## Seminario de Tecnologías Web


# Seguridad en Aplicaciones Web

manuel.freire@fdi.ucm.es – 29 de abril de 2011

# Amenazas

*Cosas malas que pueden pasar*

- Denegación de servicio
  - interrupción del servicio
  - DDOS: distribuida, con múltiples ataques simultáneos
- Acceso a datos de otros usuarios
  - ver ó modificar datos de otros sin estar autorizado
- Suplantación de identidad
  - hacerse pasar por otro usuario
- Acceso a partes privadas de la aplicación
  - ver ó modificar datos ó código de la aplicación que no están pensados para ser accedidos externamente

# Vectores

*Formas de las que pueden llegar las cosas malas*

- Externos a la aplicación (inevitables; pero disminuir su gravedad)
  - acceso físico al servidor
  - vulnerabilidad local en el servidor (ej.: programa local sin parchear)
  - vulnerabilidad remota en el servidor (ej.: php sin parchear)
- Internos a la aplicación (puedes protegerte escribiendo buen código)

  peticiones HTTP de tipo **GET** y **POST**, que pueden ocasionar
  - inyección de SQL: contra la BD
  - inyección de HTML ó JS en el PHP generado:
    contra visitantes del sitio
  - inyección de ficheros: machacando ficheros de otros usuarios, la aplicación o el sistema *(= vulnerabilidad remota para terceros)*
  - datos incorrectos: contra la validación

# Protección contra vectores internos

- Inyección SQL:

  Cuando un campo de formulario se concatena a una consulta SQL, y resulta que contiene SQL que interacciona con la consulta

  - usa "**prepared statements**" (recomendado) ó
  - escapa muy cuidadosamente todos los argumentos

- Inyección de HTML ó JS:

  Cuando un campo de formulario, al insertarse en la página HTML generada, contiene JS ó HTML que interacciona con la página circundante

  - valida todos los campos **y**
  - **codifica** según lenguaje donde vas a insertar el texto:

```
$html_escapado = htmlspecialchars($entrada);
$cadena_js_escapada =
    '"' . str_replace('"', '\"',  $entrada) . '"';
```

```
// NOTA: cadena_js_escapada podría fallar si existen lineas; usa
$entrada = str_replace(array("\r\n", "\n", "\r", '',  $entrada);
// o mejor, usa JSON: json_decode + json_encode son 100% seguras
```

# Vectores internos: ficheros

- Desconfía de los ficheros que te suben
  - No te fíes del nombre o la extensión: descártalos
  - No te fíes del contenido: usa

```
$finfo = new finfo(FILEINFO_MIME);
$type = $finfo->file($fichero);
```

- Localización:
  - Usa una carpeta `data/` en la raíz de tu aplicación
  - Cada archivo subido debe estar relacionado con una tabla; usa el nombre de la tabla como directorio: `data/usuarios/`
- Esquemas nombrado:
  - usa el ID de la fila como nombre: `data/usuarios/1.jpg`
  - para evitar acceso directo, usa `.htaccess` y sirvelos vía PHP:

```
// aquí podrías hacer control de acceso
header("Content-type: image/jpeg");
$img = imagecreatefromjpeg($rutaFichero);
imagejpeg($img);
```

código php para devolver un fichero-imagen

# Vectores internos: datos incorrectos

- Nunca asumas que lo llega es sólo lo que esperabas recibir
  - Es posible escribir URLs, sin necesidad de hacer click en enlaces
    tu seguridad no debería depender de URLs difíciles de adivinar
  - Es posible modificar parámetros GET con sólo editar una URL
  - Es posible editar peticiones POST, con todos sus parámetros

```
admin.php?op=borraUsuario&login=paco
```

admin.php debería verificar si el usuario que está haciendo la petición tiene o no derecho de visitar la página 'admin.php'…

```
carrito.php?producto=123&cantidad=2&precio=36
```

carrito.php debería recalcular el precio por su cuenta, sin mirar lo que le dice el cliente (a no ser que se trate de una puja)

*VALIDA, VALIDA y VALIDA*

# Vectores externos: gestión de contraseñas

- Muchos usuarios reutilizan contraseñas en múltiples sitios
  (y puede que sean mucho más importantes que el tuyo)

- Peligros de almacenar contraseñas "en texto claro":
  - Un atacante puede conseguir acceso a la lista
    (ya sea por vector interno ó externo)
  - Es probable que varias contraseñas sirvan también en otros sitios

- Forma recomendada de manejar contraseñas
  - Mantenlas en "texto claro" **el menor tiempo posible**
    (y **nunca las almacenes** en fichero ó log)
  - Guarda las contraseñas en un "***formato modificado***" en el que, aunque un atacante consiga la lista:
    - No pueda convertirlas a las contraseñas **originales**
    - No pueda **acceder** a tu sitio con ellas

# Hash

- *hash*: significa "troceo y remezclado", ó "triturado"
- hash criptográfico: función que
  - Calcula un bloque de tamaño fijo a partir de un mensaje arbitrario
  - Está diseñada para que sea computacionalmente imposible
    - calcular, dado un hash, un mensaje del que pueda haberse originado
    - modificar un mensaje sin modificar su hash
    - encontrar dos mensajes con el mismo hash
- ejemplos:
  - md5 (popular, algo inseguro)
  - sha1 (recomendado)
  - sha256 (su sucesor)

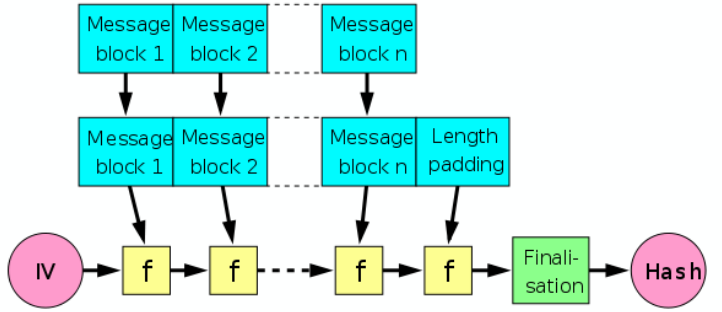


diagrama de hashes tipo merkle (sha1, …)

```
sha1('apple'); //'d0be2dc421be4fcd0172e5afceea3970e2f3d940'
```

código php para usar sha1

# Procesamiento de contraseñas

```
// hash (contraseña + sal aleatoria + constante)
function auth_encripta($pass, $salt) {
    return sha1($pass . $salt . "tablon");
}
```

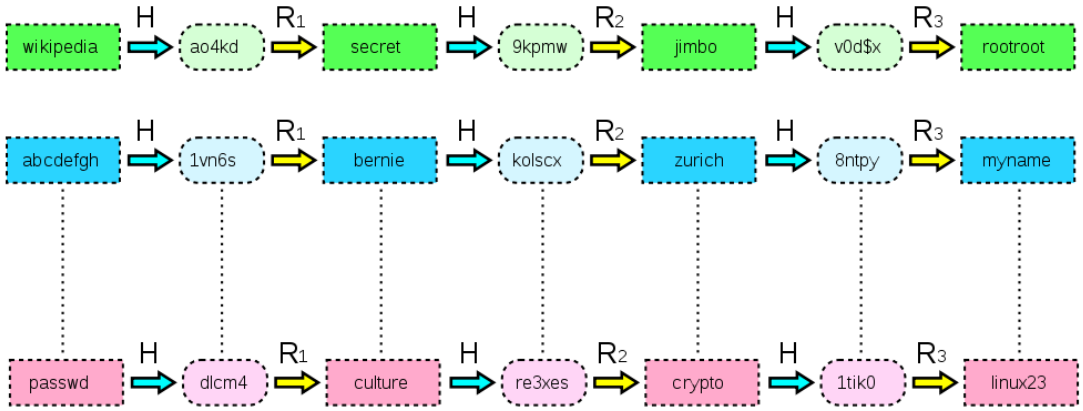
```
// inserta un nuevo usuario en la BD
function auth_add_user($db, $login_, $pass_, $role) {
    $salt = auth_aleat(12); // cadena aleatoria de 12 letras
    $hashed = auth_encripta($pass_, $salt);
    // inserta ($login_, $hashed, $salt, $role) en BD
}
```

```
// verifica que una contraseña es correcta
function auth_check_login($db, $login_, $pass_) {
    // consulta ($hashed, $salt) en BD, buscando por $login_
    return auth_encripta($pass_, $salt) == $hashed;
}
```

del código de "auth.php" en *tablon*

# **Porqué usar** sha1($pass . $sal . $cte)

- Asume el caso peor:
  tu atacante tiene acceso a la lista de contraseñas
- Si **no** hay sal, puede
  - encontrar rápidamente si hay dos usuarios con la misma contraseña
  - probar todas las palabras de un diccionario, y ver si alguna coincide con uno de los hashes de la lista (y así obtener su contraseña)
  - preparar ó adquirir una "tabla arcoiris" (equivale al diccionario)




una tabla arcoiris; ahorra memoria con respecto a un diccionario hash => palabra

- Y la constante permite usar sales algo más cortas

# Vectores externos: gestión de tarjetas de crédito

- Mucho peor que contraseñas
  - tamaño fijo: 16 dígitos, de los cuales
    - 6 son del banco (adivinables);
    - 1 es redundante (el último se calcula a partir de los anteriores)
    - los 4 últimos deberían ser buscables
      (te podrían hacer falta para resolver incidencias)
  - Romper 5 dígitos por fuerza bruta es fácil,
    hagas lo que hagas; no hay hash que valga
- Recomendación: **evita gestionarlas**
  - No las almacenes
  - Delega en expertos y usa sus APIs y servidores

*(nota: no deberíais gestionar tarjetas en vuestra aplicación, sólo incluyo esta transparencia a título orientativo)*

# Bibliografía


- Manual de PHP
  http://php.net/manual/en/index.php
- Wikipedia
  https://en.wikipedia.org/wiki/Cryptographic_hash_function
  https://en.wikipedia.org/wiki/Password_cracking
- Sobre tarjetas de crédito
  https://sites.google.com/site/yacoset/Home/how-to-secure-a-credit-card-number
- StackOverflow

  http://stackoverflow.com/questions/1263957/why-is-mime-content-type-deprecated-in-php
  http://stackoverflow.com/questions/1353850/
  serve-image-with-php-script-vs-direct-loading-image
  http://stackoverflow.com/questions/129677/
  whats-the-best-method-for-sanitizing-user-input-with-php