PHP代码审计

1、前言问题的存在

从代码级别上，也就是应用层次上考虑代码安全的话（也就是不考虑底层的语言本身等问题的漏洞），脚本安全问题就是函数和变量的问题。变量直接或者间接的接收用户不安全的输入，由于 php 本身的特性，在 php 中更容易发现这种变量的混乱（很多 php 程序都用来定义以及初始化以及接收变量，可以直接在程序中使用$id 这样的变量，初始化完全由 php 的设置来完成，如果稍不注意，就可能导致变量的混乱从而导致攻击）。

变量接收不安全的输入之后，没有做恰当的过滤又用在不同的地方，就可能造成不同的危害。如果直接进入数据库然后显示给用户就会导致跨站脚本攻击，如果用在 sql 语句中就可能导致 Sql 注射攻击，这几种攻击都是是与具体的脚本语言无关的，在各种脚本语言里都可能存在。由于 php 的变量很灵活，这些有害的变量如果用在一些逻辑语句中，就会导致关键代码的跳过如身份验证失败和跳过一些变量的初始化从而导致程序逻辑混乱而产生其他漏洞。如果这个变量用在了危险的函数如 include 等等当中，当然就会出现文件包含漏洞，出现在 fopen 函数里就会可能产生写文件的漏洞，出现在 mysql_query 函数中就是 Sql 注射漏洞，eval 以及 preg_replace 中可能导致代码的执行，出现在 htmlspecia 函数中可能导致出错而绝对路径泄露…… 变量出现的环境决定了它可能的危害。

思考了问题的存在，那么如何从代码级别上检查这种漏洞呢？当然熟悉熟悉

php 语言是最基本的，也应该是抓住函数和变量，危险的函数里如果有变量那么请确定这个变量的来源，是否正确的初始化，初始化之后是否能被用户注入敏感字符，在进入函数前这些敏感的字符是否得到了彻底的清除。对于代码审核工作的难点可能就在于对变量来源的确定，这需要对 php 特 性以及你所审核的代码的熟悉，但也并不是所有的变量的来源都清晰可见，可能一些初始化的代码并没有像想象中运行，一些变量里的东西可能也来自于你并不想他 来的地方，还有一些变量可能来自于数据库或者系统的配置文件 ,但是很可能数据库和配置文件在之前就已经被修改，或者在后面不安全的操作了这些变量，这些变 量也是不可相信的。下面我们就按照变量与函数的思路来思考脚本代码的安全。

2、变量来自哪里？

1 显示的输入

又叫叫变量来自哪里其实也就是说威胁来自哪里 ,只是从 web 上考虑的话，什么样的网站最安全？很明显，那些只提供静态 Html 页面的网站是最安全的，因为这样的网站不与浏览者进行任何交互，就好比打劫一个密不透风的银行，很难实现，但是对于一个大的论坛或者脚本程序就不一样了，你登陆的时候需要传递用户名和密码这些变量给服务器，甚至包括你登陆的 Ip 与浏览器等等都是程序抓取的对象 ,抓取一次与服务器交互的过程如发表帖子等等你就发现浏览器与服务器之间进行的数据传输 ,你可能看得见的包括提交的表单 ,地址栏参数等等，你看不见的包括 Cookie，Http 头都是提交数据也就是变量的地方。这些地方也是服务器处理数据最原始的入口。那么 php 程序是如何接受变量的呢？所有提交的变量都被 php 保存在了一些数组里，包括

$_GET

$_POST

$_COOKIE

$_FILES

$_SERVER

为了最初的方便与灵活，在 php 的设置里有这么个选项

register_globals

当这个选项为 on 的时候，上面出现的那些变量都会成为$GLOBALS 中的一员，在脚本中都不需要再取得就可以直接使用，并且以

variables_order

的顺序覆盖。很多程序考虑到了 register_globals 为 off 的情况，于是在程序初始化的时候使用如下的代码：

@extract(daddslashes($_POST));

@extract(daddslashes($_GET));

这些代码起到了 register_globals 的作用，作用也是将 POST 和 GET 的内容释放出去做为全局变量，但是危险可能更大，后面会提到。

2 隐式的输入

上面这些是最原始的，没有经过程序转换的数据，程序很多地方用到的变量都来自这里，但也不表示其他的地方没有变量传递过来，下面有一个数据传递的模式：

用户传递的数据==========>数据库==========>

程序代码处理=======>程序代码

这 个模式的意思是用户的输入可能先进入了数据库，然后程序从数据库再取得这个输入送入某些危险的函数执行，一般的程序员都会有一个意识认为从数据库中取得的 变量是安全的，但是事实并不如此，只要某些敏感字符最终送入到程序代码中，不管他中间停留在什么地方，都是危险的。与存储在数据库中类似的情况是，一些程 序把用户的输入放到文件中，如缓存文件，然后在必要的时候从里面取得，如果太过相信这些地方来的变量，这样还是会导致问题的。

3. 变量覆盖

还有很多的时候，程序收到的变量很可能来自他不应该来的地方，譬如 Dz 的代码：

```
$magic_quotes_gpc = get_magic_quotes_gpc();
@extract(daddslashes($_POST));
@extract(daddslashes($_GET));
if(!$magic_quotes_gpc) {
$_FILES = daddslashes($_FILES);
}
```

这样之后，你还觉得$_FILES 是原来的$_FILES 了么？如果我们建立一个 _FILES 的表单或者干脆在 url 里加上 php?_FILES[]=ddddd ，这样之后 $_FILES 已经完全被覆盖了，然后你代码里引用的$_FILES 就不是原来的了，在 Dz 以前的版本中曾经出现过这个问题。这应该属于变量覆盖的问题，把初始化的那个文件放大来看看吧：

```
$magic_quotes_gpc = get_magic_quotes_gpc();
@extract(daddslashes($_POST));
```

```
@extract(daddslashes($_GET));
if(!$magic_quotes_gpc) {
$_FILES = daddslashes($_FILES);
}
$charset = $dbcharset = '';
$plugins = $hooks = array();
require_once DISCUZ_ROOT.'./config.inc.php';
require_once DISCUZ_ROOT.'./include/db_'.$database.'.class.php';
if($attackevasive) {
require_once DISCUZ_ROOT.'./include/security.inc.php';
}
```

这样貌似是没有问题的，但是满足一定的条件的话还是可能出问题，假设register_globals为on 的话，我们进入全局的变量不只是$_GET 和$_POST 吧！包括$_COOKIE 和$_FILES以及$_SERVER 都是会在全局数组中产生变量的，通过上面的语句， 我们提交一个php?_SERVER[PHP_SELF]就可以覆盖掉_SERVER 数组，那么整个程序中的$_SERVER 数组

都是不可以相信的了。我也见过这样写的代码：

```
.....
require_once ROOT_PATH.'inc/database_config.php';
require_once ROOT_PATH.'inc/dv_spacemain.php';
if(PHP_VERSION < '4.1.0') {
$_GET = &$HTTP_GET_VARS;
```

```
$_POST = &$HTTP_POST_VARS;
$_COOKIE = &$HTTP_COOKIE_VARS;
$_SERVER = &$HTTP_SERVER_VARS;
$_ENV = &$HTTP_ENV_VARS;
$_FILES = &$HTTP_POST_FILES;
$_SESSION =& $HTTP_SESSION_VARS;
}
$magic_quotes_gpc = get_magic_quotes_gpc();
$register_globals = @ini_get('register_globals');
if(!$register_globals || !$magic_quotes_gpc) {
@extract(i_addslashes($_POST));
@extract(i_addslashes($_GET));
@extract(i_addslashes($_COOKIE));
if(!$magic_quotes_gpc) {
$_FILES = i_addslashes($_FILES);
}
}
.....
```

同样是在系统初始化的地方，但是变量的释放是在require_once ROOT_PATH.'inc/general_funcs.php';require_once ROOT_PATH.'inc/dv_spacemain.php';

这些关键变量初始化之后，那么我们完全可以提交一个?$host=xxx.xxx.xxx.xxx 这样的东西覆盖掉系统自己的数据库初始化文件里的数据库地址变量，然后就可以.....

4. 变量感染

这个很容易理解，当一个变量不安全的时候，与之有关的赋值等操作都是不安全的，譬如：

$id = $_GET[id];

.....

$articleid = $id;

实际过程中可能没有这么明显，但是结果是一样的，只要某个变量把敏感字符带入不该带的地方，那么就会产生威胁，不只是变量，不安全的函数会让使用这个函数的所有代码都变的不安全。

3、哪里是不安全的

变量最终是要代码处理的，代码最终是要依靠一些系统的函数和语句执行的，不正确的变量出现在危险的函数里，那么恭喜你，漏洞出现了！

1. Sql 注射漏洞

按照我们的理解，就是Sql 函数里出现了不安全的变量，在php 中执行这样的语句在系统中是很多的，在Dz 的初始化文件中有如下代码：

```
if($sid) {
if($discuz_uid) {
$query = $db->query("SELECT s.sid, s.styleid, s.groupid='6' AS ipbanned,
```

```
s.pageviews AS spageviews, s.lastolupdate, s.seccode, m.uid AS discuz_u
id,
m.username AS discuz_user, m.password AS discuz_pw, m.secques
AS discuz_secques, m.adminid, m.groupid, m.groupexpiry,
m.extgroupids, m.email, m.timeoffset, m.tpp, m.ppp, m.posts,
m.digestposts, m.oltime, m.pageviews, m.credits, m.extcredits1, m.extcr
edits2, m.extcredits3,
m.extcredits4, m.extcredits5, m.extcredits6, m.extcredits7,
m.extcredits8, m.timeformat, m.dateformat, m.pmsound,
m.sigstatus, m.invisible, m.lastvisit, m.lastactivity, m.lastpost,
m.newpm, m.accessmasks,m.xspacestatus, m.editormode, m.customsho
w
FROM {$tablepre}sessions s, {$tablepre}members m
WHERE m.uid=s.uid AND s.sid='$sid' AND
CONCAT_WS('.',s.ip1,s.ip2,s.ip3,s.ip4)='$onlineip' AND m.uid='$discuz_ui
d'
AND m.password='$discuz_pw' AND
m.secques='$discuz_secques'");
} else {
$query = $db->query("SELECT sid, uid AS sessionuid, groupid, groupid='
6'
AS ipbanned, pageviews AS spageviews, styleid, lastolupdate, seccode
FROM {$tablepre}sessions WHERE sid='$sid' AND
CONCAT_WS('.',ip1,ip2,ip3,ip4)='$onlineip'");
```

找下$db->query 函数中的美圆符号吧，呵呵，发现有好几个，也就是说可能存在漏洞了，注意，说的是可能，因为现在的代码安全性都有提高，找个没人管的变量不容易啊，一般的变量都会正确的初试化，但是跟踪$onlineip 变量就可以发现这个变量基本没有管的，因为这个是从Http 头里提取的，一般的人不怎么

会注意这个，但是偏偏问题出现了:）需要说明的是，函数产生漏洞，而不管语句是什么，所以只要是update 或者insert 或者是select 里出现了变量，该变量是我们可以控制的，并且改变量能突破程序的一些限制（什么限制我们后面会讲到）从而控制这个sql 语句的执行，那么我们的漏洞就是成立的，能不能利用就要看具体环境了。

2. Xss 跨站脚本攻击漏洞

这个漏洞其根本就是客户端的Html 注射漏洞，如果用户提交变量里含有<>或者能被解释成Html 的字符被送到数据库，然后再从数据库输出到浏览者的浏览器，那么就可能存在Xss 注射漏洞。<>字符能导致跨站脚本攻击很好理解，但是要注意的是不需要提交<>一样会有这种问题，这是很多人所误解的。假设我们的输入如一个url最终是放在一个Html 标签之内的，这样的情况很多，因为用户的头像什么的就必须这样的形式：

```
echo "<img src=\"$url\">"
```

我们控制了img 的属性从也一样实现了跨站脚本攻击。

3. 文件包含漏洞

这个主要是因为文件包含函数include 与require 等函数的参数中没有做好限制，导致用户能指定需要包含的文件如如下的代码：

```
require_once ROOT_PATH."cache/style/$cssname.php";
```

如果我们能控制$cssname 就可以控制需要包含的内容，漏洞的存在就取决于这个变量可不可以控制了，如果可以控制又可以用到../跳转的话，那么.....：）至于危害，也就是任意代码执行和目录遍历了。

4. 直接写入webshell 漏洞

这在一些文本数据库中见的很多，一些程序使用文本文件做为数据库，于是不可避免的要用到如fopen，fread 以及fwrite 这些函数，打开fopen 函数的帮助看看：

resource fopen (string filename, string mode [, bool use_include_path [, resource zcontext]])从这个帮助可以知道如果fopen 函数的第一个参数可以控制就可能打开一些不应该被打开的文件 ,其他的文件函数都是例似的 ,再次重申，变量的环境决定了它的价值：)

5. 代码执行漏洞

能产生这种漏洞的一定要有这种功能的函数，常见的有eval 函数和preg_replace 函数，如果eval 里出现了我们能控制的语句那么会产生问题，在preg_replace函数的第二个参数可能导致任意代码执行的问题！

6. 绝对路径泄露

这主要是由于php 的报错造成的 ,使用一个不存在的文件 ,mysql 查询出错，提交一个不符合类型的参数给一些函数都会导致这个问题 ,一般的程序都对函数的错误用了@抑制，但是还是存在一些爆路径的方法！有人说一个路径不能代表什么，其实一个路径可以知道操作系统，知道路径，可能知道虚拟主机的配置等等信息。

7. 逻辑混乱

如果一个变量用在了if 等逻辑语句中，那么很可能导致逻辑混乱问题，跳过一些语句的执行等等。

.....种种其他的函数出现的问题。

3、过滤与饶过过滤

很多人已经注意到了这些安全问题，php 中也包含了自己的安全机制，那就是GPC=On的时候加入了对'和"以及\的转义，很多的程序也都自己加入了这些转义。其他脚本语言可能没有这种机制，但是这种思路非常好。在被转义的情况下，他可以保证用户所有的输入都是字符串，无论这个变量进入哪里，包括sql 查询，跨站脚本等等，但是有个很重要的前提就是你所书写的程序必须也把所有的输入都当作字符串来对待。很明显，如果有以下的 2 个语句：

```
$query = $db->query("select * from user where uid=$id");
$query = $db->query("select * from user where uid='$id'");
```

哪个更安全一点呢？第一个语句做的假设是$id 是数字类型变量，第二个假设输入是字符类型变量所以使用"引用他。如果提交php?id=1 and 1=2 对于第一个语句，最后变成：

```
select * from user where uid=1 and 1=2 //and 1=2 成为了一个表达式
select * from user where uid='1 and 1=2' //and 1=2 还是字符串的一部分
```

希望这个能给你一点提示，但不仅仅是sql 注入！如果是跨站脚本漏洞的话，我们一样可以使用上面的思路，无论用户输入什么，在过滤好<>，然后只要把用户的输入当作一个字符串就行了，但是要注意，光一个转义字符\'在html 里很可能还是有'的意义，你需要去掉他在html 里的转义，用htmlspecialchars()函数，当然也要配合程序安全的代码，还是给两个例子：

```
echo "<img src=htmlspecialchars($url)>";
echo "<img src='htmlspecialchars($url)'>";
```

哪个是没有缺陷的呢？

上 面的例子是在暗示，一个变量如果只能作为字符串，就是不要赋予它特定的含义，它是做不了什么的。那么你可能已经想到了，对于上面提到的种种漏

洞，如果迫不 得已需要用户控制变量，那么你只要过滤掉特殊意义的字符，然后把输入只是作为一个字符串，那么就可以从代码级别上杜绝这些漏洞了。譬如对于fopen 函数，只要我们过滤掉../以及..\这些字符，然后将用户输入的"，'以及空字符等转义就没有问题了，对于写webshell 的问题可以通过如果是建立数据库而已就可以转义掉<>脚本标记字符，如果产生的本身就是php 文件那么可以将用户的输入限制在''或者在""之内，一切只是字符，跟跨站脚本的防御很相似吧！还有一些敏感字符可能是程序自己产生的，如下的代码：

$deldb=explode("|",$imgdb['icon']);

就赋予了$imgdb['icon']神圣的意义，所以我们在处理的时候一定先要将$imgdb['icon'] 里的|清干净！

5、其他的问题

我上面所说的仅仅是在检测漏洞与修补漏洞时的一些想法， 从函数着手分析变量的处理，但更多的时候程序的安全还是要依靠程序员清晰的思路的，但安全不仅仅是这样，譬如GBK 和Big5 的编码问题，还有上传等问题。Php 脚本是这样，其他的脚本也是这样。