Auditoria Realizada

1. Sanitización de datos

Objetivo:
Verificar que los datos ingresados al software de gestión de usuarios sean satanizados, antes de ser ingresados en la base de datos, para evitar ataques de inyección de código SQL o XSS.

Verificar y cambiar datos de sesión

Cuando un usuario ingresa datos en el sistema, sean estos a través del login o del register, estos datos ingresados pueden ser utilizados con la intención maliciosa de manipular los datos del sistema de manera que puedan acceder, modificar o eliminar los mismos, para evitar esto se “sanitizan” los datos removiendo caracteres especiales que permiten que ataques de este estilo ocurran.

¿Cómo sanitanizar los datos ingresados?

Existen diferentes métodos para realizar la sanitización de los datos. Algunos de los más comunes son:

Prepared statements:

Prepared statements está definido en la OWASP como “una manera de escribir Queries de SQL de manera más sencilla y más fácil de entender que las consultas dinámicas.
Si las consultas a la base de datos utilizan este estilo de codificación, la base de datos siempre distinguirá entre código y datos, independientemente de lo que introduzca el usuario. Además, las sentencias preparadas garantizan que un atacante no pueda cambiar la intención de una consulta, aunque inserte comandos SQL.”

Como implementar:

La misma OWASP deja fragmentos de cómo aplicar esta metodología de queries:

```
String custname = request.getParameter("customerName");
String query = "SELECT account_balance FROM user_data WHERE user_name = ?
";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

Escaping Input:

Escaping input implica añadir una barra invertida antes de los caracteres especiales de una cadena de strings para evitar que se interpreten como parte del comando SQL.

Como implementar:

El método más utilizado para este tipo de sanitización es el uso de mysqli_real_escape_string() un ejemplo de código utilizando esta función sería:

```
$nombre = mysqli_real_escape_string($conexion, $nombre);
```

2. Prevención de ataques de fuerza bruta

Objetivo:
Implementar diversas medidas para evitar y frenar cualquier intento de ataque de fuerza bruta en el momento de ingresar los datos en el login.

Bloquear el ingreso de datos

Cuando un usuario ingresa datos, a la hora de acceder a la cuenta, el mismo puede intentar obtener acceso a cuentas de manera no autorizada, a través de los llamados “ataques de fuerza bruta” que buscan vulnerar el sistema ingresando una cuenta y intentando adivinar la contraseña de esa cuenta, esto se logra con el uso de código automático que ingresa cientos de contraseñas de manera autónoma obteniendo esas contraseñas de una lista de contraseñas posibles. Para evitar esto se tiene que bloquear el ingreso de los datos de manera que el sistema automático tarda demasiado tiempo y sea inviable para vulnerar el sistema.

¿Cómo se puede bloquear el ingreso de datos?

La acción de bloquear el ingreso no es difícil, poder distinguir quién está intentando vulnerar el sistema y quien no, puede hacerlo más difícil ya que bloquear el ingreso busca hacer que el proceso de vulneración sea más tedioso pero no queremos hacer el proceso de ingresar datos en general sea más tedioso, ya que esto afectará de manera negativa al usuario común; Para poder distinguir entre usuario y vulnerador y bloquear los datos de manera no intrusiva existen diferentes métodos como por ejemplo:

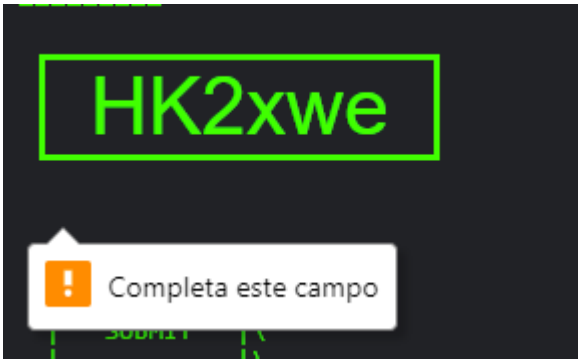
CAPTCHAs

Un CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) es un tipo de medida de seguridad que lo que hace es generar un test particular que busca distinguir a través de una acción que el usuario tiene que realizar qué usuario es humano y qué usuario es uno robot que busca vulnerar el sistema.
Este test es aleatorio y automático lo que significa es que los contenidos no son iguales cada vez que intentes hacerlo, este test va a tener que ser realizado cada vez que se intenta ingresar los datos Y si es fallado se frena el ingreso de datos y se tiene que reintentar esto evita ataques de Fuerza bruta básicos ya que la mayoría de los robots/scripts no tienen una manera de leer los captchas y llenarlos.

Como implementar

Los CAPTCHAs tienen muchas formas diferentes la que elegimos es la de una cadena de texto generada de manera automática en php su creación es bastante básica simplemente siendo una selección randomizada dentro de una cadena de caracteres y esa cadena es guardada en un $_SESSION de php.

```php
if (!isset($_SESSION['captcha'])) {
    $chars = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXTZabcdefghiklmnopqrstuvwxyz";
    $string_length = 6;
    $ChangeCaptcha = '';
    for ($i = 0; $i < $string_length; $i++) {
        $rnum = rand(min: 0, max: strlen(string: $chars) - 1);
        $ChangeCaptcha .= $chars[$rnum];
    }
    $_SESSION['captcha'] = $ChangeCaptcha;
}
```



Bloquear el login tras una cierta cantidad de intentos fallidos.

Esta solución busca, tras una cierta cantidad de intentos fallidos (en este caso, serán cinco), bloquear la sesión de login por una cantidad de tiempo que aumenta gradualmente, mientras más intentos fallidos haya, esto logrará frenar y ralentizar los intentos de ataque de fuerza bruta, al ralentizar considerablemente el proceso de su realización. Cada vez que se falla un intento, se bloquea la sesión y la herramienta va a tener que esperar 5 minutos como mínimo para volver a intentar una contraseña, lo cual alargará el tiempo de ataque de unas horas a días o semanas.

Cómo implementar

Esta solución se puede implementar de una manera simple a través de código. La manera que nosotros elegimos fue implementarla también en PHP, justamente como el CAPTCHA, ya que se hace más fácil de manejar, simplemente definiendo variables que midan la cantidad de intentos fallidos y que, al hallar una cierta cantidad, se bloquee la sesión.

```php
if ($_SESSION['attempts'] >= MAX_ATTEMPTS) {
    $_SESSION['lockout_time'] = time() + LOCKOUT_TIME;
    echo "<p>Too many failed attempts. Please try again later.</p>";
    exit;
}
```

3. Prevención de alteración de urls

Objetivo:
Prevenir el acceso a vistas de la página posteriores al login a usuarios que no han iniciado sesión o que han iniciado sesión de manera incorrecta.

Verificar el estado de la sesión

Un tipo de ataque inesperado para mucha gente es simplemente intentar cambiar la URL de la página para acceder a las páginas posteriores al login. Para prevenir esto, pensamos en una solución simple que es, verificar en todas las páginas posteriores al login si el usuario completó el login de manera exitosa.

Cómo implementar

Esta solución se puede implementar a través de código PHP al momento de realizar el login, guardando una variable llamada usuario logueado, por ejemplo. Al cargar las páginas posteriores al login, se verifica si esta variable, que comprueba si el usuario está logueado, es verdadera o falsa. Si es verdadera, el código de la página se ejecuta normalmente; si es falsa, se redirige al login.

```php
if (!isset($_SESSION['user_logged_in']) || !$_SESSION['user_logged_in']) {
    header(header: 'Location: login.php');
    exit();
}
```

4. Protección de integridad de datos

Objetivo:
Implementar métodos para la protección de datos para prevenir el acceso y/o modificación de los mismos de manera no autorizada.

Proteger los datos

Los datos almacenados en el sistema se encuentran en dos estados o estos están en “tránsito” o en “reposo”. En “tránsito” es el momento de la transferencia de datos entre dos servicios y en “reposo” es cuando los datos ya están almacenados en el sistema. En ambos momentos los datos pueden ser interceptados por gente no autorizada y adquiridos. Para evitar que los datos obtenidos sean inutilizables y inmanipulables.

¿Cómo se pueden volver inutilizables los datos?

Para lograr esto implementamos la encriptación en tránsito y en reposo:
Encriptación en tránsito se la define como una protección dada en el momento de la transferencia de datos entre dos servicios. Esta protección es lograda al encriptar los datos de manera previa a su transmisión, también autenticando los extremos, y al llegar los datos desencriptar los mismos y verificar si durante la transferencia los mismos fueron alterados.
Encriptación en reposo es una protección dada a los datos ya almacenados a través de la encriptación de los mismos, esto protege los datos contra la vulneración del sistema o los robos de datos.

Como implementar

La encriptación de datos en tránsito cuenta con numerosos métodos de encriptación open source con amplia documentación disponible pero creemos que MD5 y utilizaremos MD5 también para la encriptación en reposo.

```php
$password = md5(string: $_POST['pswd']);
```

5. Validación de Sesiones

Objetivo:
Implementar método/s para verificar que el sistema del banco esté protegido contra ataques de secuestro de sesión, CSRF, entre otras vulnerabilidades relacionadas con el manejo de sesiones.

Pruebas de Expiración de Sesiones

Las sesiones deben expirar después de un período de inactividad para evitar que un usuario permanezca autenticado indefinidamente.

Cómo implementar

Se puede implementar utilizando código de php que verifique el tiempo de sesión creando una variables que mida el tiempo de sesión transcurrida en las vistas posteriores al login.

```php
#cerrar sesion despues de 30 segundos sin actividad
$timeout_duration = 30;
if (isset($_SESSION['LAST_ACTIVITY'])) {
if (time() - $_SESSION['LAST_ACTIVITY'] > $timeout_duration) {
    session_unset();
    session_destroy();
    echo "Sesión cerrada por inactividad.";
}
```