¿Por qué es importante la seguridad?
2. Validación
3. Escapado
4. Ataques XSS
5. Ataques CSRF
5. SQL-Injection
6. Code Injection
7. Session riding

*POO = Programación Orientada a Objetos

3.6. SQL Injections

SQL injections o inyección de SQL

Un usuario utiliza alguna de las entradas (normalmente formularios) para modificar las sentencias SQL en la aplicación.

Las entradas consisten en código SQL parcial, que combinado con el existente da lugar a un comportamiento no deseado

Cómo protegerse: escapa los parámetros de tus sentencias SQL

3.6. SQL Injections

Un ejemplo (un clásico)

Formulario de acceso a una zona restriginda (formulario de login)

```
<form action="login.php" action="POST">
  usuario: <input type="text" name="username" />
  <br />
  contraseña: <input type="password" name="password" />
  <br />
  <input type="submit" />
</form>
```

3.6. SQL Injections

Un ejemplo (un clásico)

En algún lugar de nuestro script comprobamos que ese usuario existe de la siguiente manera

```
$username = $_POST['username'];
$password = $_POST['password'];
$sql = "SELECT *
        FORM users
        WHERE username = '{$username}'
        AND password = '{$password}'";
// Continuamos con el resto
if(count($results)>0){ // login }
```

3.6. SQL Injections

Un ejemplo (un clásico)

Que ocurre si el usuario introduce

```
username' OR 1=1 --
```

Tenemos esto

```
SELECT *
FORM users
WHERE username = 'username'
OR 1=1 --'AND password = 'contraseña'
```

3.6. SQL Injections