**PHP安全**

1. **所有用户的输入数据都是不可信的**

用户输入的数据必须使用合适的方式去验证其类型以及过滤;

对于字符串类型的数据应该使用addslashes()、trim()、strip\_tags()等字符串函数进行处理;

对于整型或浮点型数据应该使用intval()或floatval()函数进行处理

1. **SQL注入**

SQL注入是一种恶意攻击，用户利用在表单字段输入SQL语句的方式来影响正常的SQL执行。还有一种是通过system()或exec()命令注入的，它具有相同的SQL注入机制，但只针对shell命令。

$username = $\_POST['username'];  
 $query = "select \* from user where username = '".$username."'";  
 echo $query;  
 $db = new mysqli('localhost', 'demo', ‘demo', ‘demodemo');  
 $result = $db->query($query);  
 if ($result && $result->num\_rows) {  
 &nbsp;&nbsp;&nbsp; echo "<br />Logged in successfully";  
 } else {  
 &nbsp;&nbsp;&nbsp; echo "<br />Login failed";  
 }

上面的代码，在第一行没有过滤或转义用户输入的值($\_POST['username'])。因此查询可能会失败，甚至会损坏数据库，这要看$username是否包含变换你的SQL语句到别的东西上。

防止SQL注入，可以使用如下办法

使用mysql\_real\_escape\_string()过滤数据

手动检查每一数据是否为正确的数据类型

使用预处理语句并绑定变量

使用准备好的预处理语句

分离数据和SQL逻辑

**3.代码注入**

代码注入是利用计算机漏洞通过处理无效数据造成的。问题出在，当你不小心执行任意代码，通常通过文件包含。写得很糟糕的代码可以允许一个远程文件包含并执行。如许多PHP函数，如require可以包含URL或文件名，例如：

<form>Choose theme:  
 <select name = theme>  
 <option value = blue>Blue</option>  
 <option value = green>Green</option>  
 <option value = red>Red</option>  
 </select>  
 <input type = submit>  
 </form>  
 <?php  
 if($theme) {  
 require($theme.'.txt');  
 }  
 ?>

在上面的例子中，通过传递用户输入的一个文件名或文件名的一部分，来包含以"http://"开头的文件。

防止代码注入

过滤用户输入

在php.ini中设置禁用allow\_url\_fopen和allow\_url\_include。这将禁用require/include/fopen的远程文件。

其他的一般原则

　1. 不要依赖服务器配置来保护你的应用，特别是当你的web服务器/ PHP是由你的ISP管理，或者当你的网站可能迁移/部署到别处，未来再从别处迁移/部署在到其他地方。请在网站代码中嵌入带有安全意识的检查/逻辑(HTML、JavaScript、PHP,等等)。

　2. 设计服务器端的安全脚本:

　　—例如,使用单行执行 - 单点身份验证和数据清理

　　—例如,在所有的安全敏感页面嵌入一个PHP函数/文件，用来处理所有登录/安全性逻辑检查

　3. 确保你的代码更新，并打上最新补丁

**4.XSS攻击**

XSS(跨站点脚本攻击)是一种攻击，由用户输入一些数据到你的网站，其中包括客户端脚本(通常JavaScript)。如果你没有过滤就输出数据到另一个web页面，这个脚本将被执行。

为了防止XSS攻击，使用PHP的htmlspecialchars()等函数过滤再输出到浏览器。

****5.加密混淆javascript代码，提高攻击门槛****

    很多xss漏洞，都是黑客通过阅读javascript代码发现的，如果我们能把所有javascript代码混淆以及加密，让代码就算解密后也是混乱的（比如把所有变量名替换成其MD5 hash值），提高阅读的难度。

**6.文件上传**

上传漏洞是很致命的漏洞，只要存在任意文件上传漏洞，就能执行任意代码，拿到webshell。我们需要首先在客户端通过javascript验证用户选择的文件类型，但这只是最初提醒用户的部分，最终验证部分还是要在服务器端进行验证。

在上传文件$\_FILE数组中，会有一个MIME类型($\_FILE[‘userfile’][‘type’])，告诉服务端上传的文件类型是什么，但是它是不可靠的，是可以被修改的。在很多存在上传漏洞的网站中，都是只验证了MIME类型，而没有取文件名的后缀验证，导致上传任意文件。

  所以我们在文件上传类中完全可以忽略这个MIME类型，而只取文件名的后缀，如果在白名单中，才允许上传。

当然，服务器的解析漏洞也是很多上传漏洞的突破点，所以我们尽量把上传的文件重命名，以“日期时间+随机数+白名单中后缀”的方式对上传的文件进行重命名，避免因为解析漏洞而造成任意代码执行7

****7.限制用户权限，预防CSRF（跨站请求伪造）****

现在脚本漏洞比较火的就是越权行为，很多重要操作使用GET方式执行，或使用POST方式执行而没有核实执行者是否知情。

预防CSRF可以使用随机字符串token（由php生成，并保存在SESSION中），如果用户提交的这个随机字符串和SESSION中保存的字符串一致，才能进行下一步操作。

1. ****使用更高级的hash算法保存数据库中重要信息（如密码）****

在这个硬盘容量大增的时期，很多人拥有很大的彩虹表，再加上类似于cmd5这样的网站的大行其道，单纯的md5已经等同于无物，所以我们迫切的需要更高级的hash算法，来保存我们数据库中的密码。

所以后来出现了加salt的md5，比如discuz的密码就是加了salt。其实salt就是一个密码的“附加值”，比如A的密码是123456，而我们设置的salt是abc,这样保存到数据库的可能就是md5('123456abc')，增加了破解的难度。

但是黑客只要得知了该用户的salt也能跑md5跑出来。所以我们应该使用一个随机的salt存在数据库中，我们可以使用密码加截取md5(salt)前n位的方式进行加密，组成新的密码存在数据库中，如md5($userPwd.substr(md5($salt),0,8))

1. ****验证码安全性****

验证码通常是由php脚本生成的随机字符串，通过GD库的处理，制作成图片。真正的验证码字符串保存在SESSION中，然后把生成的图片展示给用户。用户填写了验证码提交后，在服务端上SESSION中的验证码进行比对。我们应该在留言板等功能的小程序中添加验证码，以过滤机器人自动提交，产生大量垃圾数据，浪费数据库空间。

另外表单验证中应该是前后端都需要验证，不应该只单方面的验证前端或者后端

1. ****把握整站的结构，避免泄露站点敏感目录****

**对于manager、upload等敏感目录应该做好权限控制，控制用户的访问**

1. ****配置好php.ini文件中的相关安全配置****

打开php的安全模式,safe\_mode = on;用户组安全,safe\_mode\_gid = off;

控制php脚本能访问的目录,open\_basedir = /usr/www等

**12.对url中的信息做好加密工作，不可泄漏出重要的数据**

**对需要传递中url中的重要参数使用urlencode()函数进行加密**