Buenas Practicas Para Asegurar Los Datos

Buenas Practicas Para Asegurar Los Datos Sensibles Que Maneje Tu Aplicación

Al momento de desarrollar una aplicación, es muy importante tener en cuenta el manejo de datos sensibles del usuario. En este articulo nos centraremos completamente en la tercera capa de seguridad, la **seguridad a nivel de la aplicación**. Encontrarás alguna técnicas que puedes utilizar para proteger la aplicación de ataques, y así asegurar los datos sensibles que manejes en la aplicación que desarrollas.

¿Que Son Los Datos Sensibles?

«… lo más común es usar este término para designar datos privados relacionados con Internet o la informática, sobre todo contraseñas, tanto de correo electrónico, conexión a Internet, IP privada, sesiones del PC, etc.» es una de las tantas definiciones y tal vez la mas cercana, la encuentras en

la Wikipedia. Para acercarlo a la realidad, serian los datos de conexión ó aquellos datos que puedan ser ingresados por parte del usuario, lo que provendría del inicio de sesión o registro.

Usando Una Base De Datos

Cuando se comunica la aplicación con una base de datos y con el fin de que sus datos permanecerán seguros, ponga en practica los siguientes consejos para **mejorar la seguridad cuando se usa una base de datos**:

Siempre Escapar Las Consultas

Para detener los ataques que usan la técnica de inyección SQL, usted tiene que escapar todo ingreso de datos malintencionado por un usuario en la aplicación (por ejemplo, durante un inicio de sesión). En casi todos los controladores de base de datos, para todas los lenguajes, existe una opción para escapar desde la entrada de datos por el susuario. Por ejemplo, para Node.js esta la utilidad node-mysql, en vez de realizar una sentencia como la siguiente:

[sql]connection.query('SELECT * FROM users WHERE name = \»+ username +'\' AND password = \»+ password '\';', function (err, rows, fields) {[/sql]

Puede escapar de forma automática utilizando la siguiente

sintaxis:

[sql]connection.query('SELECT * FROM users WHERE name = ? AND password = ?;', [username, password], function (err, rows, fields) { [/sql]

Con la extensión PDO podemos preparar la sentencia, con lo cual podemos realizar una sentencia «preliminar» en la cual no usaremos los datos directamente, sino mas bien, añadiremos posteriormente dato por dato:

[php]

$stmt = $dbh->prepare("SELECT * FROM users WHERE name = ? AND password = ?;");
$stmt->bindParam(1, $username);
$stmt->bindParam(2, $password);

$stmt->execute();

[/php]

Nunca Confíe Del Contenido De La Base De Datos

Incluso si usted escapa datos que puedan generar problemas en sus consultas de las bases de datos, nunca se sabe si es un bug en su aplicación ha permitido algunos códigos maliciosos a través de la misma base de datos. Si ha desarrollado su aplicación de una manera de la que siempre asume que el contenido de la base de datos es segura, sus usuarios pueden estár en peligro. Por ejemplo, el siguiente código:

[php]

```
if ($result = $mysqli-&gt;query('SELECT * FROM users WHERE name = "'. $mysqli-&gt;escape_string($username) .'";')) {
if ($row = $result-&gt;fetch_assoc()) {
if ($result = $mysqli-&gt;query('SELECT * FROM someData WHERE uid = '. $row['id'] .';')) {
…
}
}
}
```

[/php]

Adivina, el siguiente código es peligroso! Si algo sale mal y el id del usuario contiene un código de inyección SQL, entonces podría tener problemas. En $row['id'] también se podría emplear la **función escape_string()** para asegurarnos de evitar un ataque de inyección SQL:

[php]

```
if ($result = $mysqli-&gt;query('SELECT * FROM users WHERE name = "'. $mysqli-&gt;escape_string($username) .'";')) {
if ($row = $result-&gt;fetch_assoc()) {
if ($result = $mysqli-&gt;query('SELECT * FROM someData WHERE uid = '. $mysqli-&gt;escape_string($row['id']) .';')) {
…
}
}
}
```

[/php]

Usando Técnica Salt Para Las Contraseñas

Ahora imagine esta situación: algunos hackers obtienen datos de la base de datos cuidadosamente. Ellos usan el ataque de fuerza bruta para tener todas las contraseñas de los usuarios. Si ha usado la técnica Salt al momento de hashear la contraseña, puede estar tranquilo y asegurar a los usuarios que sus datos están seguros. Porque a menos que el atacante tenga un ordenador cuántico, les llevará años para obtener cualquiera de las contraseñas.

El uso de la técnica Salt significa que tiene que anexar algunos caracteres aleatorios para la contraseña antes de hashearla y almacenar la contraseña. Dado que el carácter Salt es diferente para cada usuario, aunque dos de ellos usen la misma contraseña, sus hashes serán diferentes. Esto también significa que el atacante no podrá usar diccionarios, ni otra clase de archivos que contengan contraseñas previamente reunidas.

Usar Técnica Salt En

Node.js

En primer lugar se debe generar el carácter Salt. Por ejemplo, podría utilizar la función crypto.randomBytes():

[js]

var crypto = require('crypto');

/ *
esto se debe hacer, a petición de los datos del usuario para el registro,
en lugar de limitarse a añadir a la base de datos
* /

crypto.randomBytes(16, function (e, salt) {

[/js]

Esta función lanza un error si, de acuerdo con la documentación de Node.js, «no hay suficiente entropía para generar datos que sean criptográficamente fuertes». En tal caso, debe intentarlo de nuevo ó utilice crypto.pseudoRandomBytes(), que genera bytes aleatorios no criptográficamente fuertes, pero ya que esto sólo ocurrirá de vez en cuando, se puede utilizar:

[js]

if (e) {
salt = crypto.pseudoRandomBytes(16);
}

[/js]

Ahora vamos a agregar los datos. Para simplificar, sólo se va a usar un nombre de usuario, contraseña y el carácter Salt en este ejemplo:

[js]

```
/* password and username should be extracted from request body */
var hash = crypto.createHash('sha-256').update(password + salt.toString()).digest('hex');
sql.query('INSERT INTO users VALUES(?, ?, ?);', [ username, password, salt ], function (err, rows, fields) {
/* your other app logic here, what happes after the user data is into the database */
});
});
```

[/js]

Cuando el usuario desee iniciar sesión, tiene que conseguir el Salt de la base de datos y comprobar la contraseña:

[js]

```
sql.query('SELECT salt FROM users WHERE name = ?;', [ username ], function (err, rows, fields) {
if (rows.length &lt; 1) {
/* no existe el usuario */
} else {
var hash = crypto.createHash('sha-256').update(password + rows[0].salt).digest('hex');
sql.query('SELECT 1 FROM users WHERE name = ? AND password = ?;', [ username, hash ], function (err, rows, fields) {
if (rows.length &lt; 1) {
/* contraseña equivocada */
} else {
/* si la contraseña coincide */
}
});
}
});
```

[/js]

Y eso es prácticamente todo. Ahora vamos a ver cómo podemos hacer esto utilizando PHP.

Usar Técnica Salt En PHP

En PHP es mucho más simple, porque se pueden usar las funciones password_hash() y password_verify(), ya que la primera función añade el carácter Salt y la otra lo retorna; por lo que ni siquiera necesita otra columna en la base de datos (o de campo en caso de NoSQL) . Para usarlo sería algo parecido a lo siguiente:

[php]

```
$stmt = $dbh-&gt;prepare("INSERT INTO users VALUES (?, ?);");
$stmt-&gt;bindParam(1, $username);
/* hash the password, the function adds random salt automatically */
$stmt-&gt;bindParam(2, password_hash($password));

$stmt-&gt;execute();
```

[/php]

Para comprobar el inicio de sesión, sólo tienes que usar la función password_verify(). Automáticamente obtiene el carácter Salt a partir del hash por lo que sólo necesita proporcionar el hash de su base de datos y la contraseña en texto plano para comparar. El código necesario para esto seria el siguiente:

[php]

```
$stmt = $dbh-&gt;prepare("SELECT password FROM users WHERE name = ?;");
$stmt-&gt;bindParam(1, $username);
```

```
$stmt-&gt;execute();

$row = $stmt-&gt;fetch(PDO::FETCH_ASSOC);

if ($row) {
/* verifica la busqueda*/
if (password_verify($password, $row[‘password’])) {
/*si la contraseña coincide*/
}
} else {
/* si el usuario no existe */
}

[/php]
```

Eliminar Privilegios Cuando No Son Necesarios

Este es uno de los mejores mecanismos de defensa que puede utilizar para proteger su servidor y por lo tanto los archivos y usuarios. Es verdad que es necesario contar con privilegios elevados durante un par de cosas, pero **nunca se debe ejecutar cualquier aplicación con privilegios root** cuando no es necesario, en caso de que el atacante encuentre un error en el código que les permite ejecutar comandos en el servidor y si

el código se ejecuta como un usuario con privilegios, solo sabremos algo: **Game Over!** El atacante puede hacer lo que quiera y probablemente será hecho incluso antes de que pueda notarlo. Es por eso que debe disminuir los alcances de los privilegios. En **Node.js** se vería como esto:

[js]

var app = http.createServer(…);

…

app.listen(80); // Escucha en el puerto 80, para hacer lo que tiene que ejecutar su aplicación como usuario root
process.setuid('app_user'); // Cambia los privilegios de un usuario que no sea root – sólo para plataformas POSIX

[/js]

La función **process.setuid()** va a cambiar la identidad de usuario del proceso a la que se le ha pasado, puede ser o bien un identificador numérico o una cadena de nombre de usuario (en el segundo caso, esta función bloqueará al obtener el ID del usuario). El usuario debe tener privilegios y solamente tiene acceso a los archivos de la aplicación específica, para limitar el riesgo de dar al atacante acceso a cualquier otra cosa en su máquina.

Una Alternativa: Usar authbind

Algunas personas dicen, que la solución anterior no es perfecta y que prefieren utilizar authbind en su lugar. Pero de todos modos, authbind es un comando que fue diseñado para este propósito, el cual permite a su aplicación enlazar a los

puertos inferiores a 1024 sin privilegios de root (por lo que sólo está cubriendo ese escenario). Para usarlo, primero debe crear un archivo: **/etc/authbind/byport/port**, donde port es el número de puerto que desea añadir y hacerlo ejecutable por el usuario que va a utilizar, para ejecutar su aplicación. Luego cambiar a su usuario e iniciar la aplicación, justo como lo siguiente:

[js]authbind node yourapp.js[/js]

O como lo siguiente desde la raíz:

[js]su -c ‘authbind node yourapp.js’ youruser[/js]

Con esto, puede lograr el mismo objetivo que con la solución de POSIX, ahora usando authbind lugar, si eso es lo que prefieres.

Finalmente

Esperamos que hayas aprendido algunas nuevas técnicas para trabajar con los datos sensibles que maneje tu aplicación. En el desarrollo que tus aplicaciones, tanto de escritorio, como aplicaciones web ó de dispositivos móviles, ¿Que practicas de seguridad usas al manejar los datos?