$users = User::find()->where([$attr => $value])->all();

return $this->renderContent(Html::ul(

ArrayHelper::getColumn($users, 'username')

));

}

12. But the most secure way is to use the whitelist approach, as follows:

public function actionWhiteList()

{

$attr = Yii::$app->request->get('attr');

$value = Yii::$app->request->get('value');

$allowedAttr = ['username', 'id'];

if (!in\_array($attr, $allowedAttr)) {

throw new Exception("Attribute specified is not allowed.");

}

$users = User::find()->where([$attr => $value])->all();

return $this->renderContent(Html::ul(

ArrayHelper::getColumn($users, 'username')

));

}

How it works...

The main goal when preventing SQL injection is to properly filter the input. In all cases except table names, we have used prepared statements—a feature supported by most relational database servers. They allows you to build statements once and then use them multiple times, and they provide a safe way of binding parameter values.

In Yii, you can use prepared statements for both Active Record and DAO. When using DAO, it can be achieved by using either bindValue or bindParam. The latter is useful when we want to execute multiple queries of the same type while varying parameter values:

public function actionBind()

{

$userName = 'Alex';

$passwordHash = md5('password1');

$sql = "INSERT INTO 'user' ('username', 'password') VALUES (:username, :password);"; // insert first user

$command = Yii::$app->db->createCommand($sql);

$command->bindParam('username', $userName);

$command->bindParam('password', $passwordHash);

$command->execute();

// insert second user $userName = 'Qiang';

$passwordHash = md5('password2');

$command->execute();

return $this->renderContent(Html::ul(

ArrayHelper::getColumn(User::find()->all(), 'username')