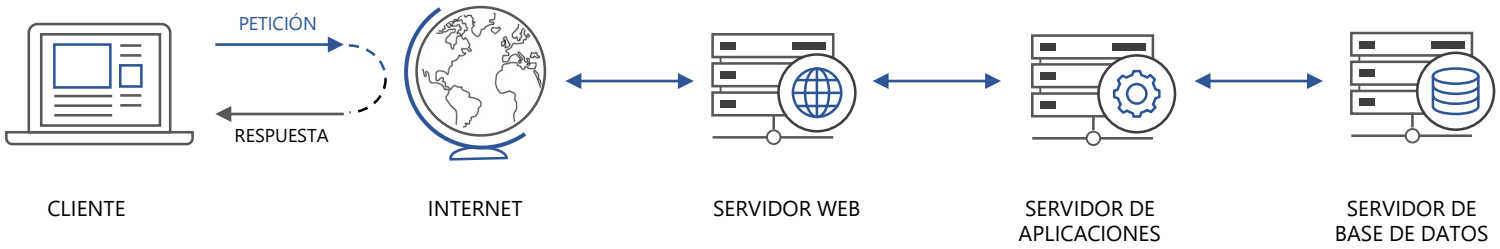
Desarrollo seguro con PHP

Controles de seguridad que todo programador debe conocer y aplicar en proyectos de desarrollo Web.

Karla Fonseca Márquez

Las Aplicaciones Web son Complejas



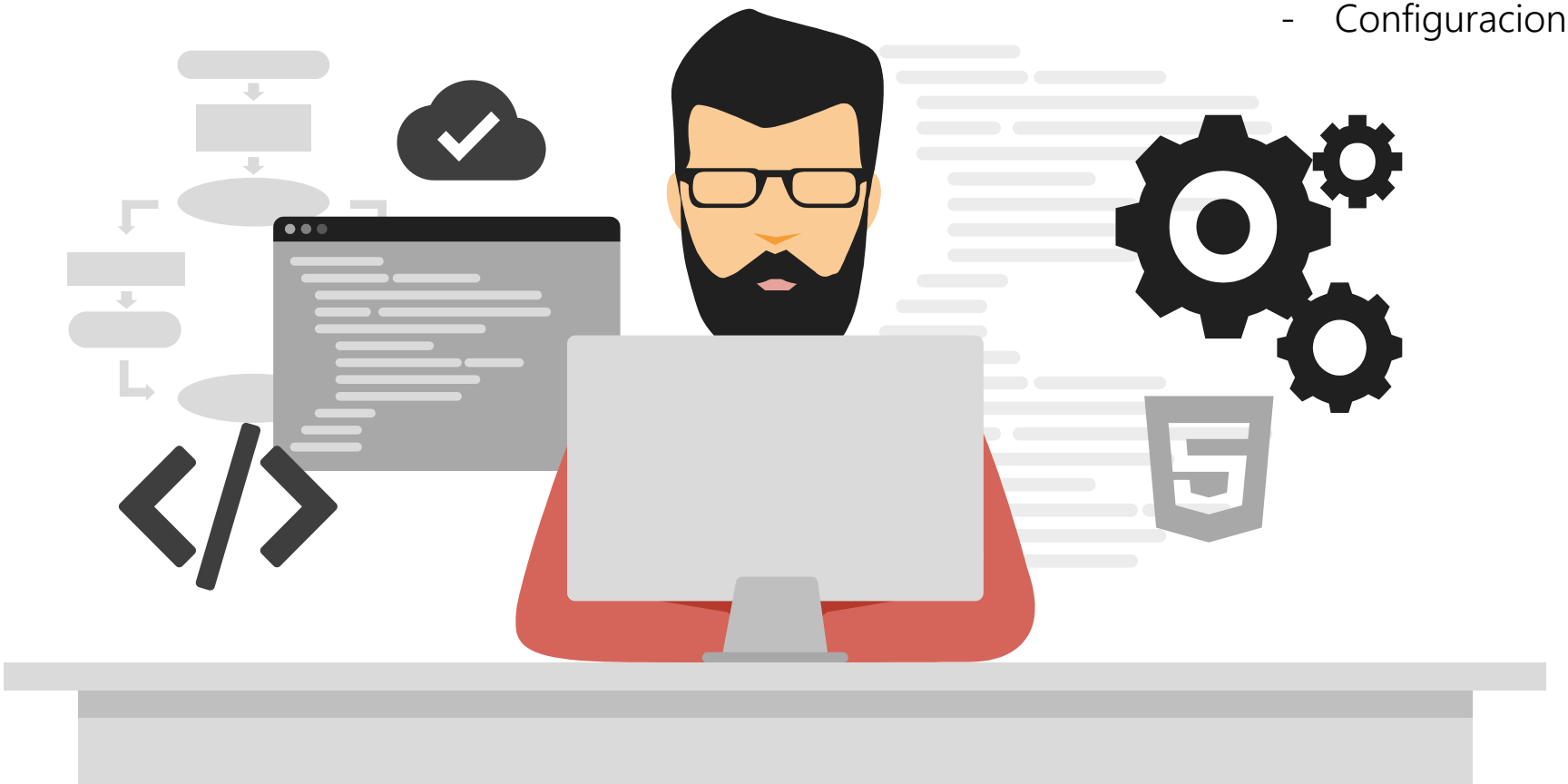
Atributos de calidad
- Disponibilidad
- Usabilidad
- Eficiencia
- Mantenibilidad
- Seguridad

Selección de tecnologías
- Lenguajes
- Plataformas
- Librerías
- Manejadores de BD

Requerimientos funcionales
- Casos de uso
- Historias de usuario

Stack
- Front end
- Back end

Ambiente
- Internet
- Red
- Sistema Operativo
- Hardware
- Configuraciones



PHP

- 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.
- PHP es un blanco muy popular para ataques.
  - Por default, se considera inseguro
- Para que sea segura, se requiere al menos de:
  - Programación segura
  - Configuraciones seguras

Programación segura

- La programación segura es la práctica de escribir software que está protegido de vulnerabilidades.
- Es importante para el desarrollo de cualquier software, tanto si se ejecuta en dispositivos móviles, computadoras personales, servidores o cualquier otro dispositivo embebido.
- Implica el utilizar diversas técnicas y herramientas a lo largo de todo el ciclo de desarrollo de software.

El programador es responsable de la seguridad de la aplicación

¿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
- Creencia que la seguridad es responsabilidad de los encargados de la red o del servidor
- Falta de experiencia y capacitación
- Creer no somos el objetivo de ningún ataque

OWASP

Open Web Application Security Project

- Organización sin fines de lucro dedicada a mejorar la seguridad del software.
- Los materiales y guías que ofrece son gratuitos y disponibles bajo una licencia de software libre para que cualquiera pueda utilizarlos.

OWASP Top Ten Proactive Controls 2018

1. Define requerimientos de seguridad

¿Requerimientos de seguridad?

- “La aplicación debe ser segura.”
- “La aplicación debe estar protegida ante todos los ataques conocidos para este tipo de aplicaciones.”
- “La aplicación debe defenderse ante los 10 riesgos del Top Ten de OWASP”

¿Cómo definirlos?

- Los requerimientos de seguridad se derivan de estándares de la industria, leyes aplicables y de vulnerabilidades pasadas.
- OWASP Application Security Verification Standard (ASVS) es un catálogo de requerimientos de seguridad y de criterios de verificación.
- Busca establecer un marco de trabajo de requerimientos de seguridad y controles requeridos al diseñar, desarrollar y probar aplicaciones Web

OWASP ASVS

01

Mínimo

Las aplicaciones con este nivel se defienden contra vulnerabilidades que son fáciles de descubrir.

02

Estándar

Nivel recomendado para la mayoría de aplicaciones. Defensas contra la mayoría de riesgos actuales.

03

Avanzado

Recomendado para aplicaciones de misión crítica, como aplicaciones bancarias, médicas, militares, entre otras.

Ejemplos

#	Description	L1	L2	L3	CWE	NIST §
2.1.1	Verify that user set passwords are at least 12 characters in length. (C6)	✓	✓	✓	521	5.1.1.2
2.1.2	Verify that passwords 64 characters or longer are permitted. (C6)	✓	✓	✓	521	5.1.1.2

#	Description	L1	L2	L3	CWE
7.1.1	Verify that the application does not log credentials or payment details. Session tokens should only be stored in logs in an irreversible, hashed form. (C9, C10)	✓	✓	✓	532
7.1.2	Verify that the application does not log other sensitive data as defined under local privacy laws or relevant security policy. (C9)	✓	✓	✓	532
7.1.3	Verify that the application logs security relevant events including successful and failed authentication events, access control failures, deserialization failures and input validation failures. (C5, C7)		✓	✓	778
7.1.4	Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens. (C9)		✓	✓	778

¿Cómo documentarlos?

- Se pueden retomar las verificaciones del estándar aplicables y plasmarlas en historias de usuario “malvadas”, o bien en casos de “abuso”.
- La ventaja es que se cuenta con requerimientos documentados desde el punto de vista del atacante que deben ser considerados en el desarrollo y probados.

Ejemplo de Historia de usuario malvada

- Ejemplo #1. "**Como** un atacante, **quiero** enviar información errónea en las URL, **para** utilizar funcionalidades a las que no tengo permiso"
- Ejemplo #2. "**Como** un atacante, **quiero** ingresar un login y password por default **para** ganar acceso a la aplicación"

2. Aprovecha las librerías y frameworks

Librerías y Frameworks

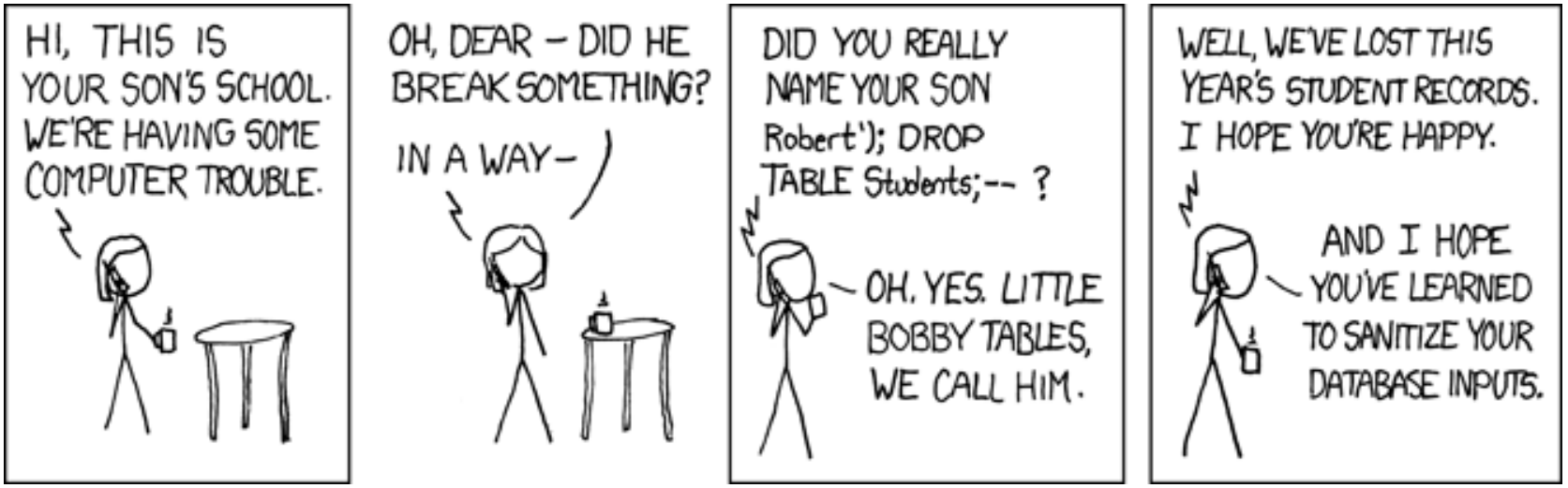
- No reinventes la rueda: usa librerías y frameworks existentes, de fuentes confiables, que cuenten con mantenimiento activo.
- Crea y mantén al día un inventario de todas las librerías que utilices.
- Mantén las librerías y componentes actualizados.
  - Utiliza herramientas como OWASP Dependency Check and Retire.JS para descubrir si alguna de tus librerías tiene vulnerabilidades conocidas.
- Reduce la superficie de ataque, encapsulando la librería y exponiendo únicamente la funcionalidad requerida en tu software.

3. Asegura el acceso a base de datos

Asegura el acceso a la base de datos

- La aplicación nunca debe conectarse a la base de datos como dueño o superusuario.
- 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.

SQL Injection



- 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.

La contraseña perfecta....

X' or '1'='1' --

✓ Mayúsculas

✓ Minúsculas

✓ Números

✓ Caracteres especiales

✓ Más de 16 caracteres

El email perfecto ....

karla'or'1'!=‘@unam.mx

✓ Cumple con RFC de email

✓ Debería pasar la validación de email válido

Ejemplo

karla'or'1'!=‘@unam.mx

```
$q = "select * from usuarios where
email='$email'";
```

```
$q = "select * from usuarios where
email=‘karla'or'1'!=‘@unam.mx'";
```

```
$q = "select * from usuarios where
email=‘karla'or'1'!=‘@unam.mx'";
```

Sentencias preparadas y parametrizadas

- La sentencia 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.

Sentencias preparadas y parametrizadas en PHP

Postgres

```
$result = pg_query_params( $dbh, 'SELECT * FROM users WHERE email = $1', array($email) );
```

Mysql

```
$stmt = $db->prepare('update people set name = ? where id = ?');
$stmt->bind_param('si',$name,$id);
$stmt->execute();
```

PDO

```
$dbh = new PDO('mysql:dbname=testdb;host=127.0.0.1', $user, $password);
$stmt = $dbh->prepare('UPDATE people SET name = :new_name WHERE id = :id');
$stmt->execute( array('new_name' => $name, 'id' => $id) );
```

4. Codifica y escapa los datos

XSS

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

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

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

Sanear datos

- Codificar caracteres
  - Reemplazar caracteres peligros con caracteres equivalentes inofensivos
  - Por ejemplo, sustituir el carácter “<“ con “<” al escribir una página HTML.
- Escapar caracteres
  - Agregar caracteres de escape antes de los caracteres peligrosos
  - Por ejemplo, agregar una diagonal invertida “\” antes de unas comillas dobles para que las interprete como texto y no como el cierre de una cadena.

Sanear datos en PHP

Función PHP	Uso
htmlspecialchars()	Convierte caracteres especiales en entidades HTML
htmlentities()	Convierte todos los caracteres aplicables a entidades HTML
urlencode()	Codifica como URL una cadena
json_encode()	Retorna la representación JSON del valor dado

NUNCA USES DATOS AQUI

```
<script> XX </script>
<!-- XX -->
<div XX=test />
<XX href="/test" />
<style> XX </style>
```

DATOS EN ELEMENTOS HTML

```
<body> XX </body>
<div> XX </div>

htmlspecialchars()
```

DATOS EN ATRIBUTOS HTML

```
<div atributo=XX />
<div atributo='XX' />
<div atributo="XX" />

htmlentities()
```

DATOS EN URLS

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

urlencode()
```

DATOS EN JAVASCRIPT

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

json_encode()
```

5. Valida todas las entradas

VALIDA TODAS LAS ENTRADAS

- Formularios
- URL
- Archivos
- Bases de datos
- Datos externos como APIs, WS, RSS

VALIDACIONES DE LISTA BLANCA

Especificar el formato y valores válidos y rechazar cualquier cosa que no cumpla

Ejemplos:
- Un ID que consta de exactamente 4 dígitos
- Una opción que solo puede ser Sí o No

VALIDACIONES EN EL SERVIDOR

- Aunque existan validaciones en el cliente, todos los datos deben validarse en el servidor.
- Diseñar un mecanismo centralizado de validaciones.
- No confiar en nada que llegue del cliente.

Mitos

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

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

6. Implementa identidad digital

Identidad digital

Autenticación

- Verificar que un individuo o entidad es quien dice ser.

Manejo de sesión

- Mantener el estado de la autenticación del usuario para que pueda continuar usando el sistema sin autenticarse nuevamente.

Autenticación con Password

- Mínimo 8 caracteres si se usa autenticación multi-factor (MFA), de lo contrario mínimo 12.
- Aceptar todos los caracteres ASCII, incluyendo espacios.
- Fomentar el uso de passwords largos o frases.
- Validar que no se utilicen passwords comunes
- https://github.com/danielmiessler/SecLists/tree/master/Passwords

Almacenar de manera segura los passwords

- Nunca almacenar las contraseñas en claro
- Utilizar algoritmos criptográficos avalados y actualizados.
- Actualmente, el algoritmo sugerido es Blowfish

```
$hash = password_hash("rasmuslerdorf“);

if (password_verify('rasmuslerdorf', $hash)) {
    echo '¡La contraseña es válida!';
} else {
    echo 'La contraseña no es válida.';
}
```

Mecanismos para recuperación de contraseñas

- Cuidar los mecanismos que se utilicen para permitir a un usuario recuperar su contraseña en caso de olvido.
- Lo recomendado es usar una combinación de MFA con secretos del usuario.
- https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Forgot_Password_Cheat_Sheet.md

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.

Proteger la sesión 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.

Identificador de la sesión

- Usar 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

Configuraciones PHP

PHP session handling

Session settings are some of the MOST important values to concentrate on in configuring. It is a good practice to change `session.name` to something new.

```
session.save_path                = /path/PHP-session/
session.name                     = myPHPSESSID
session.auto_start               = Off
session.use_trans_sid            = 0
session.cookie_domain            = full.qualified.domain.name
#session.cookie_path             = /application/path/
session.use_strict_mode          = 1
session.use_cookies              = 1
session.use_only_cookies         = 1
session.cookie_lifetime          = 14400 # 4 hours
session.cookie_secure            = 1
session.cookie_httponly          = 1
session.cookie_samesite          = Strict
session.cache_expire             = 30
session.sid_length               = 256
session.sid_bits_per_character   = 6 # PHP 7.2+
session.hash_function            = 1 # PHP 7.0-7.1
session.hash_bits_per_character  = 6 # PHP 7.0-7.1
```

7. Implementa control de acceso

Autorización

- Es el proceso 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 autenticado.

Principios de diseño de un mencanismo de control de acceso

1. Diseña el mecanismo desde el inicio
2. Todas las peticiones deben pasar por las verificaciones del control de acceso
3. Deniega por default
4. Aplica el principio de Menor privilegio
5. Evita hardcodear reglas basadas en roles
6. Almacena en bitácora todos los eventos relacionados con el control de acceso.

8. Protege los datos

Datos sensibles

- Los datos sensibles como contraseñas, números de tarjetas de crédito, registros de salud, información personal y secretos del negocio requieren protección adicional, particularmente si están protegidas por leyes de privacidad

Cifrar los datos

- Cifrar los datos en tránsito
  - TLS
- Cifrar los datos en reposo
  - Lo mejor es evitar almacenar datos sensibles
  - Buscar librerías de cifrado aprobadas
  - En PHP, vale la pena revisar Sodium, incluido por default desde 7.2

Administrar los secretos de la aplicación

- Una aplicación utiliza varios secretos:
  - Certificados
  - Contraseñas para conexiones a bases de datos
  - Credenciales para servicios externos
  - Llaves de cifrado
- Mantener esos secretos a salvos. Se pueden utilizar servicios o aplicaciones como
  - KeyWhiz
  - Vault de Hashicorp
  - Google KMS
  - AWS KMS

9. Implementa bitácoras y monitoreo de seguridad

Bitácoras y monitoreo

Las bitácoras de seguridad registran información sobre la ejecución de una aplicación.

El monitoreo es la revisión en vivo de la aplicación y de las bitácoras.

Importancia de las bitácoras de seguridad

Sin información de eventos de seguridad es difícil:

- Detectar ataques
- Detectar cuentas de usuarios comprometidas
- Detectar fraudes
- Detectar abusos de privilegios
- Responder a eventos.

Qué debe incluir la bitácora de eventos

1. Timestamp de una fuente confiable
2. Nivel de severidad
3. Identidad de la cuenta o usuario que causó el evento
4. Dirección IP asociada a la petición
5. Resultado del evento
6. Descripción del evento.
7. Etiquetado de eventos relacionados

Qué registrar

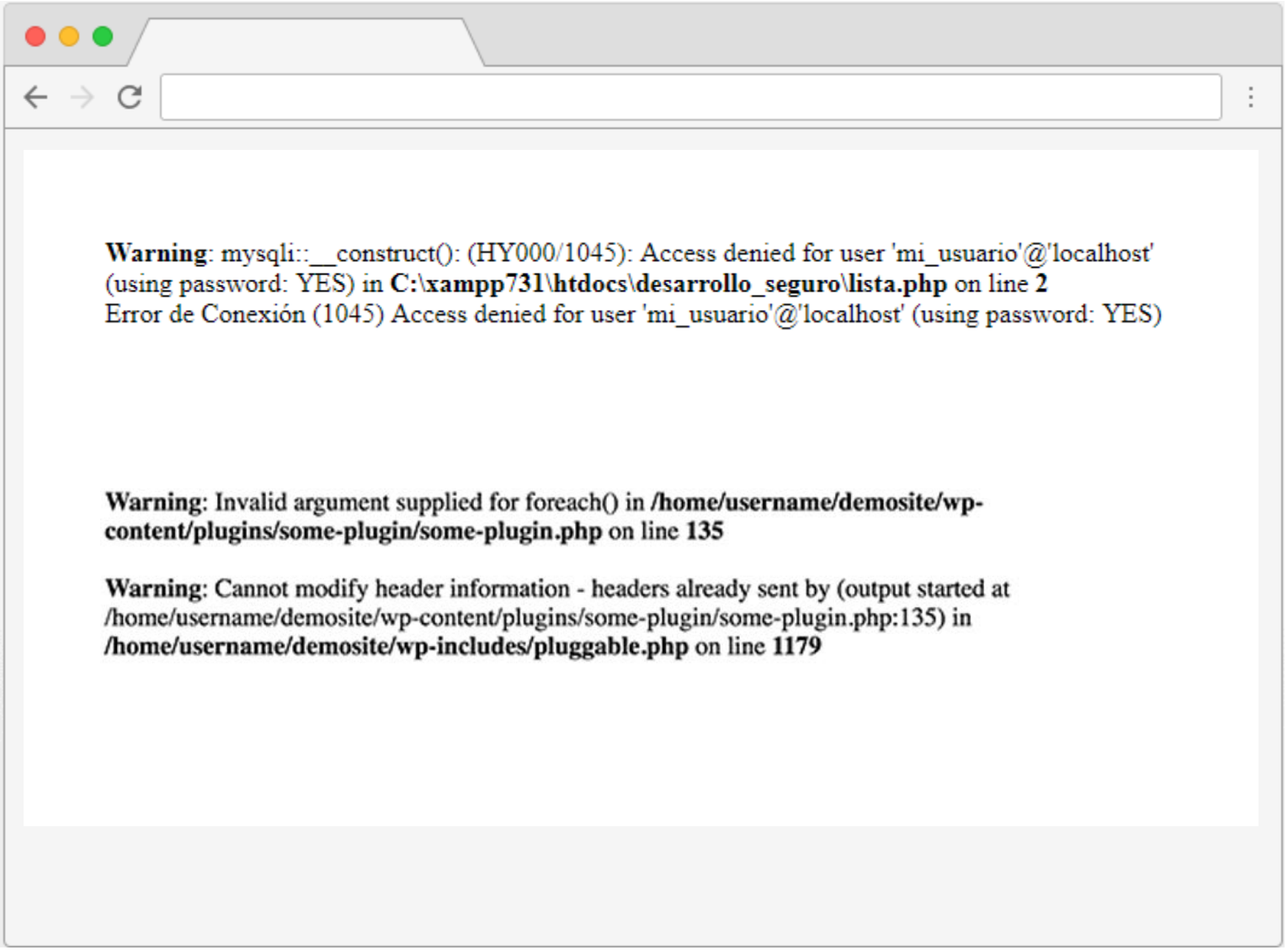
- Datos de entrada que se encuentren fuera del rango numérico
- Datos de entrada que no corresponden con los esperados (listas desplegables, checkboxes, o cualquier otro tipo de componente que restrinja los datos enviados)
- Peticiones que violen las reglas de control de acceso

- La aplicación debe registrar esos eventos en la bitácora con un nivel de severidad alto
- La aplicación debe responder en tiempo real a un posible ataque identificado, por ejemplo invalidando la sesión del usuario y bloqueando la cuenta.

Consideraciones

- Codifica y valida cualquier carácter peligroso antes de registrar datos de entrada en la bitácora.
- No registres información sensible como passwords, ID de la sesión, tarjetas de crédito, etc.
- Protege las bitácoras, considera los permisos de los archivos
- En sistemas distribuidos, centraliza las bitácoras.

10. Maneja todos los errores y excepciones



Warning: mysqli::__construct(): (HY000/1045): Access denied for user 'mi_usuario'@'localhost' (using password: YES) in **C:\xampp731\htdocs\desarrollo_seguro\lista.php** on line **2**
Error de Conexión (1045) Access denied for user 'mi_usuario'@'localhost' (using password: YES)

Warning: Invalid argument supplied for foreach() in **/home/username/demosite/wp-content/plugins/some-plugin/some-plugin.php** on line **135**

Warning: Cannot modify header information - headers already sent by (output started at /home/username/demosite/wp-content/plugins/some-plugin/some-plugin.php:135) in **/home/username/demosite/wp-includes/pluggable.php** on line **1179**

Recomendaciones

- Maneja las excepciones de manera centralizada
- Evita duplicar bloques de try/catch en el código
- Cuida que todos los comportamientos inesperados se manejen correctamente dentro de la aplicación.
- Cuida que los mensajes de error no divulguen información crítica, pero que sí sean claras para explicar el problema al usuario
- Registrar las excepciones en bitácoras, con suficiente información para que el equipo de respuesta a incidentes puedan comprender el problema.
- Considera los códigos de respuesta HTTP, en lugar de crear códigos propios.

Configuraciones PHP

PHP error handling

```
expose_php              = Off
error_reporting         = E_ALL
display_errors          = Off
display_startup_errors  = Off
log_errors              = On
error_log               = /valid_path/PHP-logs/php_error.log
ignore_repeated_errors  = Off
```

¿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).

Referencias

- OWASP Top Ten Proactive Controls 2018:
  https://www.owasp.org/images/b/bc/OWASP_Top_10_Proactive_Controls_V3.pdf
- OWASP Application Security Verification Standard 2019
  https://github.com/OWASP/ASVS/raw/master/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0-en.pdf
- Sentencias preparadas y parametrizadas:
  https://www.php.net/manual/en/pdo.prepared-statements.php
- OWASP XSS Prevention Cheat Sheet
  https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.md

Referencias

- Diccionarios de passwords más comunes:
  https://github.com/danielmiessler/SecLists/tree/master/Passwords
- Password Hashing con PHP
  https://www.php.net/manual/en/book.password.php
- OWASP Forgot Password Cheat Sheet
  https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Forgot_Password_Cheat_Sheet.md
- OWASP PHP Configuration Cheat Sheet
  https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/PHP_Configuration_Cheat_Sheet.md

¡Gracias por su atención!

¿Preguntas?