Curso POO PHP Inxección SQL

Sumario

Inxección SQL

Un dos problemas de seguridade máis habituais é a vulnerabilidade fronte a inxeccións SQL. Unha **inxección SQL** consiste en introducir xunto cos datos (por exemplo nos formularios) código que ten por obxecto modificar as consultas que se realizan sobre a base de datos.

Por exemplo, vexamos o seguinte código PHP, que recibe o nome e o contrasinal introducidos nun formulario HTML de *login* (nótese que é un exemplo ficticio; na realidade é recomendable empregar unha función de hash como md5 para almacenar as contrasinais dos usuarios):

```
$consulta = "SELECT * FROM usuarios WHERE nome='" . $_REQUEST["nome"] . "' AND contrasinal='" . $_REQUEST["contrasinal"] . "'";
$resultado = $db->query($consulta);
if($resultado) {
   $usuario = $resultado->fetch_array(); // Obtemos o primeiro rexistro
   $nome = $usuario['nome'];
   $contrasinal = $usuario['contrasinal'];
   ...
}
```

A función do código anterior é comprobar se existe na base de datos un usuario con ese nome e contrasinal. Se existe, suponse que é único e colle os datos do primeiro rexistro devolto pola consulta.

Se o usuario introducise o seguinte texto no campo "contrasinal" do formulario anterior: "' **OR 1;**" (notar a primeira comiña simple de peche no texto), a consulta que executaría PHP sería a seguinte e devolvería a lista completa de usuarios:

```
SELECT * FROM usuarios WHERE nome='xxx' AND contrasinal='' OR 1;'
```

Isto é, daríamos ao usuario por válido coa identidade do primeiro usuario da táboa.

Imos ver distintos métodos que podemos aplicar nos nosos scripts PHP para evitar a inxección SQL.

Empregar consultas preparadas

As consultas preparadas non se ven afectadas polas técnicas de inxección SQL. Os valores que reciben dos parámetros non se poden extender a outras partes da consulta.

```
$consulta = $db->prepare('SELECT * FROM usuarios WHERE nome=:nome AND contrasinal=:contrasinal');
$consulta->bind_param(":nome", $_REQUEST["nome"]);
$consulta->bind_param(":contrasinal", $_REQUEST["contrasinal"]);
$consulta->execute();
if($rexistro = $consulta->fetch(PDO::FETCH_OBJ)) {
   $nome = $rexistro->nome;
   $contrasinal = $rexistro->contrasinal;
   ...
}
```

O problema das consultas preparadas é que os parámetros non se poden empregar en algúns casos, por exemplo:

- Para substituir listas de valores (como na clausula IN de SQL).
- Para facer referencia ao nome dunha táboa ou dun campo da táboa.
- Como parte da sintaxe da sentenza SQL (por exemplo, para facer un UPDATE ou un DELETE en función da entrada do usuario).

Se é o caso, teremos que empregar consultas interpoladas e apoiarnos noutras técnicas para protexernos fronte a inxección SQL.

Validar os datos introducidos polos usuarios

Consiste en comprobar que os datos que introducen os usuarios se axustan aos requisitos necesarios para o campo concreto. Por exemplo, que o valor da idade dunha persoa é numérico positivo, ou que a dirección de correo electrónico é válida.

A validación pode facerse de varias formas:

- Comprobando o tipo de datos coas funcións **is_int**, **is_string**, **is_bool**, ..., ou con **gettype**.

```
$idade = ($_REQUEST["idade"];
if(!is_int($idade)) exit("$idade é un valor inválido para a idade!");
```

  Ou directamente forzando o tipo de datos.

```
$idade = (int) $_REQUEST["idade"];
```

- Comprobando a lonxitude dos datos introducidos, para evitar desbordamento de buffer e ataques DOS.

```
$nome = (int) $_REQUEST["nome"];
if (strlen($nome) > 255) exit("O nome NON é válido");
```

- Empregando expresións regulares.

```
$expreg = '/^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,3})$/';
if (!preg_match($expreg, $email)) exit("A dirección de email NON é válida");
```

- Empregando a extensión **Filter**, que ven activa por defecto dende PHP 5.2, xunto cun filtro de validación axeitado.

```
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) exit("A dirección de email NON é válida");
```

Sanear os datos introducidos polos usuarios

A validación das entradas dos usuarios é un procedimento que convén levar a cabo de forma xeralizada para calquera información nova que deba introducir o usuario. O saneamento dos datos consiste en verificar que é seguro a súa utilización en certo contexto; no noso caso deberemos sanear os datos para asegurarnos de que podemos empregarlos como parte dunha consulta SQL.

Por exemplo, a cadea que introduce o usuario como contrasinal no exemplo inicial, **' OR 1;** non é segura. Deberemos transformala para poder empregala como parte dunha consulta. Os caracteres que non poden ser admitidos nas consultas como parte dos valores introducidos polos usuarios son principalmente a comiña simple (**'**), a comiña dobre (**"**), o punto e coma (**;**) e a barra inversa (\), aínda que poden variar según o SXBD empregado. Normalmente cada extensión asociada a un SXBD prové unha función axustada aos seus requerimentos para sanear estes valores.

Na extensión MySQLi temos a función **mysqli_real_escape_string** (correspondente ao método real_escape_string da clase mysqli).

```
$db = new mysqli('localhost', 'usuario', 'abc123.', 'platega');
$nome = $db->real_escape_string($_REQUEST["nome"]);
$contrasinal = $db->real_escape_string($_REQUEST["contrasinal"]);
...
```

Na extensión PDO temos o método **quote**.

```
$db = new PDO("mysql:host=localhost;dbname=platega", "usuario", "abc123.");
$nome = $db->quote($_REQUEST["nome"]);
$contrasinal = $db->quote($_REQUEST["contrasinal"]);
...
```

Nalgúns casos pode suceder que a extensión do SXBD que empreguemos non implemente unha función ou método adecuado para sanear a información que introduza o usuario. Por exemplo, o controlador ODBC de PDO non o contempla. Unha alternativa é empregar un filtro de saneamento proporcionado pola extensión Filter da que falamos antes.

```
$nome = filter_var($_REQUEST["nome"], FILTER_SANITIZE_MAGIC_QUOTES);
$contrasinal = filter_var($_REQUEST["contrasinal"], FILTER_SANITIZE_MAGIC_QUOTES);
...
```

Empregar "listas brancas"

Cando os valores que introducen os usuarios deben limitarse a un grupo de opcións, é mellor asegurarse de que a súa entrada se corresponde con un dos valores válidos.

Por exemplo, se a táboa sobre a que imos facer a consulta pode ser, en función dun valor aportado polo usuario, "Empregados" ou "Clientes", poderíamos facer algo como o seguinte:

```
$taboas = array("empregados" => "Empregados",
                "persoal" => "Empregados",
                "clientes" => "Clientes",
                "persoas" => "Clientes",
                "por_defecto" => "Clientes");
if(isset($taboas[$_REQUEST["taboa"]])) $taboa = $taboas[$_REQUEST["taboa"]];
else $taboa = $taboas["por_defecto"];
```

Empregando o operador **?**, pode simplificarse a expresión condicional anterior.

```
$taboas = array("empregados" => "Empregados",
                "persoal" => "Empregados",
                "clientes" => "Clientes",
                "persoas" => "Clientes",
                "por_defecto" => "Clientes");
$taboa = $taboas[$_REQUEST["taboa"]] ?: $taboas["por_defecto"];
```

--Víctor Lourido 21:26 18 jul 2013 (CEST)