Предотвращение SQL-инъекций

SQL injection-это тип внедрения кода, который использует уязвимость на уровне базы данных и позволяет выполнять произвольный SQL, позволяя злоумышленникам выполнять такие действия, как удаление данных или привилегии.

В этом рецепте, мы увидим примеры уязвимого кода и исправим их.

Подготовка

1. Создайте новое приложение с помощью диспетчера пакетов Composer, как описано в официальном руководстве по адресу  
   <http://www.yiiframework.com/doc-2.0/guide-start-installation.html>.   
   По русски <http://yiiframework.domain-na.me/doc/guide/2.0/ru/start-installation>**.**
2. Выполните следующий SQL:

DROP TABLE IF EXISTS 'user';

CREATE TABLE 'user' (

'id' int(11) unsigned NOT NULL AUTO\_INCREMENT,

'username' varchar(100) NOT NULL,

'password' varchar(32) NOT NULL,

PRIMARY KEY ('id')

);

INSERT INTO 'user'('id','username','password') VALUES (

'1','Alex','202cb962ac59075b964b07152d234b70');

INSERT INTO 'user'('id','username','password') VALUES (

'2','Qiang','202cb962ac59075b964b07152d234b70');

1. Создайте пользовательскую модель с помощью Gii.

Как это сделать...

1. Во-первых, мы реализуем простое действие, которое проверяет правильность имени пользователя и пароля, которые пришли из URL. Создайте app/controllers/Sqlcontroller.php:

<?php

namespace app\controllers;

use app\models\User;  
use Yii;

use yii\base\Controller;  
use yii\base\Exception;  
use yii\helpers\ArrayHelper;  
use yii\helpers\Html;

/\*\*

* Class SqlController.
* @package app\controllers  
  \*/

class SqlController extends Controller  
{

protected function renderContentByResult($result)

{

if ($result) {

$content = "Success";

} else {

$content = "Failure";

}

return $this->renderContent($content);

}

public function actionSimple()

{

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

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

$passwordHash = md5($password);

$sql = "SELECT \* FROM 'user'"

." WHERE 'username' = '".$userName."'"

." AND password = '".$passwordHash."' LIMIT |1";

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

return $this->renderContentByResult($result);

}

}

1. Давайте попробуем получить к нему доступ URL /sql/simple?username=test&password=test. Поскольку мы не знаем имя пользователя, так и пароль, это будет, как и ожидалось, сбой.
2. Теперь попробуйте/sql/simple?username=%27+or+%271%27%3D%271%27%3B+--&password=whatever. На этот раз он позволяет нам войти, хотя мы все еще ничего не знаем о фактических полномочиях. Расшифрованная часть usernamevalue выглядит следующим образом:

' or '1'='1'; --

1. Закройте цитату, чтобы синтаксис оставался правильным. Добавьте или ' I ' = 'I', что делает условие всегда истинным. Используйте;--, чтобы завершить запрос и прокомментировать остальные.
2. Поскольку экранирование не было выполнено, был выполнен весь запрос:

SELECT \* FROM user WHERE username = '' or '1'='1'; --' AND password =

'008c5926ca861023c1d2a36653fd88e2' LIMIT 1;

1. Лучший способ исправить это - использовать подготовленный оператор следующим образом:

public function actionPrepared()

{

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

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

$passwordHash = md5($password);

$sql = "SELECT \* FROM 'user'"

." WHERE 'username' = :username"

." AND password = :password LIMIT 1";

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

$command->bindValue(':username', $userName);

$command->bindValue(':password', $passwordHash);

$result = $command->queryOne();

return $this->renderContentByResult($result);

}

1. Теперь проверьте /sql/prepared с теми же вредоносными параметрами. На этот раз все было хорошо, и мы получили сообщение об ошибке. Тот же принцип применяется и к ActiveRecord. Единственное отличие здесь в том, что AR использует другой синтаксис:

public function actionAr()

{

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

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

$passwordHash = md5($password);

$result = User::findOne([

'username' => $userName,

'password' => $passwordHash

]);

return $this->renderContentByResult($result);

}

1. В предыдущем коде мы использовали параметры username и password как ключ массива со стилем значения. Если бы мы написали предыдущий код, используя только первый аргумент, он был бы уязвим:

public function actionWrongAr()

{

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

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

$passwordHash = md5($password);

$condition = "'username' = '".$userName." AND 'password' = '".$passwordHash."'";  
$result = User::find()->where($condition)->one();  
return $this->renderContentByResult($result);

}

1. При правильном использовании подготовленные операторы могут избавить вас от всех типов SQL-инъекций. Тем не менее, есть некоторые общие проблемы:

° Вы можете привязать только одно значение к одному параметру, поэтому, если вы хотите(1, 2, 3, 4), вам придется создать и привязать четыре параметра.

° Подготовленные операторы нельзя использовать для имен таблиц, столбцов и других ключевых слов.

1. При использовании ActiveRecord, первая проблема может быть решена путем добавления, где, как следует:

public function actionIn()

{

$names = ['Alex', 'Qiang'];

$users = User::find()->where(['username' => $names])->all();

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

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

));

}

1. Вторая проблема может быть решена несколькими способами. Первый способ-полагаться на активную запись и PDO quoting:

public function actionColumn()

{

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

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

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

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

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

));

}

12. Но самый безопасный способ - использовать подход белого списка следующим образом:

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')

));

}

Как это работает...

Главная цель для предотвращения SQL-инъекции-это правильный фильтр на входе. Во всех случаях, кроме имен таблиц, мы использовали подготовленные операторы—функцию, поддерживаемую большинством серверов реляционных баз данных.

Они позволяют создавать операторы один раз, а затем использовать их несколько раз, и они обеспечивают безопасный способ привязки значений параметров.

В Yii можно использовать подготовленные операторы как для Active Record, так и для DAO. При использовании DAO, это может быть достигнуто с помощью bindValue или bindParam. Последнее полезно, когда мы хотим выполнить несколько запросов одного типа при изменении значений параметров:

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')