we received the Failure message. The same principle applies to ActiveRecord. The only difference here is that AR uses other syntax:

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. In the previous code, we used the username and password parameters like an array key with a value style. If we had written the previous code by using only the first argument, it would be vulnerable:

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. If used properly, prepared statements can save you from all types of SQL injections. Still, there are some common problems:

° You can only bind one value to a single parameter, so if you want to query where in(1, 2, 3, 4), you will have to create and bind four parameters.

° Prepared statements cannot be used for table names, column names, and other keywords.

1. When using ActiveRecord, the first problem can be solved by adding where, as follows:

public function actionIn()

{

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

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

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

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

));

}

1. The second problem can be solved in multiple ways. The first way is to rely on active record and PDO quoting:

public function actionColumn()

{

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

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