Documento de Seguridad

SQL Injection , Autenticación, autorización y seguridad.

En el presente documento brindamos las practicas de seguridad que utilizamos (algunas inherentes al uso correcto del framework en cuestión):

Uso de vinculación de parámetros antes de ejecutar sentencia SQL.

Para evitar ataques de SQL Injection y para mejorar el rendimiento de sentencias SQL usadas repetidas veces, podemos "preparar" una sentencia SQL con marcadores de posición de parámetros opcionales, que son marcadores que serán reemplazados con los parámetros reales durante el proceso de vinculación de parámetros. El driver subyacente de la base de datos lo hará por nosotros. La vinculación de parámetros debe hacerse antes de que la sentencia SQL sea ejecutada.

```
// una SQL con dos marcadore de posición, ":username" and ":email"
$sql="INSERT INTO users(username, email) VALUES(:username,:email)";
$command=$connection->createCommand($sql);
// reemplaza el marcador de posición ":username" con el valor real de username
$command->bindParam(":username",$username,PDO::PARAM_STR);
// reemplaza el marcador de posición ":email" con el valor real de email
$command->bindParam(":email",$email,PDO::PARAM_STR);
$command->execute();
// inserta otra fila con un nuevo conjunto de parámetros
$command->bindParam(":username",$username2,PDO::PARAM_STR);
$command->bindParam(":email",$email2,PDO::PARAM_STR);
$command->execute();
```

Los métodos bindParam() y bindValue() son muy similares. La única diferencia es que el primero vincula un parámetro con una variable PHP mientras que el último con un valor. Para parámetros que representan grandes bloques de memoria de datos, es preferible el primero por consideraciones de rendimiento.

AUTENTICACION

Autenticación es la acción de verificar quien es el usuario, ergo , la base del proceso de login. Típicamente, la autenticación usa la combinación del identificador – un nombre de usuario o dirección de email- y el password. El usuario envía estos valores mediante una forma y la aplicación compara la información enviada con la anteriormente guardada(e.j.,

registración).
En Yii, este proceso es realizado casi automáticamente dejando al desarrollador implementar yii\web\IdentityInterface, la clase más importante en el sistema de autenticación. Típicamente , la implementación de IdentityInterface es lograda usando el User model.
Una versión más completa se encuentra en el código fuente.
Abajo, solo los métodos de la interfaz, están siendo listados.

```
/class User extends ActiveRecord implements IdentityInterface
{
// ...
        /**
        * Encuentra una identidad dado un ID.
        *
        * @param string|integer $id El ID a ser buscado
        * @return IdentityInterface|null El objeto identidad matchea el  ID.
        */

        public static function findIdentity($id)
        {
                return static::findOne($id);
        }
        /**
        * Encuentra una identidad dado un ID.
        *

        public static function findIdentityByAccessToken($token, $type = null)
        {
                return static::findOne([’access_token’ => $token]);
        }
                /**
                * @return int|string current user ID
                */

        public function getId()
        {
                return $this->id;
        }

                /**
                * @return la clave del usuario
                */

        public function getAuthKey()
        {
                return $this->auth_key;
        }

                /**
                * @param string $authKey
                * @return boolean si la clave de autenticación es correcta o no.
                */
```

```
        public function validateAuthKey($authKey)
        {
        return $this->getAuthKey() === $authKey;
        }
}
```

El método getAuthKey debería retornar un string que debería único para cada usuario . Para poder crear un “unique string” confiable se debe usar el Yii::$app->getSecurity ()->generateRandomString() . Por ello debería ser parte del registro del usuario:

```
public function beforeSave($insert)
{
        if (parent::beforeSave($insert)) {
                if ($this->isNewRecord) {
                        $this->auth_key = Yii::$app->getSecurity()->generateRandomString
                        ();
                }
        return true;
        }
return false;
}
```

El método validateAuthKey solo se compara con la variable $authKey, pasada como parámetro (la misma desde una cookie), con el valor adquirido desde la base de datos.

AUTORIZACION

Autorización es el proceso de verificar que el usuario tiene suficientes permisos para hacer algo. Yii provee dos métodos de autorización : Access Control Filter (ACF) and Role-Based Access Control (RBAC).

Access Control Filter (ACF) es un método de autorización que es útil para cuando las aplicaciones solo requieren acceso de control simple. Es un filtro que puede ser albergado en un controlador (controller), o en un modulo como comportamiento. ACF va a chequear en yii\filters\AccessControl ::rules para verificar que el usuario actual puede acceder a la dirección solicitada.

EJ :

```
use yii\filters\AccessControl;
        class SiteController extends Controller
        {
        public function behaviors()
        {
        return [
                ’access’ => [
                ’class’ => AccessControl::className(),
                ’only’ => [’login’, ’logout’, ’signup’],
                        ’rules’ => [
                                [
```

```
                    'allow' => true,
                    'actions' => ['login', 'signup'],
                    'roles' => ['?'],
                    ],
                    [
                    'allow' => true,
                    'actions' => ['logout'],
                    'roles' => ['@'],
                    ],
                ],
            ],
        ];
        }
        // ...
}
```

En el código de arriba el ACF es atacheado al controlador del sitio como comportamiento. Esta es la forma de usar un "action filter".
El ACF se aplica a las acciones de login, logout. Las reglas de opción especifican que el yii\filters\AccessRule, se puede leer como sigue:
• Permitir a todos los usuarios invitados (aun no autenticados) acceder a las acciones de 'login' y 'signup' . Los roles de opciones contienen un "?" el cual es un token especial para los reconocidos como usuarios "guests".
• Permitir a los usuarios autenticados que accedan a la acción 'logout'. El carácter @ es un token especial reconocido como usuarios autenticados.

Cuando ACF realiza un check de autorización, el cual va a examinar las reglas una por una desde abajo hacia arriba hasta que encuentra una con la cual coincide. Esto permite a el valor de la regla de matcheo ser usuado para juzgar si un usuario esta autorizado. (Pattern matching) Si ninguna de las reglas matchea, significa que el usuario no esta autorizado y que el ACF va a parar la ejecucion de futuras acciones.

Por default, ACF solo hace las siguientes cosas cuando determina que un usuarios no esta autorizado para acceder a la acción actual:
• Si el usuario es invitado, va a llamar a yii\web\User::loginRequired(), el cual va a redirigir al browser a la pagina de login.
• Si el usuario esta autenticado, va a lanzar un yii\web\ForbiddenHttpException.

Role based access control (RBAC)

Role-Based Access Control (RBAC) provee un acceso de control centralizado.
Yii implementa un RBAC general de forma jerárquica, siguiendo el modelo NIST RBAC .

Provee la funcionalidad RBAC mediante el componente yii\rbac\ManagerInterface
Usar RBAC involucra dos partes de trabajo. La 1er parte es construir los datos de autorización de RBAC y la 2da parte es utilizar la data de autorización realizar chequeos de acceso en lugares donde es necesario.

Un rol representa un conjunto de permisos (por ejemplo, la creación de posts , la actualización de posts). Un rol puede ser asignado a uno o varios usuarios. Para comprobar si un usuario tiene un permiso especificado, podemos comprobar si el usuario se le asigna un rol que contiene ese permiso.

Asociado con cada rol o el permiso, puede haber una regla. Una regla representa un fragmento de código que se ejecutará durante la comprobación de acceso para determinar si el rol o el permiso correspondiente se aplica al usuario actual.
Por ejemplo, el permiso "actualización de post" puede tener una regla que comprueba si el usuario actual es el creador del post. Durante la comprobación de acceso, si el usuario no es el creador del post, él / ella será considerado no tener tal permiso.
Ambos roles y permisos pueden ser organizados en una jerarquía. En particular, un papel puede consistir en otros roles o permisos; y un permiso puede consistir de otros permisos.

Yii implementa una jerarquía de orden parcial que incluye la jerarquía más especial árbol. Mientras que un papel puede contener un permiso, no es verdad viceversa.

SEGURIDAD

Una buena seguridad es vital para la salud y el éxito de cualquier aplicación. Desafortunadamente, muchos desarrolladores toman atajos cuando se trata de seguridad, ya sea debido a una falta de comprensión o porque la implementación es demasiado de un obstáculo.
Yii incluye los siguientes features de seguridad:

Hashing y verificación de contraseñas
La mayoría de los desarrolladores saben que las contraseñas no se pueden almacenar en texto plano, pero muchos desarrolladores creen que todavía es seguro para cifrar las contraseñas utilizando MD5 o SHA1.
Hubo un momento en que usa los algoritmos hash antes mencionados eran suficientes, cosa que no podemos acertar actualmente.
La mejor opción actual es **bcrypt**. En PHP, puede crear un hash bcrypt utilizando la función “**cript**”. Yii proporciona dos funciones auxiliares que facilitan el uso de la **cript** para generar y verificar hashes de forma segura ,más fácil.
Cuando un usuario proporciona una contraseña para la primera vez la contraseña debe ser hash:

```
$hash = Yii::$app->getSecurity()->generatePasswordHash($password);
```

El hash puede asociarse con el correspondiente atributo modelo, por lo que puede ser almacenado en la base de datos para su uso posterior.
Cuando un usuario intenta iniciar sesión, la contraseña presentado debe ser verificada contra la contraseña previamente hasheada y almacenada:

```
if (Yii::$app->getSecurity()->validatePassword($password, $hash)) {
// Todo Ok, el usuario esta logueado
} else {
// Password erronea
}
```

Generando datos pseudoaleatorios

Los datos pseudoaleatorios son útiles en muchas situaciones. Por ejemplo cuando se repone una contraseña por correo electrónico que usted necesita para generar un token, guardarlo en la base de datos, y enviarlo por correo electrónico a usuario final, que a su

vez les permitirá demostrar autoridad sobre esa cuenta. Es muy importante que esta señal (token) sea única y difícil de adivinar, de lo contrario hay una posibilidad de que un atacante puede predecir el valor del token y restablecer la contraseña del usuario.
Yii colabora generando datos pseudoaleatoria:

```
$key = Yii::$app->getSecurity()->generateRandomString();
```

Hay que tener una extensión openssl instalada para poder generar criptograficamente datos aleatorios seguros.

El cifrado y descifrado

Utilizando una clave secreta, Yii nos permite que los datos se pasan a través de la función de cifrado de modo que sólo la persona que tiene la clave secreta será capaz de descifrarlo. Por ejemplo, tenemos que guardar alguna información en nuestra base de datos, sino que necesitamos para asegurarse de que sólo el usuario que tiene la clave secreta puede visualizarla (incluso si el base de datos de la aplicación se ve comprometida):

```
// $data y $secretKey son obtenidos desde la form a

$encryptedData = Yii::$app->getSecurity()->encrypt($data, $secretKey);

// store $encryptedData a la base de datos subsecuentemente cuando el usuario quiere leer la data :
// $secretKey es obtenidad del input del usuario, $encryptedData es de la base de datos

$data = Yii::$app->getSecurity()->decrypt($encryptedData, $secretKey);
```

Confirmación de la integridad de datos
Hay situaciones en las que necesita para verificar que sus datos no han sido manipulado por un tercero o incluso dañado de alguna manera. Yii ofrece una manera fácil para confirmar la integridad de los datos en forma de dos funciones auxiliares.
El prefijo de datos con un hash generado a partir de la clave secreta y datos:

```
// $secretKey nuestra aplicacion o usuario secreto, $genuineData  obtenidos desde una fuente confiable.

$data = Yii::$app->getSecurity()->hashData($genuineData, $secretKey);
```

Cosas que no provee el framework de forma “nativa” :

XSS prevention, CSRF prevention, cookie protection.